
Ph.D. COURSE IN: Information Engineering
SERIES: XXXV

Enhancing Scalability of Deep Learning Based Approaches in Semiconductor Manufacturing

Thesis written with the financial contribution of Infineon Technologies AG.

Coordinator: Prof. Andrea Neviani
Supervisor: Prof. Alessandro Beghi
Co-Supervisor: Ch.mo Prof. Gian Antonio Susto

Ph.D. student : Natalie Gentner

Università degli Studi di Padova

Department of Information Engineering

XXXV Series

Ph.D. School in Information Engineering

University of Padua 2022

Enhancing Scalability of Deep Learning Based Approaches in Semiconductor Manufacturing

Supervisor
Prof. Alessandro Beghi

Co-supervisor
Prof. Gian Antonio Susto

Director
Prof. Gaudenzio Meneghesso

Coordinator
Prof. Andrea Neviani

Ph.D. Candidate
Natalie Gentner

I don't have any particular recipe [for developing new proofs]. ... It is like being lost in a jungle and trying to use all the knowledge that you can gather to come up with some new tricks, and with some luck you might find a way out.

MARYAM MIRZAKHANI

Abstract

Semiconductor manufacturing is determined by complex processes and high variety of process components. In order to assure high product quality, processes need to be as stable and reproducible as possible and standardization is one of the key factors for successful process control. In addition, Machine Learning (ML) based technologies for production, like Virtual metrology/soft sensing, defect classification, predictive maintenance and fault detection, have been successfully applied in the past recent years in data intensive manufacturing industries, like semiconductor manufacturing, to support automation and improve process monitoring and related operations.

Reproducibility, generalization, standardization and alignment over multiple equipment is a key element to ensure industry wide adoption, implementation and scalability for ML-based technologies in complex production environment.

To deal with such generalization and scalability tasks in the production context, semiconductor related literature mainly presents two approaches: matching and transfer learning with a focus especially on adaptation.

(I) MATCHING The matching approach aims at analyzing, finding and eliminating existing differences in production to avoid future quality issues, machine breakdowns and unknown misfits with help of expert knowledge and data. This results in identical properties and settings hence overall standardization and consequently identical results for targeted equipment and processes.

(II) ADAPTATION The adaptation approach as part of transfer learning is based on domain adaptation theory and as such purely data-driven; domains respective data distributions are taken into account and dedicated methods are developed to deal with scenarios where different data distributions occur. In this area, neural networks are a particularly appealing method: so-called domain adversarial neural network (DANN). They are designed to deal with occurring data differences; therefore they are suitable for modeling tasks that include process or equipment with identical underlying physical information but naturally data shifts occur due to for example degradation or maintenance besides others.

The methods presented in the thesis - DANN-based Alignment Model (DBAM) and its extended version DANN-based Alignment with Cyclic Supervision (DBACS) - are able to learn a model that is scalable from one domain to another in settings where inconsistent data distributions occur. The underlying idea of the proposed modeling approach - compared to already existing approaches - is to keep the necessary information accessible to allow physical

interpretability and comparison of original versus aligned data while maintaining the high accuracy of a dedicated prediction or classification model. In addition, its functionality handles unpaired mapping, heterogeneous domain adaptation, comparability of source a target in bijective manner meaning in both directions hence enabling matching on top of adaptation.

Besides semiconductor and use case specific preprocessing including recipe removal with sensor data or text removal for images, we exploit adversarial training and model structures inspired by domain adversarial neural networks (DANN) combined with a domain adaptation alignment approach using a residual inspired setting as for generative adversarial networks (GAN). DBAM consists of 3 parts: (i) the baseline or reference prediction or classification model P, (ii) an alignment model called aligner A used to map the target domain to the source domain (the output of the aligner is called aligned) and (iii) a domain discriminator D for classification of the domains and that necessary for the adversarial training approach that is especially useful for semi-supervised or unsupervised settings beside the supervised one.
DBAM can be set up with different neural network architectures depending on the data type at hand hence it is applicable to a broad range of application fields. Since the aligner part is set up in style of an encoder-decoder network, it can be used for source and target domain with different dimensions respective features available. The extended version DBACS has additional aligner and discriminator models inspired by CycleGAN. Hence it is suitable for unpaired mappings and enables a bijective transformation between source and target domain. For further improvement specific loss functions like Wasserstein loss, structural similarity index measure (SSIM) and feature matching loss are applied for the training respective alignment process. All those methods are well known and proven very useful in the context of generative adversarial networks (GAN) in the field of computer vision and are transferred to semiconductor context involving different data types like stationary features, times series data and images.

Three different use cases are presented in the thesis and DBAM respective its extended version DBACS are tested against state-of-the-art methods presented in relevant literature: Virtual metrology (VM), predictive maintenance (PdM) and defect classification (DC).

Virtual metrology/ soft sensing (VM) estimations are typically monitored in process control alongside real metrology measures to widen the monitoring perspective of process results. VM is formalized as a regression task where sensor data collected from an equipment during a process is treated as input, while the corresponding inline measurements on the product itself are used as output. In detail VM aims at estimating metrology quantities that are costly to be measured by means of data-driven algorithms and exploiting the availability of historical data where such measures are performed.
The first VM use case propose a common VM model for two identical-in-design tools whose

process data follow different distributions; therefore, DBAM is working directly on the available raw sensor data in form of time series. Different kind of mismatch can be visualized like overall horizontal shifts as well as partly different functional behavior between sensors of the analyzed equipment.
The second VM use case presents a VM prediction model for one process running on two different equipment types resulting in different data representations respective different input dimensions for source and target domain while the underlying physical information is identical. Therefore, DBACS is applied to enable heterogeneous domain adaptation meaning unpaired mapping and bijective transformations between source and target. Equipment matching on top of adaptation is further discussed and evaluated.

Predictive maintenance (PdM) in general, but especially Remaining Useful Life (RUL) prediction for avoidance of uncontrolled machine breakdowns hence product losses and repair costs plays a major and critical role in automation of manufacturing. Relevant literature heavily discusses the scalability issue of PdM models when it comes to different equipment or failure causes. Here we explore a scenario where identical equipment can be split in two groups based on their process behavior. Specific preprocessing like recipe effect removal and feature selection besides others are analyzed. Since a common model cannot be successfully deployed, a dedicated prediction model is trained on equipment data from group 1 and both DBAM as well as the DBACS are applied to align data from group 2 to enable the usage of the dedicated model for all involved equipment. The interpretability properties of the model are exploited and relevant features as well as equipment group behavior especially towards the end of a life before and after the alignment and for the two groups are compared. We close with a discussion on imperfect data and its influence on RUL model performance and generalization.

Defect classification (DC) is a necessary and reoccurring part for quality assurance in semiconductor manufacturing. Before the rise of AI those inspections were done manually causing time delay, high costs and were prone to human errors. The main goals are increased productivity, speed up and creation of additional value by involvement of un- or under used data. Great results on dedicated models with a high accuracy are achieved especially by applying transfer learning to already existing pretrained networks like EfficientNet. Nevertheless, the scaling of those methods is still under research due to high complexity of the data; e.g. same defects can occur at different stages of the process. We observe a decrease in accuracy for defects occurring on such different layers resulting in different image backgrounds. Therefore, DBACS is applied to align images with complex structural backgrounds (for example horizontal line or squares) in order to neutralize their background and make them usable in an already existing classifier trained on images with steady grey background. The model abilities for un- and semi-supervised training are tested and evaluated.

Semiconductor manufacturing is one of the most exciting field for Machine Learning-based solutions, with a broad range of methodological challenges that needs to be faced: the scalability of such solutions is, in our view, one of the most important aspects to be tackled in this context. We believe that our work, by leveraging the flexibility of modern deep learning architectures and by considering a holistic set of tasks and applications, represents an important contribution in this area that we believe will be one of the most important topic in the industry in the near future.

Sommario

La manifattura di semiconduttori è caratterizzata da processi complessi e da un'elevata varietà di prodotti. Per garantire un'elevata qualità, i processi devono essere il più possibile stabili e riproducibili e la standardizzazione è uno dei fattori chiave per un'efficace controllo dei processi. Inoltre, le tecnologie basate sull'apprendimento automatico/Machine Learning (ML) per la manifattura, come la Virtual Metrology/Soft Sensing, la Classificazione di Difetti, la Manutenzione Predittiva e il Rilevamento di Guasti, sono state applicate con successo negli ultimi anni nelle industrie manifatturiere ad alta intensità di dati, come la produzione di semiconduttori, per migliorare monitoraggio dei processi e delle relative operazioni.

Riproducibilità, generalizzazione, standardizzazione ed allineamento su più macchine sono elementi chiave per garantire l'adozione e la scalabilità delle le tecnologie basate sul ML in manifatture complesse.

Per affrontare tali problemi di generalizzazione e scalabilità nel contesto della produzione, la letteratura relativa alla manifattura di semiconduttori presenta principalmente due approcci: il matching e il transfer learning.

(i) Matching Il matching mira ad analizzare, trovare ed eliminare le differenze esistenti nella produzione per evitare futuri problemi di qualità, guasti alle macchine e disallineamenti sconosciuti con l'aiuto di dati e conoscenza di dominio. Ciò si traduce in proprietà e impostazioni identiche, quindi standardizzazione generale e di conseguenza risultati identici per apparecchiature e processi mirati.

(ii) Adaptation L'approccio dell'adaptation come parte del transfer learning è basato sulla teoria dell'adattamento di dominio e come tale puramente basato sui dati; vengono presi in considerazione i rispettivi domini delle distribuzioni di dati e vengono sviluppati metodi dedicati per affrontare scenari in cui si verificano distribuzioni di dati diverse. In quest'area, le reti neurali sono approcci particolarmente interessanti, fra cui citiamo in particolare l'architettura Domain-Adversarial Neural Network (DANN). Le DANN sono progettate per attività di modellazione che includono processi o apparecchiature con caratteristiche nominali identiche ma che differiscono in realtà (si pensi, ad esempio, al fenomeni di degrado legati all'utilizzo o all'effetto di manutenzioni).

I metodi presentati nella tesi - il modello di allineamento basato su DANN chiamato *DANN-based Alignment Model* (DBAM) e la sua versione estesa *DANN-based Alignment with Cyclic Supervision* (DBACS) - sono in grado di apprendere un modello scalabile da un dominio all'altro in contesti in cui si verificano distribuzioni di dati incoerenti. L'idea alla

base dell'approccio di modellazione proposto, rispetto agli approcci già esistenti, é di mantenere le informazioni necessarie accessibili per consentire l'interpretazione fisica e il confronto dei dati originali rispetto a quelli allineati, pur mantenendo l'elevata precisione di un modello di previsione o classificazione dedicato. Inoltre, la sua funzionalitá gestisce la mappatura non accoppiata, la comparabilitá di origine e destinazione in modo biunivoco, consentendo quindi la corrispondenza oltre all'adattamento.

Negli approcci proposti in questo lavoro, oltre a pre-elaborazioni dei dati specifiche per il mondo della manifattura dei semiconduttori (ad esempio la rimozione delle ricette dal dato di sensoristica), sfruttiamo le domain-adversarial neural network (DANN) combinate con un approccio di allineamento dei dati che sfrutta le generative adversarial neural network (GAN). L'approccio proposto (DBAM) è costituito da 3 parti: (i) la previsione o il modello di classificazione di riferimento P, (ii) un modello di allineamento chiamato allineatore A utilizzato per mappare il dominio di destinazione sul dominio di origine e (iii) un discriminatore di dominio D per la classificazione dei domini e quello necessario per l'approccio formativo contraddittorio che è particolarmente utile per i contesti semi-supervisionati o non supervisionati oltre a quello supervisionato.

Il framework DBAM può essere utilizzato con diverse architetture di reti neurali a seconda del tipo di dati in oggetto ed quindi è applicabile ad un'ampia gamma di campi applicativi. Poiché l'aligner è configurato nello stile di una rete codificatore-decodificatore, può essere utilizzata per il dominio di origine e di destinazione con diverse dimensioni e rispettive caratteristiche disponibili. La versione estesa DBACS ha ulteriori modelli di allineatori e discriminatori ispirati a CycleGAN; DBACS è quindi adatto a mappature non accoppiate (dove origine e destinazione hanno diverse dimensionalitá) e consente una trasformazione biunivoca tra il dominio di origine e quello di destinazione. Per un ulteriore miglioramento, funzioni di loss specifiche - ad esempio la Wasserstein loss, lo structural similarity index measure (SSIM) e la Feature Metrix loss - vengono applicate per il rispettivo processo di allineamento dell'addestramento. Tutti questi metodi sono ben noti e si sono dimostrati molto utili nel contesto delle GAN nel campo della visione artificiale e sono in questo lavoro trasferiti al contesto della manifattura dei semiconduttori che coinvolgono diversi tipologie di dati (ad esempio tabulari, serie temporali ed immagini).

Nella tesi vengono presentati tre diversi casi d'uso per DBAM e per la sua versione estesa DBACS che rappresentano avanzamenti rispetto allo stato dell'arte scientifico in tema di soluzioni di Machine Learning per la manifattura di semiconduttori: la Virtual Metrology (VM), manutenzione predittiva (PdM) e classificazione dei difetti (DC).

Le stime di Virtual Metrology/soft sensing (VM) sono generalmente monitorate nel controllo di processo insieme a misure di metrologia reale per ampliare il monitoraggio della qualitá dei processi. La VM è formalizzata come un task di regressione in cui i dati dei sensori raccolti da un'apparecchiatura durante un processo vengono trattati come input, mentre le

corrispondenti misurazioni in linea sul prodotto stesso vengono utilizzate come output. In dettaglio, la VM mira, grazie alla disponibilitá di algoritmi di Machine Learning e storico dati, a stimare grandezze di metrologia costose.
Il primo caso d'uso considerato di VM propone un modello comune per due strumenti nominalmente identici i cui dati di processo seguono distribuzioni diverse; in questo primo lavoro, il framework DBAM viene applicato direttamente sui dati grezzi disponibili del sensore sotto forma di serie temporali. Un secondo caso d'uso di VM presenta invece un modello di previsione per un processo eseguito su due diverse apparecchiature la cui sensoristica genera dati di ingresso al modello con diversa dimensiolitá, mentre il processo fisico sottostante é il medesimo. L'approccio DBACS viene applicato per abilitare trasformazioni biiettive tra origine e destinazione.

La Manutenzione predittiva (PdM) e la sua formalizzazione piú diffusa, la Remaining Useful Life (RUL), svolgono un ruolo fondamentale nello smart monitoring industriale per evitare rotture inattese nelle macchine, con conseguenti scarti e costi di riparazione svolgono un ruolo importante e critico nell'automazione della produzione. In letteratura, il problema della scalabilitá dei modelli di PdM é ampiamente dibattuto quando si considerano diverse macchine o diverse cause di guasto. In questo lavoro, viene esplorato lo scenario in cui apparecchiature identiche possono essere suddivise in due gruppi in base al comportamento del processo. Considerando diverse macchine di processo, tipicamente un modello comune di PdM non puó essere efficace: nel nostro approccio, DBAM E DBACS vengono addestrati sui dati delle apparecchiature del gruppo 1 ed utilizzati per allineare i dati del gruppo 2 ed ottenere un modello dedicato ed accurato per entrambe le macchine coinvolte. Nell'approccio vengono sfruttate le proprietà di interpretabilità del modello e vengono confrontate le caratteristiche rilevanti e il comportamento del gruppo di apparecchiature, soprattutto verso la fine della vita prima e dopo l'allineamento e per i due gruppi. Nella tesi viene anche riportata una discussione su dati imperfetti e la loro influenza sulle prestazioni e sulla generalizzazione del modello RUL.

La classificazione dei difetti (DC) è una parte necessaria e ricorrente per la garanzia della qualità nella produzione di semiconduttori. Prima dell'ascesa dell'Intelligenza Artificiale e della Computer Vision, tali ispezioni venivano eseguite manualmente causando ritardi, costi elevati ed erano soggette a errori umani. Gli obiettivi principali del nostro lavoro in questo contesto sono l'aumento della produttività, dell'accuratezza e la creazione di valore aggiuntivo attraverso il coinvolgimento di dati non utilizzati o sottoutilizzati. Grandi risultati su modelli dedicati con un'elevata precisione si ottengono soprattutto applicando il transfer learning a reti giá esistenti pre-addestrate. Tuttavia, il ridimensionamento di tali metodi è ancora oggetto di ricerca a causa dell'elevata complessità dei dati; per esempio, gli stessi difetti possono verificarsi in diverse fasi del processo, portando ad una diminuzione della precisione per la classificazione, a causa dei diversi sfondi in immagini ottenute in fasi di processo diverse.

DBACS viene applicato per allineare immagini con sfondi strutturali complessi (ad esempio linee orizzontali o quadrati) con il fine di neutralizzarne lo sfondo e renderli utilizzabili in un classificatore già esistente addestrato su immagini con sfondo grigio costante. Le abilitá del modello vengono testate e valutate in questo lavoro sia in un set-up non supervisionata che in uno semi-supervisionata.

La manifattura di semiconduttori é uno dei campi piú interessanti per le soluzioni basate sull'apprendimento automatico, con un'ampia gamma di sfide metodologiche che possono essere incontrate in questo ambito: la scalabilità di tali soluzioni é, a nostro avviso, uno degli aspetti piú importanti da affrontare in questo contesto. Riteniamo che il nostro lavoro, sfruttando la flessibilitá delle moderne architetture di deep learning e considerando un insieme olistico di task ed applicazioni, rappresenti un contributo importante in quest'area che riteniamo sará uno degli argomenti piú rilevanti del settore nel prossimo futuro.

Contents

Listing of figures

Listing of tables

Listing of acronyms

1DCNN	One Dimensional Convolutional Neural Network
2DCNN	Two Dimensional Convolutional Neural Network
AdaBN	Adaptive Batch Normalization
AI	Artificial Intelligence
ANN	Artificial Neural Network
APC	Advanced Process Control
CA	Correlation Analysis
CCA	Canonical Correlation Analysis
CD	Critical Dimension
CNN	Convolutional Neural Network
CORAL	Correlation Alignment
DA	Domain Adaptation
DANN	Domain Adversarial Neural Network
DBACS	DANN-based Alignment with Cyclic Supervision
DBAM	DANN-based Alignment Model
DC	Defect Classification
DD	Defect Density
DL	Deep Learning
EM	Equipment Matching
EV	Explained Variance (regression score)
FDC	Fault Detection and Classification

FNN	Fully-connected Neural Network
FID	Frechet Inception Distance
GAN	Generative Adversarial Network
GTB	Gradient Tree Boosting
IQR	Interquantile Range
KL	Kullback-Leibler
KN	Keynumbers
KS	Kolmogorov–Smirnov
LDM	Linear Discrepancy Minimization
LSTM	Long Short Term Memory
LT	Layer Thickness
MAE	Mean Absolute Error
MCCV	Monte Carlo Cross Validation
ME	Maximal Residual Error
ML	Machine Learning
MLE	Maximum Likelihood Estimation
MLP	Multilayer Perceptron
MMD	Maximum Mean Discrepancy
MSE	Mean Squared Error
MVL	Multi-View Learning
NNW	Nearest Neighbors Weighting
OES	Optical Emission Spectroscopy
OLS	Ordinary Least Squares
PCA	Principle Component Analysis

PdM	Predictive Maintenance
R2	R-square (coefficient of determination)
RKHS	Reproducing Kernel Hilbert Space
RL	Reinforcement Learning
RNN	Recurrent Neural Network
RTC	Resource Time Classification
SA	Subspace Alignment
SSL	Semi-Supervised Learning
SL	Supervised Learning
SPC	Statistical Process Control
TCA	Transfer Component Analysis
TCN	Temporal Convolutional Network
TL	Transfer Learning
TrAdaBoostR2	Transfer AdaBoost for Regression
t-SNE	T-distributed Stochastic Neighbor Embedding
UDA	Unsupervised Domain Adaptation
UL	Unsupervised Learning
VM	Virtual Metrology

Part I

Introduction

1 Motivation and Thesis Organization

This chapter gives the motivation for the presented work as well as the organization and structure of the book at hand. It is an enriched consolidation of already published work ([70], [69], see Chapter 4 and first part of Chapter 8), manuscripts that are submitted for publication but are not yet published (see extension of first part of Chapter 8, Chapter 5, second part of Chapter 8 and Chapter 10) and unpublished studies (see Chapter 9).

Infineon Technologies AG is gratefully acknowledged for the financial support of this research in form of an industrial PhD position and further cooperation.

1.1 Motivation

Process control and monitoring are standard components of any automated production environment. *Virtual metrology (VM)* also known as soft sensor for example is based on a statistical model that predict wafer inline properties based on process information and sensor measurements. VM together with *predictive maintenance (PDM)* and *defect classification (DC)* besides others, is not only a key mechanism for direct/early fault detection but also an enabler for quality - by increasing the monitoring capacity -, for control - for example in combination with a run-to-run system that facilitates real time process corrections - and for smart capacity usage - by avoidance of unscheduled downs, preparing the input for smart sampling strategies and improved decision making. All those production control mechanisms including control systems like *advanced process control (APC)*, *run-to-run (R2R)* and *metrology* combined with *statistical process control (SPC)* are key factors in (semiconductor) manufacturing to monitor and assure quality, speed and stability in a highly automated and increasingly digitized production environment.

Thanks to the availability of structured information, big data and new technologies to build powerful infrastructures, Machine Learning (ML)-based solutions are becoming pervasive in (semiconductor) manufacturing to further improve equipment and operations efficiency as well as process control, to increase production performance [149] and bring manufacturing yield to the next level.

A broad range of machine and especially deep learning methods are already successfully implemented, trained and made usable for a whole bunch of different data types. In the realm of ML-based applications for production and related application focused research there are mentioned technologies like VM [182, 219, 183], PdM [192] and DC [4]. In addition a lot of effort and research is present for *anomaly detection (AD)* [106, 25, 200], *equipment health factor* or equipment monitoring (EHF) [21] and *fault detection and classification (FDC)* [59] in general. R2R and ML shows a limited amount of overlap and research, nevertheless [193] and [37] presents promising directions and insights.

Beside the rise and hype-up of ML-based technologies to level up automation in production, there are several issues that still limit their diffusion and the broad success of such: limited data availability in the phase of cold start or due to cost and time limitations leading to small data issue, lacking statistics, various data types, lack of standardization, low/inconsistent data quality, and over all high data complexity and data fragmentation. This accumulates into transfer and generalization as one of the main obstacles to achieve next level fab automation and overall digitalization. Therefore, the focus is now shifted towards standardization and scalability especially for application driven research.

The traditional (and without a doubt expedient) approach to deal with differences in production is overall standardization from technical as well as technological point of view. Identical environments enable easy, fast and straight forward deployment, role-outs, control and decision transfer and updates. Unfortunately, this copy exactly approach is often affiliated with extreme effort and costs; if possible at all. Even in industries where standardization of production is already adopted, data - and all its related aspects of identical as well as different systems (parallel running equipment, same process equipment or equipment from different vendors, different process equipment, ...) - is typically versatile, elusive, imperfect and highly variant. Therefore, production automation requires high-effort and most of the time dedicated development of ML models. Given the financial and technological efforts that needed to be spent for method/model development, 24/7 support for critical production infrastructure and maintenance - on top of the already existing high degree of automation - ML-based technologies are mandatory to be rolled out and deployed on broadest scale possible (e.g. for all available equipment/processes), hence they need to be scalable.

The power of ML already inherits a solution here: if overall standardization is not achievable, only partly achievable or achievable but limited in practice by cost, effort or time, data-driven transfer approaches combined with the power of ML - and especially DL methods - can be used to successfully tackle model generalization and enable scalability. The smart usage of advanced ML methods including its ability to combine, explain and visualize, finally results in successful and high quality transfer interventions with long term benefits on quality, stability and speed.

This work describes a newly developed approach to tackle model transfer in the style of domain adaptation for homogeneous as well as heterogeneous data representations to support generalization and scalability of DL-based statistical control methods. State of the art methods and well known and established classical methods are compared and tested against each other. A broad range of use cases are presented to showcase functionality but also variability of the approach.

Scalability inherits two different views: View one addresses active learning and the so called cold start problem. Active learning has its focus on the time component, meaning that at the beginning no or only very few data samples are available; known as the 'cold-start' problem. Active learning is applied to use available related data to improve the model performance for the new domain faster over time. Both cases show promising results by applying domain adaptation methods. By having only one model for multiple equipment we allow direct comparison but also reduced model maintenance effort, hence support scalability. By active learning the training of a model for new and unknown domain is enabled, allowing to fast include new data to a digitalized fab for example.

In the thesis a different view is addressed since the focus is on enabling the direct usage of an already trained and well performing dedicated domain-specific model to another identical (homogeneous) or non-identical (heterogeneous) domain where the underlying information but not the data representation is the same. Main attention lies on static use cases where data is available but unlabeled or where labels are partly (semi-supervised) or even fully (supervised) available. Nevertheless, since involved domain data is collected already for a certain amount of time, the final goal is to scale up and deploy one well performing model to multiple domains independently of the data representation.

Even if the thesis content including presented methodology is inspired by the needs, highest standards and complexity of semiconductor manufacturing, it can be seen as representative and paradigmatic of industry4.0 problems for variety, volume, complexity of products, processes and data in all kind of productive industrial environments.

1.2 Thesis Overview and Organization

The rest of the present work is organized as follows: Content related to semiconductor manufacturing are covered after this overview section.

Part II: Methods and Mathematical Tools focuses on general theory, methodology and mathematical foundations of all methods independently of any application:

Chapter 3: Basics recaps the building blocks of deep learning. Section 3.1: Neural Networks first covers important foundations of deep learning like learning paradigm, supervision, architectures and training of neural networks including hyperparameter tuning and backpropagation and a comparison of individual characteristics between discriminative and generative models. The presented basic ideas and methods are important for overall understanding of methodological design as well as use case sequence and set up.

Section 3.2: Generative Adversarial Networks (GAN) and Adversarial Training Approach introduces generative adversarial networks (GAN) including important literature. The focus here lies on one side on an existing methodological extension of classical GAN using Wasserstein loss to improve training and stability and on the other side on an architectural extension called CycleGAN that enables unpaired mapping and different kinds of property transfer; mathematical basics as well as their advantages are discussed. Wasserstein loss and the bijective unpaired mapping from CycleGAN are later reused for development of the main methodology.

Section 3.3: Domain Adaptation Theory and Domain Adversarial Neural Networks (DANN) covers theory of transfer learning with a focus on domain adaptation. Domain adaptation is based on information theory, learning theory, task relatedness and generalization. It deals with the fact that the assumption of train and test data coming from the same underlying distribution does no longer hold. The definition of a domain is given and used to summarize literature that define divergence measures and prove generalization bounds. Based on the theoretical framework of domain adaptation, domain adversarial neural networks (DANN) are introduced and mathematically explained including further research done in this field.

Inspired by the previous elucidations and content, **Chapter 4: DBAM** defines the DANN-based alignment model (DBAM). DBAM is a domain adaptation inspired DL method with the following advantages:

- deal with high context hence data complexity (e.g. caused by multiple systems in production) and is applicable to various data types like time series data, stationary data and image data;
- able to tackle supervised, unsupervised as well as semi-supervised regression and classification task for data containing different distributions and covariate shifts;
- assure interpretability and comparability of all parts of the model.

A deep-dive into the mathematical formalism of the corresponding optimization including selected loss functions and training algorithm is given: The loss functions are defined, adversarial training algorithm is detailed out and Wasserstein loss is put into context. The chapter closes with a synthetic data example where two gaussian distributed data sets with a shift in the mean are aligned with each other to showcase DBAM.

In **Chapter 5: DBACS: Extended DBAM** DBAM is extended with a cyclic supervision approach covering the topic of unpaired mapping and heterogeneous data representation in order to enable broad range of application and highest possible input flexibility. The method is called DANN-based Alignment with Cyclic Supervision (DBACS) and has the following additional advantages:

- heterogeneous domain adaptation;
- unpaired sample mapping;
- aligned features are comparable and interpretable in both directions;
- enables (feature) matching.

This chapter also closes with an the synthetic data example from the previous chapter to showcase functionality of the extended approach.

Part II closes with **Chapter 6: Benchmark Methods and Models** that gives an summary of all relevant benchmark models that are selected for the different use cases following literature recommendations and performance winners in similar environments and use case settings. Hence Chapter 6 covers classification and regression methods, ensemble learning and decision trees, linear and kernel transformations, feature- as well as instance-based domain adaptation methods, semi- and unsupervised transfer methods including DL-based methods as well as classical approaches.

In **Part III: Applications and Case Studies**, three main applications are presented, formalized, analysed and evaluated:

- **Chapter 8: Virtual Metrology (VM)**;
- **Chapter 9: Predictive Maintenance (PdM)**;
- **Chapter 10: Defect Classification (DC)**.

Those are selected since:

- all 3 are standard control applications for manufacturing and highly critical for production success;
- all 3 are often used applications for deep learning methods applied to semiconductor manufacturing while being active research fields themselves;
- all 3 show high data complexity while corresponding to different modeling aspects like different statistical processes (regression versus classification) and different data types (time series data, stationary data and image data);
- all 3 show the necessity for scaling and transfer in order to showcase their full potential and benefits in an production environment.

Chapter 8: Virtual Metrology (VM) introduces the first use case where the topic is on soft sensing/VM. The goal of such metrology technologies is to enrich collected data sets containing quantities and properties of a product after a process that are important for quality/control purposes but are impossible or costly to be measured. The chapter starts with an introduction to the metrology topic followed by more details on etching equipment and etching process that are analyzed. This is followed by a literature deep-dive covering methodological as well as application and task related research. Then two different case studies are presented: First virtual metrology for identical-in-design-equipment, covering solutions for VM regression task as well as homogeneous supervised domain adaptation including DBAM, is studied. Second virtual metrology for equipment with heterogeneous data representation dealing with data from two non-identical equipment running the same process with a view on the prediction task itself as well as domain adaptation possibilities including DBACS are investigated. Inspired by heterogeneous domain adaptation and the functionality of DBACS, Section 8.5: Enabling Equipment Matching focuses on (equipment or feature) matching before the final section revisits and sums up results and findings of the whole chapter.

Chapter 9: Predictive Maintenance (PdM) present the second use case that is connected to process control namely predictive maintenance (PdM). In manufacturing, maintenance describes the process of equipment life cycle planning including inspections of the equipment, the equipment repair and anything else related to know and control the condition of production equipment. Similar to VM, PdM systems exploit sensor data that is already collected during a production process by the equipment itself avoiding additional costs to predict and estimate life cycle relevant values, here remaining useful life (RUL), to avoid uncontrolled equipment breakdown. The chapter starts with more details about PdM and discuss its advantages for production. Next the concrete task including necessary background information namely a prediction model for RUL for a group of implant equipment is explained. This is followed by a literature deep-dive divided into different sections: classification task for PdM, regression task for PdM and adaptation related methods for PdM. Next the use case Remaining Useful Life (RUL) prediction for PdM is presented with a description of the input data, modeling of target labels, data preparation including details about the specific handling of data including repetitive lives and finally modeling details including DBAM as well as DBACS, benchmark models and hyperparameter. Results are presented in detail and discussed. To close up, the topic of imperfect data and its influence on prediction quality for PdM modeling is addressed. This use case is ended with a recap and summary.

The third and final process control application presented in **Chapter 10: Defect Classification (DC)** is about different wafer defects occurring in front end production: Detecting and classification of different defects on the wafer is one of the most important mechanism for assuring highest quality and early as possible identification of faulty or malfunctioning products and hence increasing overall yield. With the introduction of convolutional neural networks including powerful architectures able to classify complex data sets including versatile classes, literature shows tremendous success in fully automated defect detection and classification. After giving a quick recap on defect classification, the chapter starts with a literature overview focusing on semiconductor related publications. Then the task as well as experimental design is explained followed by a detailed data preparation journey. Next DBACS and AdaMatch are introduced - both are models able to deal with various domains characterized by different image backgrounds - and architecture as well as hyperparameter settings are detailed out. Finally results are presented including a deep-dive into specific modeling details like distribution alignment and imbalance classes that deserve further attention.

Finally the thesis closes with a sneak peak into adjoining methods and an overall conclusion in **Part IV: Conclusion**.

2 Elements of Semiconductor Manufacturing

Semiconductor manufacturing is very complex and a complete explanation of processes and procedures is outside the scope of this work. Here instead, we give a rough overview of the first part of semiconductor manufacturing called front end and detail some aspects of direct interest for the work reported in this thesis.

2.1 Semiconductor Wafer Processing

The basis for every semiconductor manufacturing process is a thin disc of pure (99.9999%) silicon e.g. called raw wafer. A wafer is the base of microelectronic devices called chips. On average there are several thousand chips build up on each wafer layer by layer depending on its diameter and on the chip size. A wafer typically describes one discrete iteration or sample for any modeling task considered in this work. Typically one box called lot is packed up with 25 wafers with common properties that will undergo similar processing steps.

The fabrication plant - called fab or site - is divided in different areas where dedicated process steps are executed. An area includes multiple workcenter with a number of identical and different equipment (same vendor and model, same vendor different model, different vendor and different model). A workcenter is defined as a group of equipment that pairwise have at least one common process step. A so called cluster equipment consists of multiple chambers able to perform the same processes in parallel supported by some common gas and energy sources for example. A cluster equipment can also describe an equipment where multiple chambers are available but each chamber is responsible for a sub-step of the total process step. For some process steps each wafer is processed individually whereas some equipment are able to process one lot (lot-by-lot process) or even multiple lots at once (batch processing).

Lots are transported by an automated transport system in general referred to as automated material handling system and distributed via a so called dispatching system. Different dispatching rules are available depending on utilization and scheduling besides other influential factors. In a modern fab - especially for 300 mm wafers - this is rule-based or fully automated and not the responsibility of humans named operators anymore.

The product portfolio in front end manufacturing follows a certain hierarchy: Technology - process class - process group - process line - product. Each product is defined via routes, sometimes sequences of routes in case multiple sites are involved. Routes consist of a sequence of several hundred single process steps (SPS) that are well defined and deterministic hence can be reused for different products. In addition wafers can be booked on additional routes for inspections or defect measurements but return afterwords to the original route if they fulfill product specific measurement criteria. The SPS defines the process a wafer has to undergo, it is also called recipe. Usually the assignment of SPS to equipment is neither unique nor equipment specific meaning multiple chambers as well as different equipment are able and allowed to run the same SPS. A so called dedication matrix holds the necessary information and permissions for that. The fastest available equipment stated in the matrix is chosen to carry out the necessary process step.

2.2 Production Flow and Processes

Semiconductors are constructed by linking circuit structures on various layers build upon a wafer. The realization of those necessary layers including their complex individual structures and patterns are done in a sequential manner in the front end processing. Front end relates to the structure building part of chip fabrication and happens in a regulated environment called *clean room*. There are multiple different process steps that are performed in a front end wafer fab (see Mönch et al. [148]) to build up the electrical circuits on the surface of the wafer:

- Oxidation/diffusion with *furnace* for either healing of crystal damage and activation of ions or thermal processing in order to grow or deposit a layer of material on the surface of a wafer.

- *Deposition* processes like physical and chemical vapor deposition as well as expitaxy or metalization in order to deposit thin films on the wafer.
- *Photolithography* for structuring the wafer surface.
- *Etching* for (local) silicon as well as oxide, nitrite, metal removal. We distinguish between wet and dry etching, where liquids resp. gases are used. A more detailed description is given in section 2.2.
- *Ion implantation* for definition of conductivity of the silicon.
- *Planarization* also called chemical-mechanical polishing as equalizer of the wafer surface. A more detailed description is given in section 2.2.
- In addition a sample of processed wafers are tested in the *metrology* after a process step to detect defects or any kind of particles and dirt on the surface. A more detailed description is given in section .

Regular repetition of process steps and re-entry into the production flow are part of the advanced, highly specialized and highly automated fabrication process. In the following more details about two highly complex and highly researched processes - *etching* and *implantation* - are given, since they give the basis and source of the involved data analysed later in Chapter 8 and Chapter 9.

Etching Process

This section follows the description of Hilleringmann [88], Chapter 5. The etching process erodes a material from a surface or transfers a structure built up during the lithography step to the layer below. This processes can be done by *wet etching* based on chemical liquids or by *dry etching* based on physical processes together with chemical gases. In general etching can be *isotropic* where the etching evolves equally in all directions or *anisotropic* where clear and well-controlled features are built up. The ratio of the etch rate in two different materials is called selectivity. Wet etching cannot be used for creating precise structures, so it is mostly applied to create uncritical structures, removal of whole layer or cleaning.

Dry etching is a process with a high reproducibility resulting in homogeneous isotropic as well as anisotropic structures. The process involves vaporous material that are activated through a high frequent alternating energy field in an area with sub-atmospheric pressure range. The kind of erosion depends on the gases that are involved in the process.

Sputtering is a method based on purely physical reactions. For inert gases we see a purely physical erosion since accelerated ions transfer their energy onto the surface of the wafer causing atoms or molecules to be knocked out. A chamber consists of two electrodes installed opposite of each other. Applying an alternating energy field between those electrodes while creating a chamber pressure in the area of 5 PA, an electrode voltage called bias is generated that influences the positive loaded ions. Thus an electrostatic potential in the chamber is build up. Instead of two parallel plates one can also use an electromagnetic coin to generate plasma. Aggressive radicals arise as a consequence of collisions in the plasma attacking the wafer positioned on the grounded electrode leading to a chemical isotropic reaction on the surface of the wafer.

Plasma etching is a combination of physical and chemical reactions. For reactive gases, the chemical reaction is supported by additional energy of ionized gas molecules. Reactive-ion etching uses the same chamber construction described above plus a high-frequent alternating energy field that is applied on the cathode the wafer is positioned on. Ions with positive loading within the plasma are accelerated to the wafer where they hit its surface upright due to their high kinetic energy. This etching is anisotropic. The atoms of the wafer surface are freed from the crystal lattice resulting in a partly physical and a partly chemical reaction in addition to the chemical etching of the radicals described before.

In summary, the result of the etch process depends on various factors: the material that has to be etched, the properties of the preprocessing like layer thickness, layer constitution, dimension of the mask and the process properties like the pressure, the applied high frequency voltage, the gas type, the gas flow and the wafer as well as the electrode temperature and as a consequence on bias, etching rate and selectivity. The physical quantities are explicitly or implicitly measured by the preinstalled sensors and collected. The etching recipe can include a fixed etching time or an automated end point detection is installed to detect the best possible time point to end the process where all the necessary material but not more is eroded. Atomic emission spectroscopy can be used for such an end point detection as well as interferometry. If the detection is based on optical properties, a change in the chemical components of the plasma or a change in the itensity of the light waves (meaning the spatial period if there is a transition from one layer to another) is recognized. By using interferometry, a direct measurement of the etch thickness is possible on the center of the wafer.

A plasma etching process step can consists of a high number of sub-steps with different duration and gases involved that are defined in the recipe. Before the start of a new sub-step, the chamber has to be in a stable condition hence a settling time has to be taken into account.

The end point detection resp. the etching time are very critical, highly sensitive and highly correlated to other involved variables like gases, pressure, current, temperature. Incorrect etching times, insufficient end point detection but also unpredictable reactions and interference in the chamber can lead to so called under or over etching. In both cases the geometry especially the etch thickness is influenced negatively, leading to failure or malfunction of the chip. In all cases undirectional etching has to be prevented.

Implantation Process

. This section follows the description of Sze and Lee [195], Hilleringmann [88], Chapter 5 as well as May and Spanos [146], Chapter 2, Section 2.1.4. First introduced in the 1950s as *impurity doping* in semiconductor manufacturing, wafer implantation also know as *ion implantation* - besides *diffusion* - is one of two possible processes of impurity doping. A so called impurity - meaning energetic dopant ions - is implanted into the semiconductor atomic structure by means of an ion beam. The implant profile and distribution is mainly defined by properties like mass and energy of the ion itself. One distinguish between high, middle and low current *implanter*. The implanter dose is defined as the number of ions implanted in 1 cm^2 of semiconductor surface area. A so called ion source is releasing source gas that is then broken up into single positive charged ions. An important part of the ion source is the filament. It is stressed a lot and regular breakages occur. Hence it is one of the main causes for maintenance. Figure 2.1 showss a picture of an ion source including different filament conditions by Scheibelhofer et al. [176].

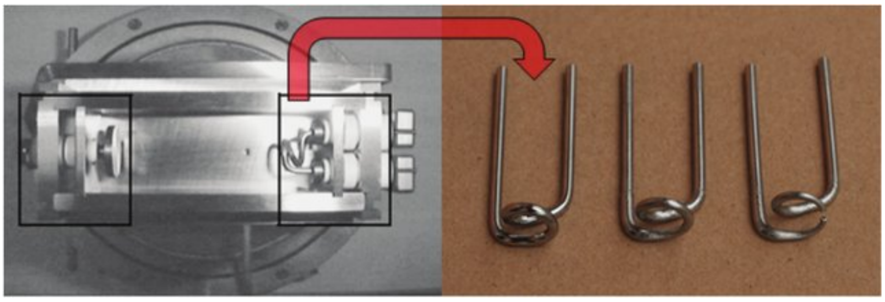


Figure 2.1: Implant ion source filament. Implant ion source and different filament conditions as presented in [176].

A mass analyzer is filtering released ions with the right properties. Selected ions are then accelerated to the necessary implantation energy. This charged particle beam (ion beam) is scanned finally over the wafer surface and the ions are implanted. Therefore the ions loses some of their energy by crashing with electrons until they come to rest at a certain depth within the lattice of the wafer.

Due to the ion collision the wafer lattice can be damaged. (Thermal) annealing for example with a furnace can be used to repair those damages. A plasma flood gun (see [100]) is used to neutralize the charge-up of the target surface by the ion beam. It provides a flow of low-energy electrons and directs it over the wafer surface. It avoids the breakage of the insulating film on the wafer if the surface charge grows greater than the insulation breakdown voltage of the film. Unwanted effects corresponding to the wafer that can occur during implantation are sputtering or crystallographic damage besides others.

2.3 Process Control: Maintenance and Metrology

Introduction to Process Control

On the equipment, a process step is specified by a recipe. It defines process properties like number of steps and equipment conditions for those steps. During the wafer processing, sensors on the equipment collect physical measurements of important parts and materials involved. Unfortunately not all parameters of interest can be measured. One example is the exact wafer temperature. Instead the temperature of the e-chick is measured. It is useful to distinguish between raw sensor data called *trace data* and descriptive statistics called *fault detection and classification (FDC)* or *keynumbers* that are directly computed from the trace data. On the one hand, those measurements are used to keep the process stable meaning automated regularization by the equipment itself installed by the vendor. On the other hand it is the main data source for process control where for example each recipe includes some limits within an unproblematic flow of the process is assured. In univariate process control, the exceed of such limits raises an alarm that can - if detected early enough- avoid damage of a wafer or an unscheduled down hence unplanned maintenance action. Interruptions like this or equipment shut downs with unknown cause are cost intensive. Hence a lot of research and control effort is put into the so called predictive maintenance (PdM). Based on FDC data the goal is to predict as early as possible and therefore avoid as much unplanned down times as possible to assure a smooth process and production flow without unnecessary damage on equipment and wafers. Besides that there are frequent scheduled maintenance actions based on experience and empirical values in order to replace highly worn-out parts (due to sedimentation, dirt or degradation for example) of the equipment in time to prevent poor process results. So called unscheduled downs (UD) as well as scheduled downs (SD) are documented in the *resource time classification (RTC)*. The ratio between SD and UD is supposed to be as high as possible with a low overall number of actions in order to guarantee a smooth process flow and highest fab performance.

Maintenance

In manufacturing, *maintenance* describes the process of equipment life cycle planning including inspections of the equipment, the equipment repair and anything else related to know and control the condition of production equipment. For an introduction to maintenance including its history see Mobley [147], for a broad range of applications in different industrial fields see Cerquitelli et al. [27]. There are different modes and strategies available how to tackle maintenance in order to find the best balance between running/production time and cost avoidance:

- *Run-to-Fail* describes a strategy without human involvement before the machine breaks down with the risk of damaging processed wafers and other equipment parts. Nevertheless, this provides accurate and highest information data to train a statistical (prediction or classification) model;

- *Reactive Maintenance* stands for the responds to a breakdown and the then triggered repair. Run-to-fail can be seen as a subcategory of reactive maintenance.

- *Preventive or Scheduled Maintenance* stands for planed and in regular time intervals reoccurring inspection or repair of equipment to avoid possible uncontrolled breakdowns;

- *Condition-based Maintenance* describes the usage of fixed indicators or triggers. If those are sending alarms then a maintenance is performed;

- *Predictive Maintenance* is the one mostly connected with industry 4.0. PdM stands for some kind of heuristic or statistical method/model used to foresee upcoming breakdowns and time to failure or estimate the time often referred to as remaining useful life (RUL) - until such a breakdown occurs next. Usage-based Maintenance is a subcategory where utilization is a main indicator for failure;

- *Prescriptive Maintenance* Prescriptive Maintenance is an expanded PdM with a root cause analysis on top, meaning it automatically detects why a failure happens on top of what fails and when it is going to fail. It is seen as a whole intelligent analytical monitoring system.

Down events can be caused by a versatile amount of root causes often not clearly distinguishable. They depend on the degradation effect of run processes and life span of different equipment parts. Other impact factors on maintenance schedule are logistic, availability of personal, spare parts, equipment utilization and internal prioritization rules.

Virtual Metrology and Run-to-Run

If the process resp. the process technology is well understood - meaning you can build a model to describe the influence of process parameters on the process result for example influence of etching time on etching thickness or gas flows on profile width of contact wholes- an *advanced process control (APC)* system called *run-to-run (R2R)* can be introduced to an equipment. It allows automated modifications of the recipe between two runs (between two wafers are processed) in order to minimize drifts, shifts or variance and therefore increases the control over the process step. After a lot is processed a sample of wafers from this lot go to

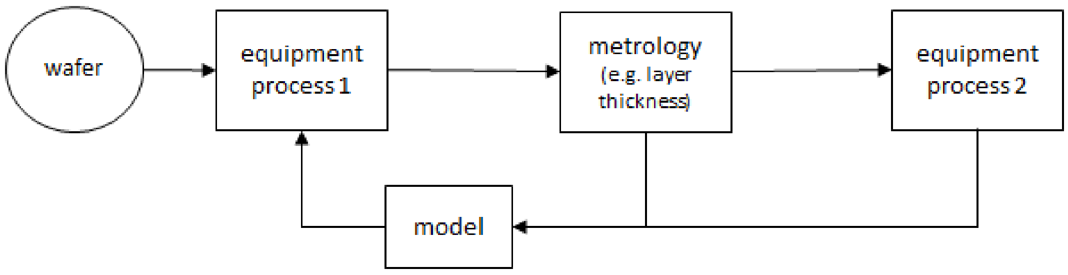
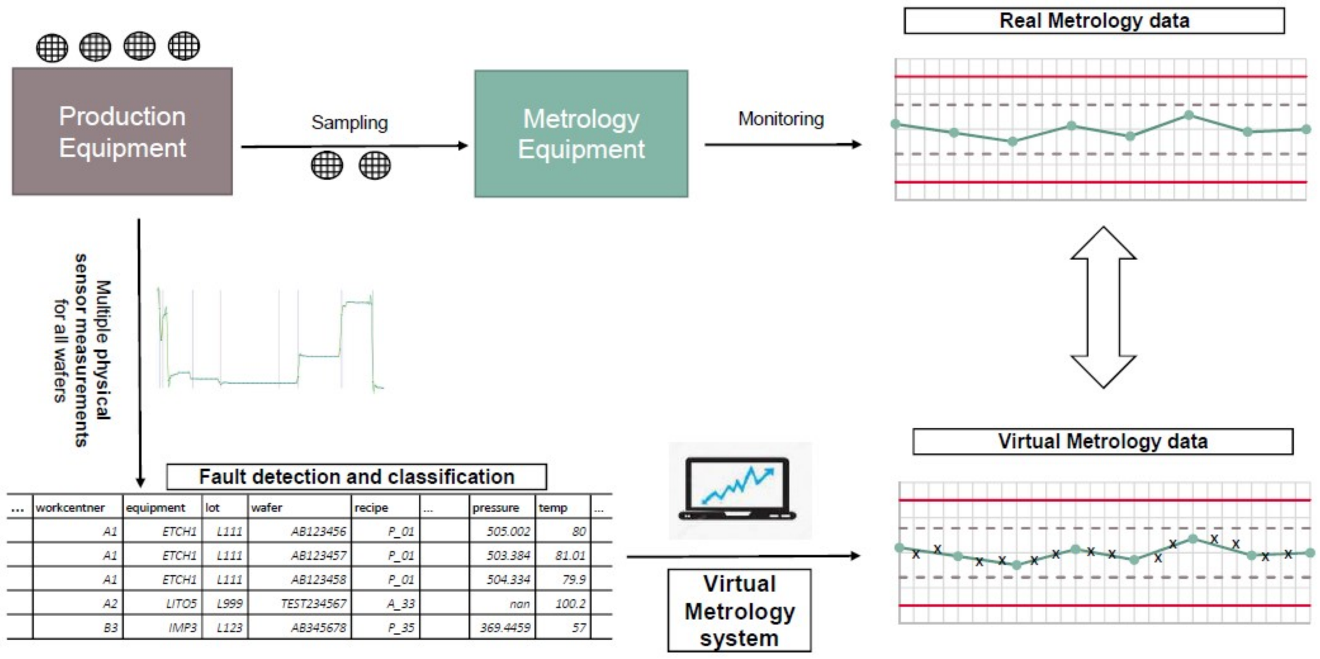


Figure 2.2: Visualization of a run-to-run system. Visualization of a run-to-run system (R2R)

a metrology equipment where product measurements are taken in order to assure quality and to verify the target of the process step. In the *statistical process control (SPC)* data multiple product related measurements are taken on multiple spots on the selected sample wafer. Since this causes high costs or even effect the functionality of the tested wafer, another APC system called *virtual metrology (VM)* or soft sensing (SS) can be provided for an equipment if necessary knowledge is available and a statistical model can be formulated and trained.



...	workcentner	equipment	lot	wafer	recipe	...	pressure	temp	...
	A1	ETCH1	L111	AB123456	P_01		505.002	80	
	A1	ETCH1	L111	AB123457	P_01		503.384	81.01	
	A1	ETCH1	L111	AB123458	P_01		504.334	79.9	
	A2	LITO5	L999	TEST234567	A_33		nan	100.2	
	B3	IMP3	L123	AB345678	P_35		369.4459	57	

Figure 2.3: Virtual metrology schemata. Visualization of virtual metrology (VM) [70].

VM is a model based statistical prediction of the wafer properties based on sensor data measured during the process and therefore takes the place of the indispensable costly product measurements.

Semiconductor Scanning Electron Microscope Metrology

A *scanning electron microscope (SEM)* is a type of electron microscope which produces images of a specimen by scanning its surface with a focused beam of electrons. The electrons, by interacting with atoms in the specimen, produce several informative signals about the surface topography and composition of the specimen. Secondary electrons emitted by the specimens' atoms excited by the electron beam can be detected by using an in-lens detector or an external detector. Depending on the type of detector, SEM images can be therefore divided into in-lens detector images and external detector images. This means that there may exist more than one image of a single defect. These images are very similar but present variations, for example a small translation, or different contrast due to the different tilt angle of the in-lens and external detectors as presented in Figure 2.5.

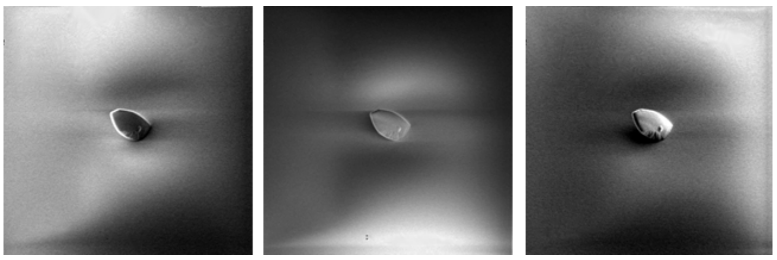


Figure 2.4: SEM image variations based on detector and angle. Example of possible image variations for one specific defect sample. The appearance depends on the used detector and angle when the image was taken.

Defect detection and classification is a fundamental part of production control and management to ensure highest product quality and production performance. Scanning Electron Microscopes (SEM) images of the wafer are acquired after the completion of certain process steps, in particular between etching and deposition.

The collected data is examined by humans or automatic systems and divided into different defect types. Often detected defect classes are *particles*, *scratches*, *spots*, *points* or *irregular connections* besides others. Twelve examples of defect classes are shown in Figure 2.5. If a defect is detected, corrective actions or declaration of 'defective' products are triggered. The detailed inspection procedure involving SEM images is depicted in Figure 2.6. The procedure follows the following steps:

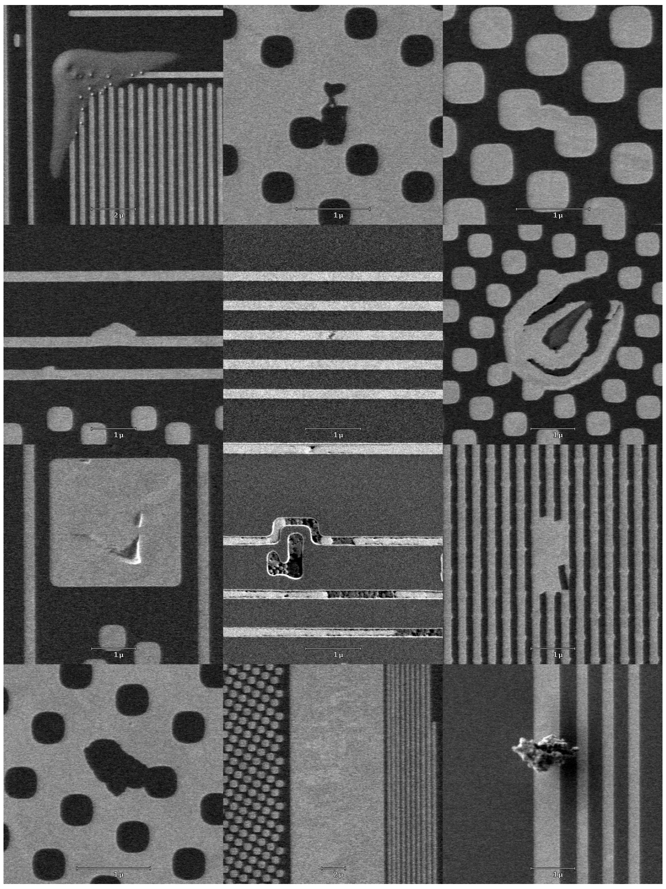


Figure 2.5: Twelve examples of SEM defect images in silicon wafer. Example of possible SEM image defects variations The appearance and background depends various context categories.

(i) an optical inspection tool locates defects (marked in red) by comparing multiple neighbouring dies on a wafer and looking for anomalies;

(ii) high resolution SEM images are taken at each defect location;

(iii) defects are classified.

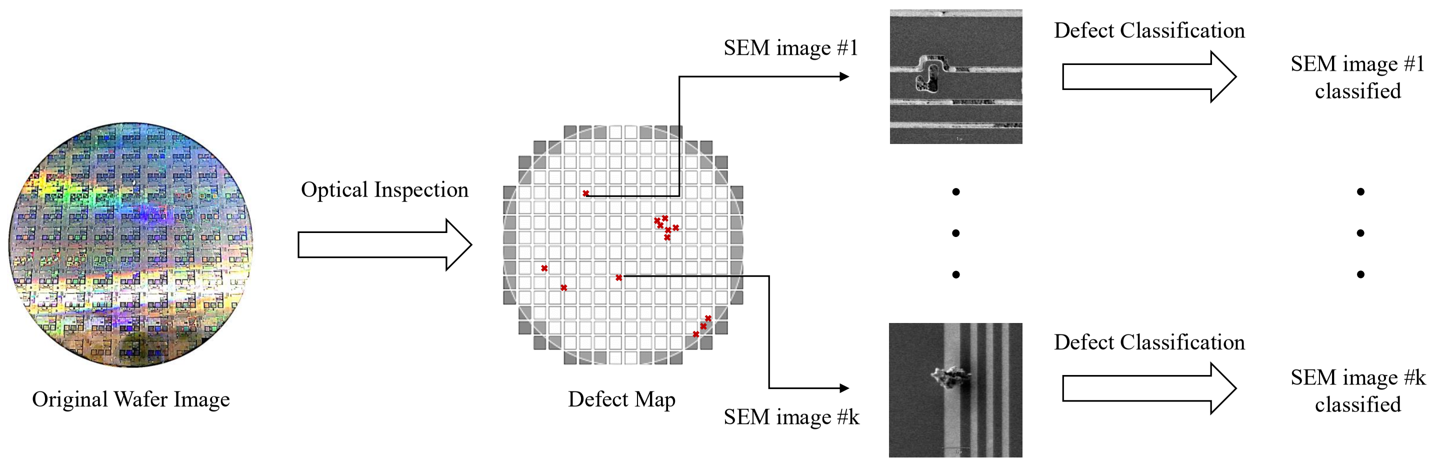


Figure 2.6: Defect classification schemata for process control. Defect classification procedure for the semiconductor manufacturing process at hand: (i) an optical inspection tool locates defects (marked in red) by comparing multiple neighbouring dies on a wafer and looking for anomalies; (ii) high resolution SEM images are taken at each defect location. (iii) defects are classified without knowing the location on the die or wafer. Differences in the image background structure are visible.

Part II

Methods and Mathematical Tools

3 Basics

In this chapter we introduce methods and models that are later exploited for use case evaluation in Chapter 8, Chapter 9 and Chapter 10. Again, we want to stress that presented methods are not bound to semiconductor field in general or limited to semiconductor manufacturing use cases. Hence they are formally introduced and theoretically analyzed completely independent from Part III of the thesis.

3.1 Neural Networks

Introduction and Idea

Artificial or feedforward neural networks (ANN) also called multi-layer perceptrons (MLPs) are the simplest but also most known and applied deep learning models. A summarized historic overview including important references is given for example by Schmidhuber [178] starting from the first theoretical ideas in the 1940s over development of backpropagation to usage of restricted boltzmann machine to model each layer. While Goodfellow et al. [74] also shortly addresses the historical development at the beginning, it focuses on a detailed theoretical explanation of main concepts and methods applied in deep learning.

The biggest advantage of a neural network compared to classical machine learning approaches are its ability to model nonlinear relationships without knowing the underlying functional structure. In addition its model quality is independent of the concrete representation of the data, as long as the hidden or relevant information is identical; for example cartesian versus polar coordinates, logarithmic vs normal scale.

The basic idea is to successively connect simpler, affine linear functions in a way that allows to achieve the necessary functional complexity needed for the modeling problem at hand. An illustration of deep learning model structure dependent of the corresponding modeling task is given in Figure 3.1: Each circle called node represents multiple parameter whose values have to be learned during the training process based on training data as well as a performance measure called loss. Each 'column' or layer of nodes increases the depth of a ANN. The depth adds complexity but also enables a new latent feature representation for extracting, evaluating and keeping the information hidden in the data.

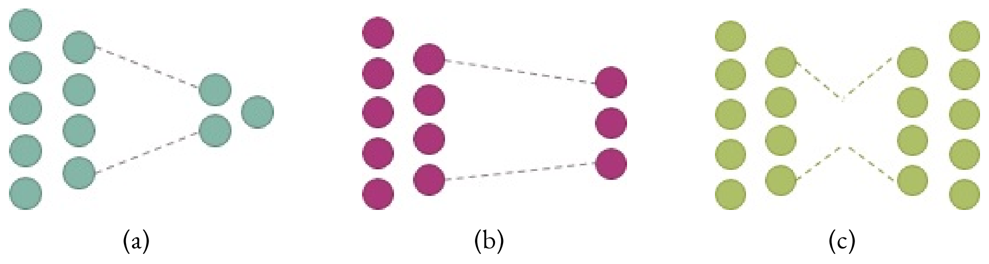


Figure 3.1: Illustration of basic deep learning model structures. Graphical representation of the concept of deep learning models: (a) regression, (b) classification and (c) autoencoder. Regression and classification are typical examples for supervised learning problems while autoencoder belongs to the unsupervised category.

The training for minimizing the loss is done in a repeating manner until a stopping criteria is reached. First evaluating the performance with an algorithm called forward-propagation, then the parameters are updated using an algorithm called back- propagation. The upcoming sections give further insights into neural network methodology, architecture and training.

Learning Paradigm and Supervision

In general there are 3 basic 'paradigms' of deep learning:

- **Supervised Learning (SL)**: Given an input X as well as an output Y called labels or target, the objective is to learn a mapping respective the parameter values of the neural network based on given (input,output)-sample pairs such that it gives the best possible representation of the unknown underlying function.

- **Unsupervised Learning (UL)**: No output data and hence no labels are available so the training has to work with indirect feedback or based on input data where the focus is put on the density or distribution itself.

- **Reinforcement learning (RL)**: A reward function instead of direct comparison with a loss is used to assess actions in an environment the so called agent can interact with.

In cases where only a small data set or only a limited amount of labeled data is available, a hybrid paradigm between supervised and unsupervised learning called **semi-supervised learning (SSL)** can be introduced. Chapelle et al. [29] gives an introduction and describes different assumptions used for SSL as well as algorithms and practical examples. If only a few samples or only one single labeled sample is available, specified SSL methods called few-shots or one-shot learning show very promising results, see for example the approaches presented in Koch et al. [115] and Vinyals et al. [208].

Another closely related hybrid category that becomes more and more relevant due to its success in the field of natural language processing (NLP) is **self-supervised learning**. A well-known example called *bidirectional encoder representations from transformers (BERT)* is presented by Google in Devlin et al. [52]. Compared to more traditional SSL approaches, self-supervised learning focuses on a pre-training step. The goal is to presents a well initialized network that is task independent on the one hand but on the other hand already includes powerful context-based latent representations. With the help of self-supervised learning state-of-the-art results are also achieved in the field of computer vision, see for example Grill et al. [76].

Discriminative versus Generative

ML methods and ANN are categorized into **discriminative** and **generative** models, depending on their study and usage of probabilities (see [102]).

In the **discriminative** case a functional representation of the conditional probablility is assumed and training data is used to estimate corresponding defining parameters. Supervised machine learning methods for solving classification or regression problems - like (multi-)linear regression, support vector machines and decision tree based models - are typical examples for discriminative models. Figure 3.1 presents a structural scheme of ANN for (a) regression and (b) classification task. ANN used in that context are also discriminative.

The goal of a **generative** model is to describe the underlying data distribution (or the distribution of different classes) based on the sample data. Hence it tries to learn defining parameters of the joint probability of input and output and therefore often does not rely on the availability of labels. Well known generative methods in the field of deep learning are variational autoencoder (VAE), bayesian networks and generative adversarial networks (GAN).

[74], Chapter 20 contains a general description of generative models. GAN are a main source of inspiration for the presented work hence a detailed description is presented in Section 3.2: Generative Adversarial Networks (GAN) and Adversarial Training Approach. It is closely related to domain adversarial neural networks (DANN) described in Section 3.3: Domain Adaptation Theory and Domain Adversarial Neural Networks (DANN). Evaluating generative models is difficult because the ground truth distribution is unknown. see Goodfellow et al. [74], Chapter 20. Deeper discussion about occurring problems and challenges are presented in Theis et al. [201]: It shows that state-of the art methods are independent of each other in the sense that consistent results cannot be expected above them. Hence, a selection of methods can only be done application specific. Following this approach, use case and data type specific benchmarking is introduced and use case specific evaluation is done in the corresponding use case chapters.

Neural Network Architectures

Depending on the input data type, different deep learning architectures are developed. In the following we summarize the explanations given in Ramsundar and Zadeh [164] and Goodfellow et al. [74]. Some valuable insights are also presented in MacKay [143] and Bishop [20].

Fully connected neural network (FNN) Fully connected neural networks (FNN) consist of a series of layers where each neuron, also called activation units, is connected to all other neurons of the neighboring layer. Three layer schemata are distinguished in general as visualized in Figure 3.2:

- Input layer: First layer of the neural network that takes the input data and features to subsequent ones without doing any transformations;
- Hidden layer: A network can have one to many layers that connect the input and the output. They are build up using different building blocks and the main transformations are performed here. If a network has multiple hidden layers it is called deep. An investigation of deep versus shallow neural network is given by Delalleau and Bengio [51].
- Output layer: Last transformation before the final result is read out. Output depends on the problem type.

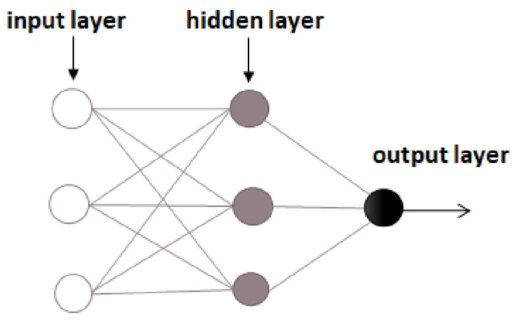


Figure 3.2: Layer schemata for neural network.

Since no assumptions need to be made about the input (this is called "structure agnostic" in [164]), FNN can be used for all kinds of data and problems and are suitable for a broad field of applications. Top-performing for stationary one-dimensional data, it shows a weaker performance applied to for example image data where dedicated architectures are developed.

Let $f : x \rightarrow y$ define a modeling step/ transformation step of a fully connected layer with $x \in \mathbb{R}^n$ be the input and $y \in \mathbb{R}^m$ be the output. Then y is computed via matrix multiplication

$$y = f(x) = \sigma\left(W \cdot x\right) \tag{3.1}$$

with σ defining a fixed nonlinear transformation (applied element-wise) called activation function, and the layer transformation f is parameterized by $W \in \mathbb{R}^{m \times n}$ where W is described as a matrix with the parameters to be trained. The activation function is used to define the output of the layer and can be as simple as a affine linear function (default value) but also highly nonlinear. Nonlinearity can enhance the representation skills of a network, hence is a more powerful modeling tool for various tasks. A special activation often used for classification in the final output layer is the so called sigmoid (binary classification) and softmax (multiclass classification) activation that maps the output to a probability distribution over classes. For more details on output units see [74], Section 6.2.2. Another example is the rectified linear unit (ReLU); see [74], Chapter 6, Section 6.3. It is a piecewise linear function, that is identical to the identity function for positive values and zero for negative values. It avoids the vanishing gradient problem, hence it is fast and stable in training. A variant is called LeakyReLU that has a small slope instead of a flat slope for negative values. Visualization of LeakyReLU and sigmoid are presented in Figure 3.3. More interesting details on the aspect of nonlinearity are discussed for example in Oja et al. [152] in the context of principle component analysis versus neural networks.

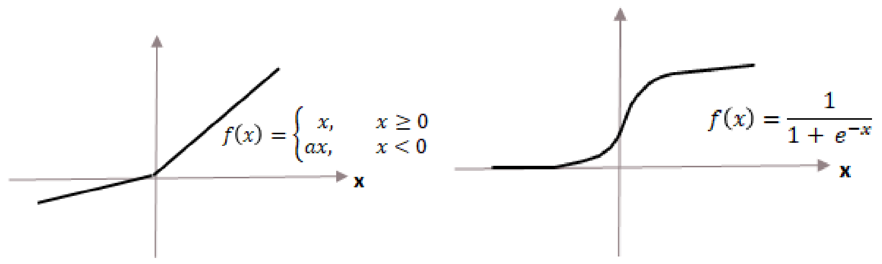


Figure 3.3: Visualization of LeakyReLU and sigmoid activation function. Left graph shows LeakyReLU, right graph shows the sigmoid function.

Convolutional neural network (CNN) Convolutional neural networks (CNN) are developed and optimized for analyzing grid-based data like time series or images. First introduced in the late 1980s - for one of the first publications including the base architecture see LeCun et al. [121], for one of the first describing and applying the backpropagation algorithm see Lecun et al. [122] - it is now thanks to high effort in research e.g. architectural structure, training methods as well as availability of high compute power and hardware - the go-to and state-of-the-art architecture for solving computer vision tasks. For example the model scaling of CNN for solving image classification tasks based on transfer learning (see Tan and Le [199]).

A convolution is mathematical operation based on the integral over the product of two functions (input and kernel). A multi layer network is called CNN if at least one of the layers conduct a convolution since most of the time at least the final layers are build using fully connected layers. For a mathematical description and example see [74], Chapter 9.1. With $x(t)$ as input function, $\omega(t)$ as kernel, $s(t)$ the output function and t describing the index of the corresponding grid (one dimensional vector for time series, two dimensional for images), a discrete convolution can be defined as

$$s(t) = (x * \omega)(t) = \sum_{a+s=t} x(a)\omega(s). \tag{3.2}$$

For real world data, the assumption that t is an integer value and the assumption that there exists only a finite number of addends (equals to a finite sum) holds true. The kernel ω is a multidimensional array with parameters that needs to be learned during training. A well-known example of such an operation is the Fourier transformation or the discrete Laplace operator (the kernel there is known and predefined). Some variants of the original convolutions are available that can level-up the model depending on specific use case at hand, examples are transposed convolutions, dilated convolution, separable convolution and depthwise convolution besides others. The mathematical operations that are often combined with a convolutional layer is activation and pooling. While activation is the same as for FNN since it is applied element-wise, pooling stand for a simple mathematical operation based on descriptive statistics like min, max or average in order to change (mostly reduce) the output dimension. For an overview of advances and alternatives see Gu et al. [77].

Bello et al. [12] shows the advantage of combining both CNN and so called self-attention to include more global or longer-term information instead focusing only on local neighborhoods. In comparison to pooling or convolution a similarity function between hidden units is introduced and a weighted average of those values is computed based on the similarity. Hence the importance of relative location can be replaced by important long term relations. For neural networks, let $W(t)$ and $b(t)$ be respectively the layer's weights and bias, and $x(t)$ the input at time t. Then, the attention weight $a(t)$ at time t is computed with

$$a(t) = softmax(tanh(x(t) \cdot W(t) + b(t)). \tag{3.3}$$

Softmax is used to ensure that the sum of all attention weights is 1. The attention layer output is then the sum of observation weighted with the attention weight. A very common attention function called additive attention is presented in Bahdanau et al. [8] for example.

One dimensional convolutional neural network (1DCNN) CNN uses filters that extract valuable information from a region of a certain size to detect underlying shapes and topologies from the data it is applied on. Since convolutional filters of any dimension can be applied, CNN are also usable for one-dimensional time series input data: Figure 3.4 shows a graphical explanation of a one dimensional convolutional layer and a pooling layer with time series data. A convolutional layer suitable for one dimensional input includes a kernel that only slides along one dimension. The kernel size is representing the width of the kernel and stands for the number of time-steps that are considered at once comparable to a moving window. The filters defines the number of output features you have from this layer. Kiranyaz et al. [114] for example presents an overview including different industrial applications and case studies.

Temporal convolutional neural network (TCN) Temporal convolutional neural network (TCN) presented by Lea et al. [120] are an advanced version of CNN especially

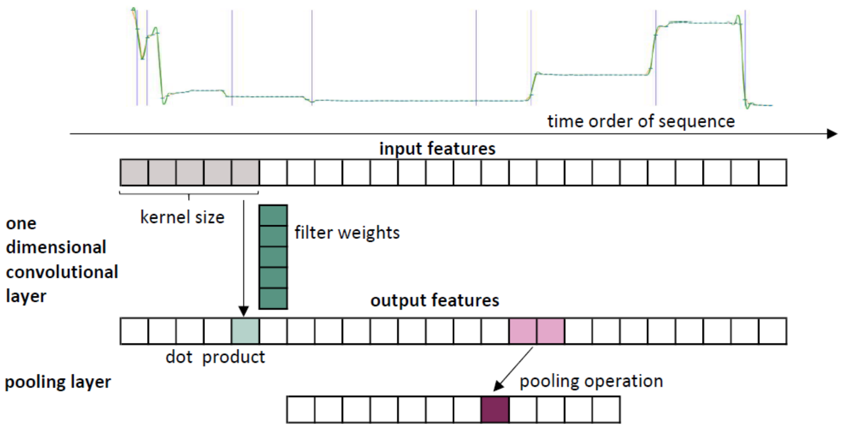


Figure 3.4: Visualization of a 1-dimensional convolution. Representation of a one-dimensional convolutional layer followed by a pooling layer applied on time series input data [70].

designed for time series input data and include further time series specific methodological advances. An intensive study on RNN and TCN for time series data in general is given by Bai et al. [9]. TCN architecture is based on CNN. The convolutions are causal to avoid information leakage and zero padding of length (kernel size -1) is added to keep the subsequent layers of the same length as the previous ones. Main enhancements in TCN are the implementation of dilated convolution by Yu and Koltun [227] and residual connections by He et al. [84]. The WaveNet model by van den Oord et al. [205] shows the advantages for a text to speech /speech synthesis use case.

A dilated convolution is a widened convolutional filter with spaces or skipped steps in the kernel element. An additional parameter d called dilation factor tells how much the input is expanded. Let again $x(t)$ be a time series input function and let $\omega(t)$ define a (finite) kernel and t describing the index of the corresponding one dimensional grid. Then the dilated convolution operation s_d on element t of the sequence is defined as:

$$s_d(t) = (x *_d \omega)(t) = \sum_{a+d\cdot s=t} x(a)\omega(s). \tag{3.4}$$

with $*_d$ dilated convolution and d is the dilation factor. The summation makes the skipping of some values visible, see Equation (3.1). Dilated convolution with dilation $d = 1$ yields the standard convolution.

Residual connections are used to connect functional representation learned during training with the original input. So called shortcut connections that skip one or more layers are introduced, working as identity functions. The identity output is then simply added to the output of the preexisting layers, the activation function is finally applied to the whole sum. Since no additional parameter are added and the computational effort is unchanged, it addresses the degradation problem (exploiting or vanishing gradient, accuracy loss,...) of deep neural nets without adding any complexity. The residual connection set up makes deep neural networks seem very similar to shallow networks in a ensemble setting.

Long short term memory network (LSTM) Long short term memory (LSTM) by Hochreiter and Schmidhuber [91] is a variant of recurrent neural network (RNN) with improved information flow and consideration of long-term dependencies. LSTM has a typical RNN chain structure with repeating modules and recurrent connections. One module called memory cell includes four neural network layers. A visualization of such a memory cell by Yuan et al. [232] is given in Figure 3.5. The notations in Figure 3.5 are reused for further explanations. There are three different gates in a memory cell:

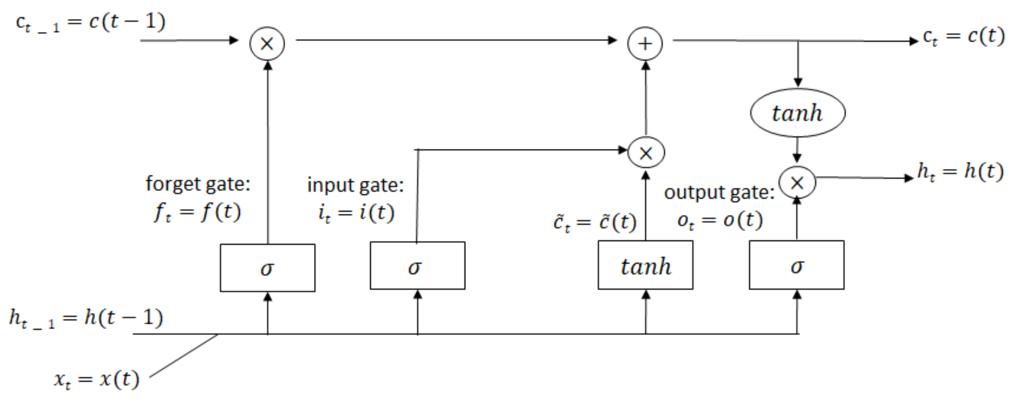


Figure 3.5: LSTM cell overview from [232].

- Input gate;
- Output gate;
- Forget Gate.

First step is that the forget gate manages how much information from the current input and the output of the previous cell is kept. The notation from Figure 3.5 is used and the explanations follow [232]. Let $x(t)$ be the current input for timestep t and $h(t-1)$ the hidden representation from the previous step; let σ be an sigmoid activation function. Let f be the layer transformation that is parameterized by W_f. Then, the forget gate/layer output $f(t)$ is defined as (in form of a neural network layer output with sigmoid activation function)

$$f_t := f(t) = \sigma(W_f \cdot [h(t-1), x(t)] + b_f) \tag{3.5}$$

where W_f as parametrization can be expressed as weight matrix for the network layer and b_f is the bias term.

The input gate has the task to decide what new information to consider. This is an update of the so called cell state. A sigmoid layer gives the input what values should be updates and a tanh layer gives candidates for the new values to be added to the state. Let i be the sigmoid layer from the input gate parameterized by W_i and the bias b_i. Let $\tilde{C}$ be the tanh layer to form the input gate parameterized by W_c and bias b_c. Then, the input gate/layer outputs $i(t), \tilde{C}(t)$ are defined as (in form of a neural network layer output with sigmoid and tanh activation function)

$$i_t := i(t) = \sigma(W_i \cdot [h(t-1), x(t)] + b_i); \tag{3.6}$$

$$\tilde{C}_t = \tilde{C}(t) = tanh(W_c \cdot [h(t-1), x(t)] + b_c). \tag{3.7}$$

The new cell state $C(t)$ is then computed by multiplying the old state $C_t := C(t-1)$ with the sigmoid output from the forget gate plus the new values as output from the sigmoid layer from the input gate scaled by the tanh layer output from the forget gate.

$$C_t := C(t) = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t. \tag{3.8}$$

The final output is a filtered version of the cell state. Let the output gate be defined as a sigmoid layer o parameterized by W_o and bias b_o. The sigmoid layer is used to decide the parts that are finally used. A $tanh$ activation function is applied to the cell state to decide on the values that are outputted. The final selection for the output is then computed via

$$o_t := o(t) = \sigma(W_o \cdot [h(t-1), x(t)] + b_o) \tag{3.9}$$

$$h_t := h(t) = o_t \cdot tanh(C_t). \tag{3.10}$$

For a successful application in the area of gene sequence classification see Hochreiter et al. [90]. A variation of LSTM by Cho et al. [39] (see Chung et al. [43] for a first empirical evaluation) uses Gated Recurrent Unit (GRU) that significantly can speed up the training since it has a reduced number of gates (reset and update).

Neural Network Training

A neural network consists of layers that contains a number of connected neurons. A neuron is a parametrization of a mathematical function. It has an input (input vector) and weight (weight vector plus bias) that are learned during training. Optional a activation function can be added. Therefore a layer consisting of neurons is a combination of inputs and weights - vector multiplication meaning linear combination of single vector entries - and a nonlinear function. In a neural network with multiple stacked layers the neurons are connected hence the linear functions are cascaded. Training the neural network means changing the weights to get an optimized approximation of the true underlying (unknown) functional relationship between input and output. The training starts with randomly initialized weights. To evaluate the performance of the neural network a loss function is selected. It compares the neural network output with a predefined target value. More details and examples for loss functions are given in Chapter 7.

Gradient Descent Different gradient-based optimization methods are applied in the training of DL methods. Stochastic Gradient Descent (SGD) (Ruder [168]), RMSprop by Hinton [89] and ADAM by Kingma and Ba [113] can be named as most common ones (Wilson et al. [215]) that are contained in all state-of-the-art deep learning library (for example keras [40]). ADAM is explained further since it is applied in Chapter 8, 9 and 10. For all other methods we refer to the cited relevant literature.

ADAM is a stochastic gradient-based optimization algorithm that combines the advantages of RMSprop and AdaGrad by Duchi et al. [57]. Let f be the objective function that is parameterized by θ. The goal is to minimize the expected value of f with regard to it parameters θ. The function is evaluated at t (or on a batch/minibatch of multiple steps $t \in [1, \dots, T]$). It is assumed that f is differentiable towards θ and the gradient of f is defined as the vector of partial derivatives of θ at step t. The parameter update is computed using moving averages via the following update rules ([113]):

$$\text{Get gradients at step t: } g_t = \nabla_\theta f_t(\theta_{t-1}); \tag{3.11}$$

$$\text{Update biased first moment estimate: } m_t = \beta_1 \cdot m_{t-1} + (1-\beta_1) \cdot g_t; \tag{3.12}$$

$$\text{Update biased second raw moment estimate: } v_t = \beta_2 \cdot v_{t-1} + (1-\beta_2)(g_t)^2; \tag{3.13}$$

$$\text{Compute bias-corrected first moment estimate: } \tilde{m}_t = m_t/(1-\beta_1^t); \tag{3.14}$$

$$\text{Compute bias-corrected second raw moment estimate: } \tilde{v}_t = v_t/(1-\beta_2^t); \tag{3.15}$$

$$\text{Update parameters: } \theta_t = \theta_{t-1}\alpha \cdot \tilde{m}_t/(\sqrt{\tilde{v}_t} + \epsilon); \tag{3.16}$$

where θ_0 is the randomly initialized parameter vector, α defines the stepsize and $\beta_1, \beta_2 \in [0, 1)$ describes the exponential decay rates for the moment estimates. The bias-corrected estimates are computed to balance out the towards zero biased estimates ($m_0 = 0, v_0 = 0$). The algorithm runs until convergence. More details on the stepsize that is an important property of ADAM including upper bounds are discussed in [113].

Backpropagation A key element of successfully training deep learning models with stacked layers is the backpropagation algorithm. It was first introduced by Rumelhart et al. [169]. It is a special version of a gradient based optimization approach. To address all the neurons representing parameters that are involved, partial derivatives with relation to the model parametrization θ are computed applying the *chain rule*. The chain rule is used to compute the derivative of a function that can be expressed as composition/product of two simpler functions. For further details and explanations see Bishop [20], chapter 5. For a bayesian focused look onto neural network training see MacKay [143].

Hyperparameter Tuning Hyperparameter optimization is done on all models (including benchmarks) to select best suitable hyperparameters and to maximize model training and performance - the model itself is trained with respect to the model parameters θ after selecting and fixing hyperparameter λ. Therefore a p-dimensional - p is the number of parameters to be set - search/parameter space Λ is defined where each of the p hyperparameter spans one dimension of the space. Each point in the space corresponds to one specific model configuration. Let $x \in X$ be samples from data set X following the distribution D_x. Then, the search for the best model configuration $\lambda^{(*)}$ can be defined as optimization problem [16] for $\lambda \in \Lambda$:

$$\mathbb{E}_{x \sim D_x}[L(x, A_\lambda(f, \theta))] \tag{3.17}$$

λ the hyperparameter configurations, $\lambda \in \Lambda$ as search space and A_λ the actual learning algorithm for fixed λ that optimizes f parameterized by θ for each fixed parameter setting λ.

In general, a data set used for solving a modeling task is split in three parts: training data used to train the learner and learn the model parameters θ, test data used for evaluation when model training is finalized and parameters are fixed and for additional parameter λ search in the hyperparameter optimization part, the third so called validation data set is used. Therefore, the expectation value over D_x is approximated by the mean of a validation set S_v, in general better known as cross-validation technique. Then

$$\lambda^{(*)} = \arg\min_{\lambda \in \Lambda} \text{mean}_{x \in S_v} [L(x, A_\lambda(f, \theta))] \tag{3.18}$$

This is a valid method if S_v is independent of the data sample used for training model parameters θ respective data used by A_λ.

For an efficient hyperparameter search, Λ as search space is restricted. There are different strategies available to select trial points $\{\lambda_1, \ldots, \lambda_S\}$ for fast convergence and best λ selection: Grid Search is often used because of its simplicity. A grid is defined on the search space and each point is evaluated in an iterative manner. If the parameter space is real-valued or unbounded, discretization including bounds is recommended. Hence it is best suited for small search spaces to avoid high computational costs. Stochastic Search uses a stochastic distribution for trial points selection; Random Search for example selects them randomly, iteratively evaluate them and best performing parameters are chosen at the end. It should be preferred when the search space is large e.g. for ANN or XGB.

3.2 Generative Adversarial Networks (GAN) and Adversarial Training Approach

The following descriptions assume a basic knowledge about probability and information theory including maximum likelihood estimation (MLE) and divergence metrics. An introduction on those topics can be found in Goodfellow et al. [74] (see Chapter 3 for a solid understanding of basic concepts and Chapter 5 for MLE), other well-known reference are MacKay [143] and Bishop [20] mentioned before. More details on distribution-based losses that are based on divergence measures is given in Chapter 7.

Probability distance metrics

Main property of generative models like GAN is the generation, comparison or adaptation of probability distributions. Selection of metrics highly influences possible numerical approximation and upper or lower bounds and drawn convergence conclusions. To cover a distance between probability measures, two classes of probability distance metrics are considered here: F-divergences and integral probability metrics (IPM). Based on Sriperumbudur et al. [185] the following general class definitions are given. First F-divergences is defined via:

$$D_\phi(\mathbb{P},\mathbb{Q}) := \begin{cases} \int_M \Phi\left(\frac{d\mathbb{P}}{d\mathbb{Q}}\right) d\mathbb{Q} & \mathbb{P} \ll \mathbb{Q} \\ \infty & \text{otherwise.} \end{cases} \tag{3.19}$$

where $\mathbb{P}, \mathbb{Q}$ are two probability measures, M is a measurable space, $\Phi : [0,\infty) \to (-\infty,\infty]$ and $\mathbb{P} \ll \mathbb{Q}$ describes that $\mathbb{P}$ is absolutely continuous w.r.t. $\mathbb{Q}$. Examples used in GAN or DANN settings from the class of F-divergences are

- Kullback–Leibler (KL) with $\Phi(t) = t\log(t)$;
- Jensen–Shannon (JS) a smoothed and symmetric version of KL.

integral probability metrics (IPMs) are defined by:

$$\gamma_F(\mathbb{P},\mathbb{Q}) := \sup_{f\in F}\left|\int_M f d\mathbb{P} - \int_M f d\mathbb{Q}\right| \tag{3.20}$$

where F is a class of real-valued bounded measurable functions on a measurable space M. Examples used in GAN or DANN settings from the class of IPMs are

- Maximum Mean Discrepancy (MMD) with $F = \{f : \|f\|_H \neq 1\}$, H reproducing kernel hilbert space;
- Wasserstein with $F = \{f : \|f\|_L \neq 1\}$, L Lipschitz semi-norm.

Generative Adversarial Networks (GAN)

Generative Adversarial Networks (GAN) are first introduced by Goodfellow et al. [75]. The goal is to generate synthetic data samples. A GAN consists of two models, a discriminator D and a Generator G. The generator's task is to create data samples as close as possible to a given data distribution and the discriminator's task is to classify generated versus original samples. The generator tries to generate samples that the discriminator cannot distinguish from the original anymore meaning label them as original. The discriminator ties to detect the synthetic data samples. Compared to other generative approaches it introduces a two-player game approach for training where the two parts D and G of the network pursue opposite goals. The training happens iterative with a certain ration and each player is only able to update its own parameters during training. The used minimax is inspired by game theory. A global loss is defined based on difference of expectation of true versus generated data distributions and it can be shown that minimizing the D loss (G parameters are fixed) is equivalent to minimizing Jensen–Shannon divergence (JSD) that is based on the Kullback–Leibler divergence (KLD). More theoretical/mathematical insights about training dynamics for GAN including for example instability and choice of divergence metrics is given by Arjovsky and Bottou [5].

Let X be defined as the input space and Y as the output space. Let $h_D : X \to Y$ define the domain discriminator meaning a statistical model for classification of two domains. Let h_D be parameterized by θ_D. Let $D(x, \theta_D)$ be the parameter representation of the domain classifier where D is the model function with parameters θ_D that outputs the prediction for $x \in X$. Let $h_G : Z \to X$ be the statistical model function mapping a second data space Z to X and $G(z, \theta_G)$ be its parameterized representation where G is the model function with parameters θ_G that outputs the prediction for $z \in Z$. Let D_{data} be the data generating distribution (identical to what is called source domain for domain adaptation) based on X and D_z a noise distribution (identical to what is called target domain for domain adaptation) based on Z. Using the definition of the metric for JSD as basis for defining a loss function for a GAN network and the introduced notation, the optimization problem respective the value Function V is presented as follows:

$$V(D, G) = \mathbb{E}_{x \sim D_{data}}[\log D(x)] + \mathbb{E}_{z \sim D_z}[\log(1 - D(G(z)))], \tag{3.21}$$

with $\mathbb{E}$ the expectation value as defined by measure theory. For detailed theorems and proofs see Goodfellow et al. [75].

One of the first and very well known example of a successfully implemented GAN based on convolutional layers called DCGAN is presented by Radford et al. [163]. It applies the idea of unsupervised representation learning and its advantages compared to original GAN comes from a set of carefully selected constraints on the chosen architecture.

Wasserstein Generative Adversarial Networks (WGAN)

Difficulties like instability and mode collapse can occur during GAN training. Hence so called Wasserstein GAN (WGAN) is introduced by Arjovsky et al. [6]. Compared to the original GAN setting, it makes use of the Lipschitz property of Wasserstein distance - 1-Wasserstein distance is also called earth mover's distance (EMD). It is stable meaning well behaved in the sense that if the input changes a little, the output also does not change heavily. Instead of a binary classification using sigmoid as output function), the discriminator D outputs continuous scores for realness or fake. The linear output function allows values that go to infinity respective minus infinity in order to differentiate. The sensitivity towards the training ratio of D and G does no longer exists and unlike before one explicitly wish to train D to its optimum since vanishing gradient due to zero loss when coming too close to the optimal D is no longer a problem. Using the Kantorovich-Rubinstein duality for the Wasserstein metric as basis for defining a loss function for a GAN network, the optimization problem respective the value Function V is presented as [6]:

$$V(D,G) = \mathbb{E}_{x \sim P_{data}}[D(x)] + \mathbb{E}_{z \sim P_z}[(D(G(z)))], \tag{3.22}$$

with D and G being discriminator and generator as described before, $\mathbb{E}$ the expectation value as defined by measure theory, P_{data} is the data generating distribution and P_z the prior noise distribution.

By definition of the loss, D needs to fulfill the properties of a set of 1-Lipschitz functions. As presented in Arjovsky et al. [6] this can be done by so called weight clipping that restricts the maximum weight values. Tuning the values for the clipping is still a challenge and bad choices lead to poor convergence and low quality outputs.

With further improvement of WGAN presented in Gulrajani et al. [78] (WGAN-GP) the problems arising with weight clipping for standard WGAN and overfitting in the original GAN are addressed. A regularization term called gradient penalty (GP) is introduced that regularizes the gradient during training instead of weight clipping where poor settings for the corresponding hyperparameter can lead to decaying (selected too small) or exploding (selected too large) loss again. In addition, WGAN-GP assures that if D struggles to distinguish between real and generated samples, the generated synthetic data is truly similar to the targeted one. In detail: The Lipschitz constraint is covered either by applying weight clipping as described in [5] or by introducing gradient penalty (GP) regularization term as described in [78]. GP is defined as:

$$GP = \lambda_{GP} \cdot \mathbb{E}_{z \sim P_z} \left[(\|\nabla_z D(G(z)\|_2 - 1)^2 \right]. \tag{3.23}$$

with λ_{GP} as weight factor for the total loss. As presented by Arjovsky et al. [6] and Gulrajani et al. [78], the Wasserstein based GAN shows convenient behavior towards the gradient of the discriminator with favor of the gradient penalty that shows the most stable training.

Cycle Generative Adversarial Networks (CycleGAN)

GAN is a powerful method and heavily used for different applications and very present in literature especially for computer vision task. A specific GAN construction that enables unpaired sample matching is the so called CycleGAN presented in Zhu et al. [239]. It targets domains where no 1:1 matching of samples is possible. Therefore a second generator and a second discriminator is introduced and a cycle consistency loss is formulated comparing input-recreated input pairs instead of input-output pairs. Another optional loss is the identity loss assuring that the generator is close to an identity mapping in cases where real data meaning data that does not need a distribution shift is given as input. Further improvements of CycleGAN are now available like for example Xie et al. [221] introducing a self-supervised aspect in addition and Chen et al. [32] that includes attribute transfer for improved style transfer for more complex images. A combination of domain adversarial methods like presented in the following section and the CycleGan for an image semantic segmentation task is presented in Hoffman et al. [92].

DBACS, the method introduced in Chapter 5, is based on that idea and gives the opportunity to apply domain adaptation on domains with different representations.

3.3 Domain Adaptation Theory and Domain Adversarial Neural Networks (DANN)

Domain adaptation (DA) is a sub field of transfer learning (TL) developed to transfer a well performing dedicated statistical model trained on a specific data distribution to a second similar but non-identical distribution. So called homogeneous DA assumes that the feature space have the same dimensions and features, while in heterogeneous DA the features space differ resulting in different space dimensions.

Transfer Learning

Transfer learning (TL) is the idea to use already existing knowledge about data, modeling task as well dedicated statistical model to achieve better and faster results for the same or similar modeling task for another related respective biased data set. An extensive overview of transfer learning and its versatile aspects is given by Zhuang et al. [241]. In literature one can find different paradigm of TL, here a deep learning focused categorization given by Bashath et al. [10] is presented:

- Instance-based: re-weighting instances by minimizing distribution distance;
- Mapping-based: finding a common (lower dimensional) space with greater similarity by optimizing a mapping function;
- Network-based: reusing weights for network initialization;
- **Adversarial-based**: find domain invariant (latent) features using an adversarial training approach.

In this work the focus is on homogeneous as well as heterogeneous adversarial-based DA.

Domain Adaptation Theory

In order to assure generalization of a trained model, it is assumed that train and test data come from the same underlying distribution. If this is not the case - meaning that data sets show differences beyond just noise while the underlying relevant information is the same - *domain adaptation theory* overcomes resulting generalization issues as well as scaling limitations.

Based on the mathematical formalism of statistical learning theory - see Vapnik [207] for a general introduction including detailed descriptions - as well as computational learning theory - for an introduction see Kearns and Vazirani [111] -, theoretical discussions about generalization are done by Baxter [11] in the context of single versus multi task training of neural networks. It shows advantages of representations learned by *multi task training* for model transfer to additional tasks and advantages about the number of necessary training samples that are needed. Based on those findings, Ben-David and Schuller [15] states the mathematical formulation of *task relatedness* including error bounds for *multi task learning* as well as mathematical proofs of its situation specific advantages concerning generalization. Related work but with focus on noise - respective an algorithm able to deal with noisy data - in a multi task setting is presented by Crammer et al. [46].

While mentioned results rely on the one hand on labeled data as well as on the other hand on a given measure of *similarity* and task relatedness, Ben-David et al. [14] addresses the situation where the similarity of two different data sets with partly missing labels is in focus.

Definition 3.1 *Let F define a modeling task, X be defined as the input space and Y as the output space. A domain is defined as a distribution over $X \times Y$: in more details as a pair consisting of a distribution over the input space X as well as a modeling function $f : X \to Y$. Then, a supervised learning algorithm is provided with a labeled sample set S drawn i.i.d. from a domain D_S where*

$$S = \{x_i, y_i\}_{i=1}^n \sim \{D_S\}^n, \tag{3.24}$$

with $n = |S|$ being the number of drawn samples.

The theoretical generalization properties described above (easier model transfer, less training samples) for example multi task learning for one selected domain does not hold when the trained model is applied to a second domain based on data samples following a different distribution. Let now D_S describe the original domain - called from now on *source domain* following the naming conventions of domain adaptation theory - used for the model training. Let T be a second labeled sample set drawn i.i.d. from a different domain D_T over $X \times Y$ with a different distribution in case of labeled samples and D_T^X over X with a different marginal distribution in case of missing labels. D_T (D_T^X) is called *target domain*.

Then [14] states a generalization error bound of a classifier on the target domain D_T when transferred from a source domain D_S. Furthermore, Ben-David et al. [13] extents this result in the direction of semi-supervised learning: It describes settings and properties in order to optimize the accuracy for both source and target while considering the amount of available labels and overall sample sizes. For more insights into useful measure for divergence between source and target is we refer to the given literature.

Domain Adversarial Neural Networks (DANN)

Neural Networks are heavily used models in the field of domain adaptation since they characterize as a data driven and data centric approach. Ganin et al. [68] first present a neural network architecture used for domain adaptation: *domain adversarial neural network*

(DANN). DANN including its adversarial training algorithm is applicable to both unsupervised as well as semi-supervised (or totally supervised) training. [68] provides theoretical explanations and mathematical formalism based on [14], [13]. Here a quick overview is given:

First the paper recaps the results given in [13]: The $\mathcal{H}$-divergence $d_{\mathcal{H}}$ between source and target domain is defined as metric. $d_{\mathcal{H}}$ is based on a hypothesis class $\mathcal{H}$, meaning a set of all possible modeling functions that are considered. The empirical H-divergence $\hat{d}_{\mathcal{H}}$ can be computed using two sample sets from source respective target domain using the discrete representation. Then an approximation of $\hat{d}_{\mathcal{H}}$ is computed using properties of a chosen learning algorithm respective its model error. Finally, using upper bounds for $\hat{d}_{\mathcal{H}}$, properties are given under which the target risk- the probability that the prediction for input samples from target domain is different from the corresponding respective paired output label - can be kept small.

Ganin et al. [68] conclude that in order to enable domain generalization one needs to find a common feature space for source and target. The common feature space has the following properties: The original domain differences do not appear anymore while the source risk is kept small.

Therefore, the idea of DANN is to map domain specific input features to a domain-invariant latent feature space while keeping the necessary task specific information available. First case studies including synthetic and real world data are presented [68]. The DANN architecture plus the training routine (see [68], Chapter 4.2) can be summarized as follows:

Let $F : X \rightarrow \mathbb{R}^m$ be a function that maps the input space X into a domain invariant representation space with the dimension m and let $F(\cdot, \theta_F)$ be the representation of the neural network with parameters θ_F, let $P : \mathbb{R}^m \rightarrow Y$ the model function for prediction or classification with the output space Y. Let $P(\cdot, \theta_P)$ be the representation of the neural network with parameters θ_P. Last, let $D : \mathbb{R}^m \rightarrow [0, 1]$ the model function for classification of two domains with the labels 0 for source and 1 for target and $D(\cdot, \theta_D)$ be the representation of the neural network with parameters θ_D. Let the prediction loss and the domain classification loss called **adversarial loss** be defined as

$$\mathcal{L}_P^i(\theta_F, \theta_P) := \mathbf{L}_P(P(F(x_i, \theta_F), \theta_P), y_i) \tag{3.25}$$

$$\mathcal{L}_D^i(\theta_F, \theta_D) := \mathbf{L}_D(D(F(x_i, \theta_F), \theta_D), d_i) \tag{3.26}$$

with $d_i \in [0, 1]$ describing the domain label and $(x_i, y_i) \in X \times Y$ and where $\mathbf{L}_{(\cdot)}$ are selected loss functions. Then the optimization objective is defined as

$$E(\theta_F, \theta_P, \theta_D) = \frac{1}{n}\sum_{i=1}^{n} \mathcal{L}_P^i(\theta_F, \theta_P) - \lambda \left(\frac{1}{n}\sum_{i=1}^{n} \mathcal{L}_D^i(\theta_F, \theta_D) + \frac{1}{n'}\sum_{i=n+1}^{N} \mathcal{L}_D^i(\theta_F, \theta_D \right). \tag{3.27}$$

with a labeled sample set S drawn i.i.d. from a domain D_S where $n = |S|$ being the number of drawn samples and another unlabeled sample set T drawn i.i.d. from a marginal domain D_T^X where $n' = N - n + 1 = |T|$ being the number of drawn unlabeled samples. The function is solved using an adversarial training approach such that

$$(\tilde{\theta}_F, \tilde{\theta}_P) = \underset{\theta_F, \theta_P}{\operatorname{argmin}} E(\theta_F, \theta_P, \tilde{\theta}_D) \tag{3.28}$$

$$\tilde{\theta}_D = \underset{\theta_D}{\operatorname{argmax}} E(\tilde{\theta}_F, \tilde{\theta}_P, \theta_D). \tag{3.29}$$

[68] uses a gradient reversal layer (GRL) for tackling the maximising part of the optimization problem using gradient descent. A similar approach called adversarial deep averaging networks (ADAN) but with an stability advantage is presented in Chen et al. [35]: The used architecture is the same but the paper applies an approximation of the Wasserstein-distance including weight clipping for the domain discriminator loss similar to Arjovsky et al. [6] for GANs. This increases not only the training stability for their language sentiment classification use case but makes the usage of the GRL redundant. More details on adversarial training are presented in Chapter 3.2.

Based on the presented work by [68], lots of related methods and architectures are introduced including all kinds of applications: An up-to-date survey focused on visual applications is presented by Zhao et al. [238]. The work by Berthelot et al. [18] with the method called *AdaMatch* that makes use of pseudo-labels can be considered state-of-the-art of UDA for computer vision related use cases. More work is done in the field of semantic segmentation (see Zou et al. [242] for example) as well as object detection (see the survey presented by Oza et al. [154]).

One of the first works of domain adaptation in NLP for a cross lingual sentiment classification is presented in Chen et al. [35] also using Wasserstein loss and weight clipping as introduced in Arjovsky et al. [6]. Wang et al. [211] put powerful state-of-the-art transformer architectures used for NLP in relation to adversarial training and combines the two approaches for increased model robustness.

Time series data and regression problems are underrepresented in domain adaptation compared to computer vision and classification tasks. Nevertheless Farshchian et al. [62] introduces not only domain adaptation for both regression and time series but also considers a different distribution based metric by evaluating the distribution of the residuals of original versus reconstructed time series data. Therefore it extents the work from Warde-Farley and Bengio [214] by using autoencoder inspired architectures for the discriminator. Another important publication on domain adaptation for time series data is Purushotham et al. [162]

Supervision in DANN

Generative models like DANN combined with adversarial training are especially designed for unsupervised or semi-supervised learning settings where labels for source data are available but none or only a few for target. Overall there are different so called data shifts that can occur in a DA setting, for an extensive summary we refer to Pan and Yang [157]. Most of the publications about DA assume covariate shifts related to the input space meaning either marginal or conditional shift occurs in the distributions. Hence, methods are often set up in a way that the target is aligned according to the source label distribution. In cases of so called target shift - for example due to class imbalance - this can lead to poor performance. For an early discussion on so called distribution alignment see Zhang et al. [236] and for a more recent discussion on *"...multi domain adaptation under target shift"* based on optimal transport see Redko et al. [165].

In the supervised settings the prediction or classification loss of target labels automatically takes care that alignment is happening according to class distributions by comparing the pseudo target label distribution (the predicted labels are called pseudo labels) with the real target label distribution. Hence class imbalance for target domain is already indirectly considered. If no target labels are available the target label distribution needs to be approximated as closely as possible. Therefore, in the optimal case, the target pseudo label distribution fits the true target label distribution. Selecting the available source label distribution is an obvious always available choice for distribution alignment and often improves adaptation results. A survey targeting UDA is given by Wilson and Cook [216] structuring approaches on the applied methods and comparing them. State-of-the-art UDA presented by Berthelot et al. [18] takes a factor based on expectation value of source label distribution if no labeled target information is available into account. Nevertheless, if one expects significant differences between source and (unknown) target label distribution those can still lead to poor performance. Methods that are available for approximating the target label distribution prior to the alignment (besides source data and expert knowledge) are given by Jiang et al. [103] for example. For a more theoretically focused paper see Tachet des Combes et al. [197]. An interesting dimension to this topic is introduced by Yang and Xu [225] discussing class-imbalanced learning in a self-supervised setting. An improvement of CycleGan using self-supervised learning seems as well a powerful tool for improving domain adaptation results as presented in Xie et al. [221].

4 DBAM

In this chapter we describe the DANN-based Alignment Model (DBAM) based on the basics, descriptions, figures, graphs and notations introduced in Section II. DBAM is presented and published in Gentner et al. [70] and Gentner et al. [69], where all explanations are targeted towards a specific VM application use case. Here, we address the theoretical part of DBAM independently of any concrete use case. Therefore, even if the presented methodology is inspired by the needs, highest standards and complexity of semiconductor manufacturing, the following mathematical explanations of all methods and their usage are independently of any application.

4.1 DANN-based Alignment Model (DBAM)

The idea of the so called DANN-based Alignment Model (DBAM) is to create one common model for multiple datasets showing different behavior. Therefore it addresses settings where

- different data distributions occur;
- no direct model transfer of a dedicated model is possible;
- a generalized model would suffer accuracy decay.

The main purpose of DBAM is to maintain the high accuracy of a dedicated model while keeping all necessary data information in order and allowing interpretability and comparison of all data sets and features involved.

Therefore, the idea of Domain Adversarial Neural Networks (DANN) by Ganin et al. [68] is exploited and combined with a domain adaptation alignment approach using a residual inspired setting as presented by Farshchian et al. [62]. In addition, the training approach is adopted from Gulrajani et al. [78] including gradient penalty (GP) regularization and Wasserstein loss. The main contributions of the DANN-based Alignment Model (DBAM) with regards to related literature can be summarized as follows:

- a novel approach that can cope with high context complexity respective multiple systems as well as a large variety of data types like stationary feature based data, time-series data and image data;
- able to tackle supervised, semi-supervised and unsupervised regression as well as classification tasks for two (and more) domains following different distributions while assuring interpretability and flexibility towards input dimensions and feature representations of different domains;
- applicable to broad range of (semiconductor) industry applications independent of task (regression or classification) and availability of labels.

The following notations are used throughout the whole section. Let f define a modeling task, let X be defined as the input space and Y as the output space. In a typical scenario where only numerical input data is present, the feature space is selected as $X \subset \mathbb{R}^N$. In case of categorical data, text data or image data a function called embedding $E : I \to X$ is defined that maps the set of instances I to a numerical feature space X. For time series data we define $X \subset T \times \mathbb{R}^N$ where T describes the set of considered points in time and $x_t \in \mathbb{R}^N$ a sample from the feature space taken at a fixed point in time $t \in T$. For a supervised modeling task - semi-supervised and unsupervised settings are discussed in the DC use case in Chapter 10 - the output space Y is defined as $Y = \{0, 1\}$ in case of a binary classification respective $Y = \{0, 1\}^c$ in case of a categorical classification with c classes and $Y \subset \mathbb{R}$ in case of a regression task (assuming data is already normalized). A *domain* is defined as a distribution over $X \times Y$: in more details as a pair consisting of a distribution over the input space X as well as a modeling function. Let a hypothesis class $\mathcal{H}$ be a set of all possible modeling functions h that are considered for a specific task.

First a statistical task model $h_P : X \to Y$ is build so that the modeling error L_P defined on a metric L suitable for regression or classification is minimized. Hence a supervised learning algorithm is provided with a labeled sample set S drawn i.i.d. from a domain D_S where

$$S = \{x_i, y_i\}_{i=1}^{n} \sim \{D_S\}^n, \tag{4.1}$$

with $n = |S|$ being the number of drawn samples and D_S is the so called source domain. Then, the **prediction loss** L_P for source is defined as

$$L_P(X) := L_{P,D_S}(X) = \mathbf{L}_{(x,y)\sim D_S}\left(h_P\left(x\right), y\right). \tag{4.2}$$

with $\mathbf{L}$ defining a selected loss function suitable for supervised learning based on the modeling task at hand. The name source is chosen based on the naming conventions of domain adaptation theory - for a detailed definition including useful properties see Chapter 3.3.

Let h_P be parameterized by θ_P and $P(x, \theta_P)$ be the representation of our statistical model from input to output where P is the model function with parameters θ_P that outputs the predicted target value for $x \in X$. Then the optimization problem used to train P on source in order to minimize the parameterized error function L_P can be formulated as follows:

$$\min_{h_P\in\mathcal{H}} L_P(X) = \min_{h_P\in\mathcal{H}} L_{D_S}(h_P(X)) = \min_{\theta_P} L_{D_S}(X, \theta_P) = \min_{\theta_P} \mathbf{L}_{(x,y)\in S}\left(P\left(x, \theta_P\right), y\right). \tag{4.3}$$

In DA scenarios, at least two data sets are given. Let T be a second labeled sample set following a distribution D_T called *target domain*. In the SSL setting, it is distinguished between labeled and unlabeled data: Let $T = TL \cup TU$ be the second data set T drawn i.i.d. from target domain D_T with a distribution over $X_T \times Y_T$, $X_T \subset X, Y_T \subset Y$, and consisting of unlabeled TU and/or labeled TL samples.

$$TL = \{x_T^j, y_T^j\}_{j=1}^{m-l} \sim \{D_T\}^{m-l}; \tag{4.4}$$

$$TU = \{x_T^j\}_{j=m-l+1}^{m} \sim \{D_T^X\}^{l}; \tag{4.5}$$

with m being the number of drawn target samples. Let D_S^X, D_S^Y and D_T^X, D_T^Y be the marginal distributions of D_S and D_T over X_S, Y_S and X_T, Y_T respectively. Then:

- $D_S^X = D_T^X$ and $D_S^Y = D_T^Y \Rightarrow$ no transfer needed
- $D_S^X = D_T^X$ and $D_S^Y \neq D_T^Y \Rightarrow$ label or target shift
- $D_S^X \neq D_T^X$ and $D_S^Y = D_T^Y \Rightarrow$ marginal or conditional shift

- $D_S^X \neq D_T^X$ and $D_S^Y \neq D_T^Y \Rightarrow$ strong or generalized shift

Here the focus is on cases where the same underlying learning task is considered. DBAM is generated for dealing with marginal and conditional shift for homogeneous input space representation where a direct model transfer is not possible. For the supervised case involving labels for both source and target domain automatically handles target shift and assures correct causal alignment with target prediction loss being part of the overall optimization objective. A generalized shift would be a class-imbalanced domain adaptation task. Further discussion are given in the context of the affected use case, Chapter 10: Defect Classification (DC). Further assessment related to unsupervised or semi-supervised domain adaptation under target or generalized shift hence discussion on published methods and ideas how to deal with label shift using distribution alignment especially on top of conditional shift is also presented in Chapter 10.

The DBAM approach, as visually presented in Figure 4.1, consists of three parts. Each one corresponds to a specific part of the overall DBAM optimization objective:

- the baseline or reference *prediction* model P, already introduced at the beginning of this section;
- an encoder/alignment model called *aligner* A used to map the target domain to the source domain. The output of the aligner is called aligned. It enables the usage of a dedicated prediction model for aligned target data;
- a domain *discriminator* D for classification of source and target domain. Its loss is used for the adversarial training approach. Different output activation functions are necessary, depending on the specific choice of the loss function.

The predictor model itself is a dedicated model trained only on source data. The weights are frozen after successful training. The goal is to reuse the dedicated model as predictor for aligned target data in order to keep the high source prediction accuracy while enabling model usage under comparable prediction accuracy for aligned target data.

Let $h_D : X \rightarrow I$ - where $I \subset \mathbb{R}$ or $I = \mathbb{R}$ depending on the objective and implementation - be a domain discriminator meaning a statistical model for binary classification of source and target domain. It can be modeled as a domain classifier that has a high positive or high negative scalar value as output indicating the corresponding input domain (source or target) or as classical binary or multi categorical classification model. It is the first competitor

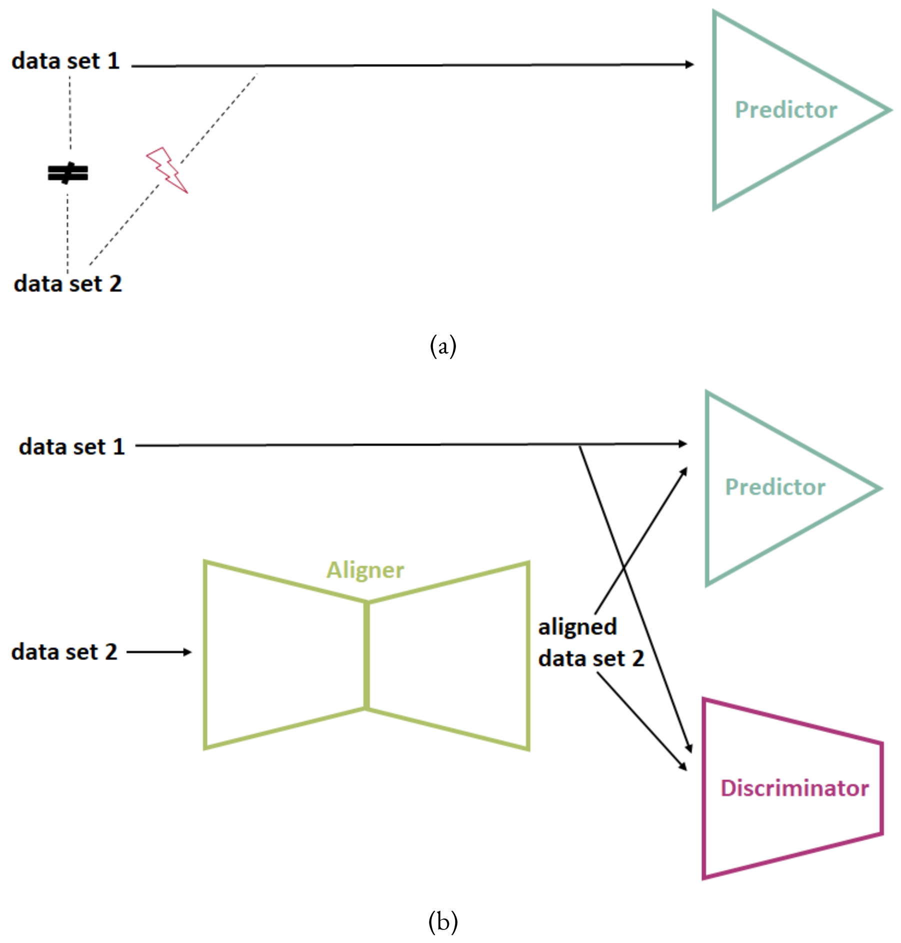


Figure 4.1: Visualization of DBAM [69]. Graphical representation of DBAM (panel (b)) exploiting input data from two different domains where a direct model transfer is not possible (panel (a)). The arrows represent data flow during forward propagation.

of the adversarial training that is applied. Let h_D be parameterized by θ_D. Let $D(x, \theta_D)$ be the parameter representation of the domain classifier where D is the model function with parameters θ_D that outputs the prediction for $x \in X$.

An encoder model called aligner as second competitor of the adversarial training is considered. The aligner is the main part of DBAM where the actual feature mapping happens. Let $h_A : X \to X$ be the statistical model function aligning target to source and $A(x, \theta_A)$ be its parameterized representation where A is the model function with parameters θ_A that outputs the prediction for $x \in X$.

Using discriminator D and aligner A, the **adversarial loss** L_{adv} is defined as

$$\begin{aligned} L_{adv}(X) &:= L_{D,D_S}(X) - L_{D,D_T}(X) = \mathbf{L}_{x\sim D_S^X}\left(h_D\left(x\right)\right) - \mathbf{L}_{x\sim D_T^X}\left(h_D\left(h_A(x)\right)\right) \\ &= \mathbf{L}_{x\sim D_S^X}\left(D\left(x,\theta_D\right)\right) - \mathbf{L}_{x\sim D_T^X}\left(D\left(A(x,\theta_A),\theta_D\right)\right) \end{aligned} \tag{4.6}$$

where $\mathbf{L}$ is a selected loss function, based on the training strategy of two competing networks as well as their selected architecture. The adversarial training routine using L_{adv} for training DBAM hence of both aligner and discriminator are described in the following:

- The first competitor of the adversarial training is the discriminator trained to distinguish between source and aligned target data meaning minimizing a classical classification loss. Then, the optimization of the discriminator loss $L_{D_{total}}$ is defined as

$$\begin{aligned} \max_{\theta_D} L_{D_{total}}(X) &= \max_{\theta_D} L_{adv}(X,\theta_D,\theta_A) \\ &= \max_{\theta_D} L_{D,D_S}(X,\theta_D) - L_{D,D_T}(A(X,\theta_A),\theta_D) \\ &= \max_{\theta_D} \mathbf{L}_{x\in S^X}\left(D\left(x,\theta_D\right)\right) - \mathbf{L}_{x\in T^X}\left(D\left(A(x,\theta_A),\theta_D\right)\right) \end{aligned} \tag{4.7}$$

  where S^X, T^X define the part of the drawn sample set S, T that corresponds to the input space X.

- In the supervised setting where labels are also available for the target domain we define L_A using both the prediction as well as the adversarial loss. Hence, (in case of labeled target data) the aligner is on the one hand updated in order to minimize prediction loss L_P also for aligned target domain. On the other hand, the adversarial part of the aligner loss is set in opposite direction compared to the discriminator. If training the discriminator involves a loss minimization with respect to the discriminator parameters, training the aligner means a loss maximization with respect to the update of the aligner parameters and vise versa. Based on equation (4.7):

$$\begin{aligned} \min_{\theta_A} L_A(X) &= \min_{\theta_A} L_{adv}\left(X,\theta_D,\theta_A\right) + \lambda L_P\left(A(X,\theta_A),\theta_P\right) \\ &= \min_{\theta_A} -\mathbf{L}_{x\in T^X}\left(D\left(A(x,\theta_A),\theta_D\right)\right) + \lambda\cdot\mathbf{L}_{(x,y)\in TL}(P\left(A(x,\theta_A),\theta_P\right)) \end{aligned} \tag{4.8}$$

  where $\lambda \geq 0$ is a coefficient that needs to be defined and optimized in order to balance out possible differences in rank and influences between the discriminator and the predictor loss. For $\lambda = 0$ no prediction loss is considered, hence it defines the aligner loss of an unsupervised setting where no target labels are available. A gradient penalty regularization term (for details see Chapter 3.2) is added in addition based on the recommendations of [78] that are followed during training of the aligner.

The training itself happens in an adversarial setting with a two-player game approach: the pseudo-code of the approach is reported in Algorithm 4.1 similar to [69] for unsupervised training with no target labels. For a graphical representation see Figure 4.2 [70].

Algorithm 4.1 DBAM training
Default values to be set: λ, m, ratio, β_1, β_2, lr$_{disc}$, lr_{align}

```
Require: loss weights $\lambda$, the $ratio$ between discriminator and aligner update, batch size m,
    optimizer, hyperparameters $lr$ (learning rate), $\beta_1$, $\beta_2$, $L_P$, $L_D$, $L_A$
Require: pretrained and frozen predictor parameter $\theta_P$, initial discriminator parameter
    $\theta_{D_0}$, initial or pretrained aligner parameter $\theta_{A_0}$
 1: while $\theta_A$ and $\theta_D$ have not converged
 2:   for $t = 1 ... ratio$
 3:     unfreeze $\theta_D$, freeze $\theta_A$
 4:     sample $S_m = \{x\}_{i=1}^m \sim \{D_S^X\}^m$, $T_m = \{z\}_{i=1}^m \sim \{D_T^X\}^m$
 5:     update $\theta_D$
 6:   end for
 7:   freeze $\theta_D$, unfreeze $\theta_A$
 8:   sample $S_m = \{x\}_{i=1}^m \sim \{D_S^X\}^m$, $T_m = \{z\}_{i=1}^m \sim \{D_T^X\}^m$
 9:   update $\theta_A$
10: end while=0
```

The following step-by-step description summarizes the training procedure:

1. First a dedicated prediction model is trained on labeled samples from the source domain. During training the parameters respective weights θ_P are fitted via minimization of the L_P loss. The source domain is selected under consideration of quality, stability, availability of data respective labels and best performance for the corresponding modeling task. For all following training steps the predictor weights are kept frozen in order to preserve the high quality of the dedicated model.

2. The aligner is mapping the target domain to the source domain and is set up in a way that after training it allows the direct comparison of the source domain with the aligned target domain and enables interpretability and further analysis of domain differences. The set up as well as its initialization and the possibility of a pretraining step depends highly on the underlying input structure of source and target. A pretraining is recommended due to better initialization. The aligner is trained on samples from source and target domain based on the discriminator output in the adversarial setting. The parameters respective weights θ_A are fitted via optimization of the L_A loss.

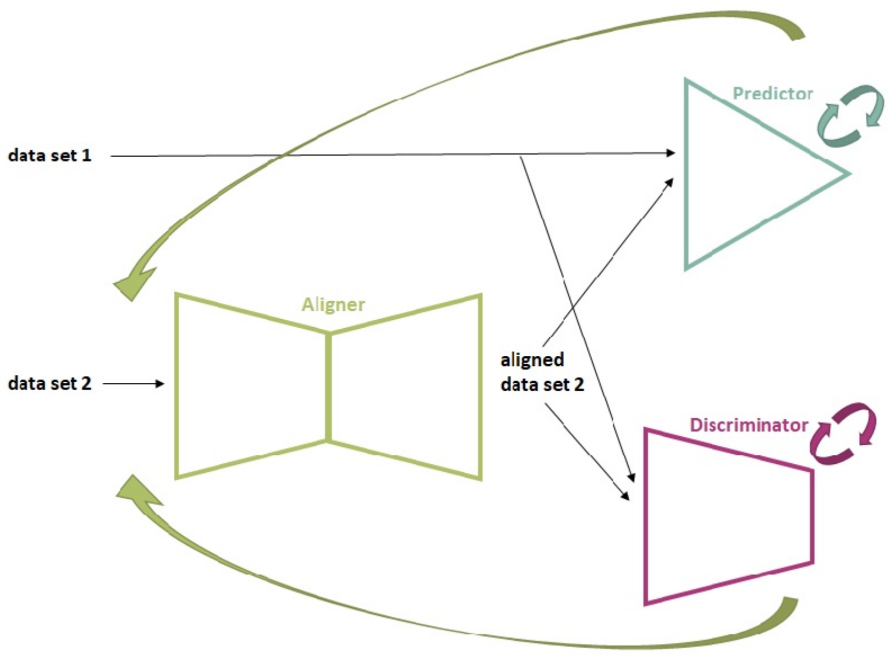


Figure 4.2: Visualization of DBAM training routine. Graphical representation of the proposed training procedure of DBAM exploiting input data from two different domains. The arrows represent the different feedback during training.

3. The discriminator model is trained on samples from source as well as aligned target data and $L_{D_{total}}$ is optimized in opposite direction compared to the aligner loss and its weights θ_D are updated accordingly. In the adversarial setting θ_A are fitted via minimization of the L_A loss that contains (besides the prediction loss for aligned target data in supervised setting) the part of $L_{D_{total}}$ that depends on θ_A while for θ_D the $L_{D_{total}}$ is maximized and θ_D is fitted accordingly. The iterative training is set up with a predefined training ratio between aligner and discriminator update.

Details, specifications and tuning of hyperparameter are data respective use case specific and therefore discussed in the dedicated application chapters.

For an improved training stability it is recommended to follow the approach presented in Gulrajani et al. [78] and define the adversarial loss based on an approximation of the Wasserstein metric:

$$L_{adv}(X) = \mathbb{E}_{x \sim D_S^X}\left[D\left(x, \theta_D\right)\right] - \mathbb{E}_{x \sim D_T^X}\left[D\left(A(x, \theta_A), \theta_D\right)\right] \qquad (4.9)$$

where $\mathbb{E}$ defines the expected value, $x \in X$. For a detailed introduction of Wasserstein loss including related methods and literature see Chapter 3.2.

Different implementation methods are possible for achieving the necessary min-max two player game between discriminator and aligner e.g. a so called gradient reverse layer (GRL) is suggested for DANN [13]. Here, an approach is used that makes use of changing a maximization to a minimization task by reversing the sign of the function to be optimized plus artificial domain labels and linear output function of the discriminator models combined with a distance based loss:

Therefore, an approximation of Wasserstein loss as discriminator loss function as presented above and in section 3.2 is implemented. Hence, the goal of the discriminator is to increase distance/divergence (Wasserstein is a divergence measure) between domain distributions, the goal of aligner is to decrease distance/divergence of domain distributions. Parameter updates of the aligner are done by minimizing Wasserstein distance hence backpropagation of Wasserstein loss as usual. Forward pass of source data is not going through the aligner, therefore does not influence the parameter update and the loss function can be simplified to only minimizing expectation value of aligned target data. For parameter updates of the discriminator the goal is to maximizing Wasserstein distance. Instead of maximizing we can also minimize the negative Wasserstein distance. This is implemented by assigning artifical -1 labels to the samples.

Based on the implementation using linear output activation, Wasserstein has no bounds, meaning we theoretically allow the loss going to $-/+\infty$. Wasserstein loss cannot be interpreted in an absolute manner hence is not usable for direct comparison of GAN/DANN models. As relative distance measure, it highly depends on model configuration and used dataset. It is consistent for a given discriminator model and convergence of the aligner does correlate with better generated data quality.

Negative scores for source and positive scores for target are possible but not mandatory. An offset based on initialization can lead to all positive or all negative scores. The loss function encourages a separation between scores for source and target as larger and smaller, not necessarily positive and negative.

As described in the paper [78] a perfect discriminator using Wasserstein is working well - a discriminator always able to distinguish. As the training continuous, the goal is to make it harder for the discriminator to distinguish between the domains, therefore the aligner needs to improve and converge - that is the goal at the end. Both, aligner and discriminator are trained by an iterative process with a defined ratio. Therefore, for stable training corresponding losses need to be balanced by the right setting of ratio and learning rate for a given architecture.

4.2 Synthetic Data Example

In order to showcase the functionality of the presented approach, a simple use case with synthetic data is presented. The same study is presented in [70], for the sake of completeness we report a short summary of the presented results. For proof of concept a synthetic scenario for a regression model using stationary data is shown. For the same example but with time series input see [70].

Let $X = \mathbb{R}$ be the input space and $Y = \mathbb{R}$ the output space. Let source and target domain be defined by a normal/gaussian distribution over $X_S \subset X, X_T \subset X$ with source mean $\mu_S = 0.6$, target mean $\mu_T = 0.2$ and standard deviation $\sigma_S = \sigma_T = 0.2$. Let

$$D_S = \mathcal{N}(\mu_S, \sigma_S); \tag{4.10}$$

$$D_T = \mathcal{N}(\mu_T, \sigma_T). \tag{4.11}$$

Labels are generated by quadratic mappings

$$q_S : X_S \to Y,\ y = q_S(x) = x^2, \tag{4.12}$$

$$q_T : X_T \to Y,\ y = q_T(x) = (x - 0.4)^2. \tag{4.13}$$

q_S, q_T are the underlying (unknown) modeling functions. Input sample sets S respective T are generated by sampling i.i.d. 1000 times from both domains.

The three parts of DBAM are set up in the following way:

- In order to approximate the mapping q_S we define a statistical model predictor $h_P : X \to Y$ using a ANN architecture. Let h_P be parameterized by θ_p hence represented by $P(x, \theta_p)$ with $x \in X$. P is trained and then fixed using the sample set S and Huber loss. Using P directly on the target domain gives low accuracy due to the shifted mean value when calculating corresponding target labels.
- To enable the usage of P for the target domain with high accuracy DBAM is applied as described by defining aligner $h_A : X \to X$ using a ANN parameterized by θ_A hence $A(x, \theta_A)$ defining the aligner model output for $x \in X$. The aligner is pretrained with

mean squared error in an autoencoder style before it is trained as part of the adversarial training using S and T.

- The discriminator $h_D : X \to \mathbb{R}$ is defined using a ANN parameterized by θ_D hence $D(x, \theta_D)$ defining the model output for $x \in X$. It is trained as second part of the adversarial training using S and T. The discriminator is set for usage of Wasserstein loss (see Section 3.2 and3.2).

All models are set up using ANN architecture with the following details:

- The ANN predictor model consists of 1 dense layers with dimension 4 and linear activation and an output layer with sigmoid activation. The model is trained by minimization of huber loss and default learning rate using Adam optimizer.
- The domain discriminator has 1 dense layer with dimension 4 with leaky ReLU activation function and linear activation for the output layer.
- The aligner consists of 1 dense layers with dimension 4 and relu activation (bottleneck layer) and an output layer with sigmoid activation. For an improved initialization the aligner is pretrained in an autoencoder style hence trained to recreate its input. Adam optimizer with MSE as loss function is used for pretraining using ADAM with learning rate $lr = 1e - 2$.

The whole system is trained according to Algorithm 4.1 with ratio 1:20 and gradient penalty with weight 10. Labels for source and target are used during training.

The results of the alignment corresponding to aligner A are presented in Figure 4.4 showing the alignment of the two domains after 0, 100 and 200 epochs, results corresponding to the predictor P are shown in Figure 4.3 of both source and target before and after alignment.

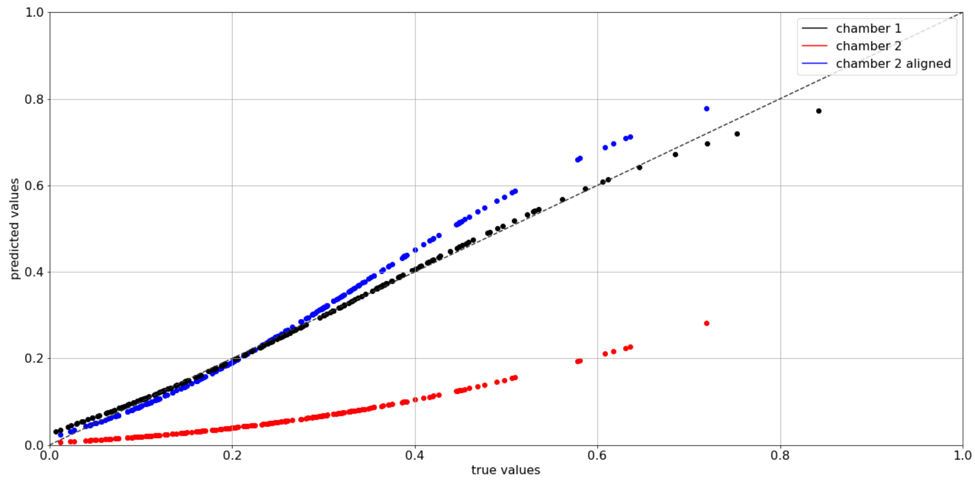


Figure 4.3: DBAM predictor scatter plot for synthetic data use case [70]. Graphical representation of true vs. predicted labels for synthetic test data for X_s (black), X_t (red) and X_t after alignment (blue) for stationary

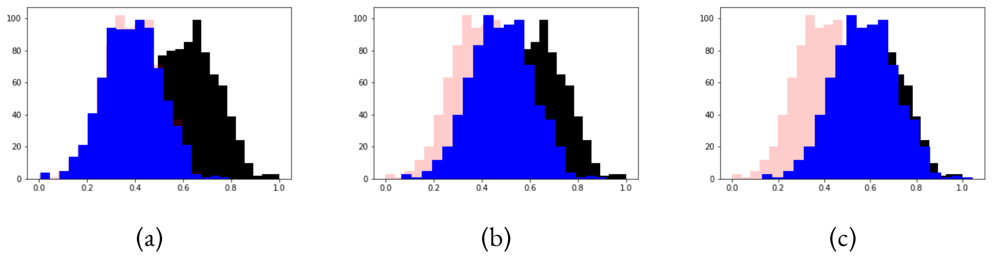


Figure 4.4: DBAM aligner histogram for synthetic data use case [70]. Stepwise shift (blue) during training procedure of the normal distributed training data from X_t (red) to X_s (black) after (a) 0, (b) 100 and (c) 200 iterations. Graph originally published in [70].

5

DBACS: Extended DBAM

In this chapter we present DANN-based Alignment with Cyclic Supervision (DBACS). Since DBACS is a methodological extension of DBAM introduced in the previous chapter, the main focus is to address the additional theoretical parts of DBACS in a general matter and independently of any concrete use case. Therefore, even if the presented extended methodology is again inspired by the needs, highest standards and complexity of semiconductor manufacturing, the following mathematical explanations of all methods and their usage are independently of any application. DBACS is applied in the use cases in Chapter 8 for VM, Chapter 9 for PdM and Chapter 10 for DC.

5.1 DANN-based Alignment with Cyclic Supervision (DBACS)

Inspired on the one hand by work done in the field of GANs (see Chapter 3.2) to improve training, transfer as well as quality besides others and on the other hand by the necessity of scaling up the method and to offer the most general approach as possible suitable for all kind of production application and data types, a extended version of DBAM called DANN-based Alignment with Cyclic Supervision (DBACS) is presented. Using the base architecture of DBAM, a second encoder model named again aligner as well as a second discriminator in the style of so called CycleGAN (see Zhu et al. [239]) is added to enable adaptation in more complex settings, meaning:

- Heterogeneous DA and adaptation of domains that have different dimensions;

- Unpaired sample mapping;
- Aligned features are comparable and interpretable in both directions;
- Trained cycle architecture enables matching on top of adaptation.

Besides the ability to showcase a broad range of semiconductor use cases, this also enables improved alignment (related to prediction as well as alignment quality).

For formalizing the extended model structure of DBACS, the notations used in Chapter 4 are adopted and extended. The source and target input space not necessarily have a common subspace. Hence, the modeling task itself can be domain specific. A common output space is assumed, so no target shift besides class imbalance - hence subsets of the overall output space - is considered explicitly.

Let f_S define a modeling task, let a hypothesis class $\mathcal{H}$ be a set of all possible modeling functions $h \in \mathcal{H}$ that are considered for a specific task. Let $X_S \subset \mathbb{R}^{N_S}$ be defined as the first input space and Y as the output space. The output space Y is defined as $Y = \{0, 1\}$ in case of a binary classification respective $Y = \{0, 1\}^c$ in case of a categorical classification with c classes and $Y \subset \mathbb{R}$ in case of a regression task. A distribution over $X_S \times Y$ is called source domain. In case of not only numerical data but also categorical data, text data or image data, a function (called source embedding) $E_S : I_S \to X_S$ is defined that map the set of instances I_S to the numerical feature spaces X_S. For time series data, let $X_S \subset \mathcal{T} \times \mathbb{R}^{N_S}$ where $\mathcal{T}$ describes the set of considered points in time and $x_S^t \in \mathbb{R}^{N_S}$ a sample from the source feature space taken at a fixed point in time $t \in \mathcal{T}$. A learning algorithm is provided with a source data set S drawn i.i.d. from the source domain D_S with $X_S \times Y_S$, $X_S \subset X$, $Y_S \subset Y$. In the SSL setting, it is distinguished between labeled and unlabeled data and define $S := SL \cup SU$ where SU stands for the unlabeled source sample subset and SL for the labeled one. Without loss of generality for UDA and SSL, $SU = \emptyset$ sine the source domain is assumed to be labeled. Hence

$$S = \{x_S^i, y_S^i\}_{i=1}^{n_S} \sim \{D_S\}^{n_S}, \qquad (5.1)$$

with n_S being the number of drawn samples (all labeled) and therefore $S = SL$ and $X_{SL} = X_S \subset \mathbb{R}^{N_S}$, $Y_{SL} \subset Y_S \subset Y$ in case of numerical data.

A learning algorithm is provided with a second set T drawn i.i.d. from the target domain D_T with different data distribution, representation and feature space. Hence, let $T = TL \cup TU$ be the second called target data set T drawn i.i.d. from a target domain D_T with a distribution over $X_T \times Y_T$, $X_T \subset \mathbb{R}^{N_T}$, $Y_T \subset Y$, and consisting of unlabeled TU and/or labeled TL samples.

$$TL = \{x_T^j, y_T^j\}_{j=1}^{n_T - l} \sim \{D_T\}^{n_T - l}; \tag{5.2}$$

$$TU = \{x_T^j\}_{j=n_T-l+1}^{n_T} \sim \{D_T^X\}^l; \tag{5.3}$$

with n_T being the number of drawn target samples, therefore $X_{TL} \subset X_T \subset \mathbb{R}^{N_T}$, $Y_{TL} \subset Y_T \subset Y$ and $X_{TU} \subset X_T \subset \mathbb{R}^{N_T}$ in case of numerical data. Again, in case of not only numerical data but also categorical data, text data or image data, a function (called target embedding) $E_T : I_T \to X_T$ is defined that map the set of instances I_T to the numerical feature spaces X_T. For time series data, let $X_T \subset \mathcal{T} \times \mathbb{R}^{N_T}$ where $\mathcal{T}$ describes the set of considered points in time and $x_T^t \in \mathbb{R}^{n_T}$ a sample from the target feature space taken at a fixed point in time $t \in \mathcal{T}$.

The DBACS approach (as presented in Figure 5.1 for binary domain adaptation using source and target domain) consists of three extended parts:

- the baseline or reference *prediction* model P that is identical to DBAM in use and role;
- an encoder/alignment model called *aligner* F used to map the target domain to the source domain (the output of the aligner is called aligned), connected to a second encoder/alignment model *aligner* G that maps the source domain to the target domain. By combining both aligner, it is possible to introduce *cycle-consistency* by comparing source samples with its cycled sample and target samples with its cycled samples.
- a domain *discriminator* A for classification of source domain versus aligned target domain. Adversarial training in both directions is enabled by adding a second domain discriminator (called *discriminator* B) for target versus aligned source comparison.

Let $h_P : X_S \to Y$ be a dedicated statistical model already trained on a labeled source sample set S drawn i.i.d. from a domain D_S with $n_S = |S|$ being the number of drawn samples. The neural network representation is parameterized by θ_P and $P(x_S, \theta_P)$ where P is the model function with parameters θ_P that outputs the prediction for $x_S \in X_S$. The

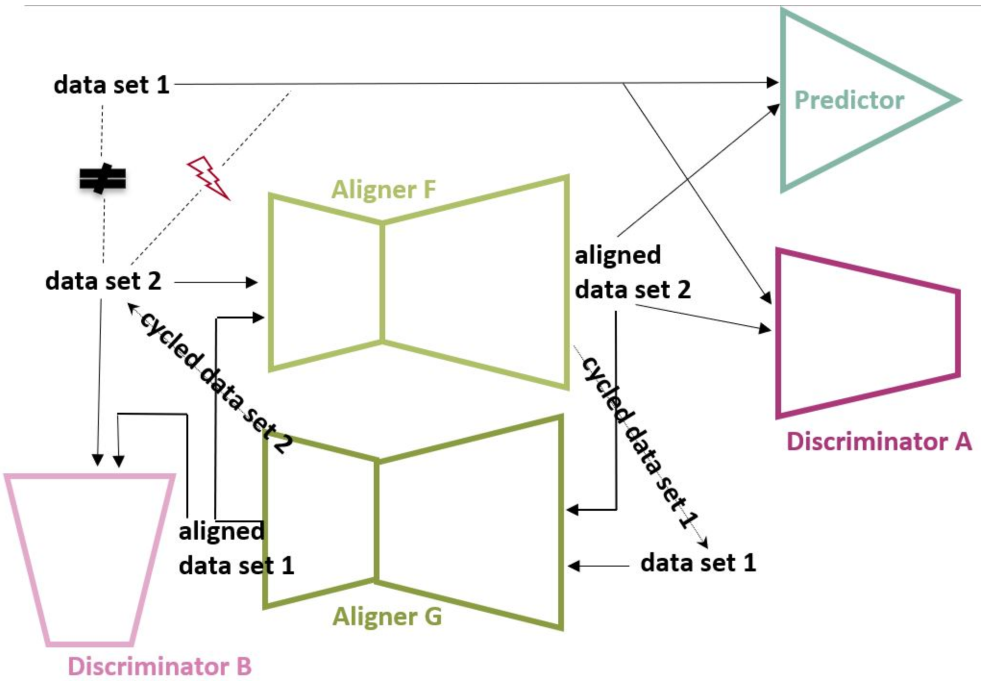


Figure 5.1: Visualization of DBACS including data flow. Graphical representation of DBACS system exploiting input data from two non-identical domains. The arrows represent the data flows. An autoencoder shaped aligner can be used for noise reduction especially for homogeneous DA but is not mandatory.

loss L_P used for training and minimization is taken from Chapter 4, equation (4.2) and (4.3):

$$\begin{aligned}\min_{h_P \in \mathcal{H}} L_P(X_S) = \min_{h_P \in \mathcal{H}} L_{D_S}(h_P(X_S)) &= \min_{\theta_P} L_{D_S}(X_S, \theta_P) \\ &= \min_{\theta_P} \mathbf{L}_{(x_S, y) \in S} \left(P\left(x_S, \theta_P\right), y\right). \qquad (5.4)\end{aligned}$$

The DBAM architecture - see Figure 4.1 and explanations in Chapter 4 - is used as basis and is extended based on the already existing aligner - called aligner F for better distinction - by adding a second aligner G. The two are connected in a cyclic manner to enable forward/backward transformation between source and target feature spaces; hence transformations in both directions and in a bijective manner. This leads to the introduction of an additional loss summand called *cycle-consistency* loss. Here, the optimization goal is to reproduce a bijective mapping so that each $x_S \in X_S$ is mapped to X_T and back to X_S with $F(G(x_S)) \approx x_S$. The same goes for $x_T \in X_T$ with $G(F(x_T)) \approx x_T$.

Let $h_F : X_T \to X_S$ be a statistical model function aligning target to source and $F(x_T, \theta_F)$ be its parameterized representation where F is the model function with parameters θ_F that outputs the prediction for $x_T \in X_T$. Let $h_G : X_S \rightarrow X_T$ be a statistical model function aligning source to target and $G(x_S, \theta_G)$ be its parameterized representation where G is the model function with parameters θ_G that outputs the prediction for $x_S \in X_S$. Then, the **cycle-consistency loss** is defined as:

$$\begin{aligned} L_{cycle_S}(X_S) := L_{G,F,D_S}(X_S) &= \mathbf{L}_{x_S \sim D_S}\left(F(G\left(x_S, \theta_G\right), \theta_F)\right) \\ &= \mathbf{L}_{x_S \sim D_S}(F(G(x_S)), x_S) \end{aligned} \tag{5.5}$$

$$\begin{aligned} L_{cycle_T}(X_T) := L_{F,G,D_T}(X_T) &= \mathbf{L}_{x_T \sim D_T}\left(G(F\left(x_T, \theta_F\right), \theta_G)\right) \\ &= \mathbf{L}_{x_T \sim D_T}(G(F(x_T)), x_T) \end{aligned} \tag{5.6}$$

To give an example we follow the description in [239] where the L_1 norm is used as cycle loss function:

$$L_{cycle_S}(X_S) = \mathbf{L}_{x_S \sim D_S}(F(G(x_S)), x_S) = \mathbf{E}_{x_S \sim D_S}\left[\|F(G(x_S)) - x_S\|_1\right] \tag{5.7}$$

$$L_{cycle_T}(X_T) = \mathbf{L}_{x_T \sim D_T}(G(F(x_T)), x_T) = \mathbf{E}_{x_T \sim D_T}\left[\|G(F(x_T)) - x_T\|_1\right] \tag{5.8}$$

In short, the cycle consistency loss is defined as

$$\mathcal{L}_{cyc}(F, G, X_S, X_T) = L_{cycle_S}(X_S) + L_{cycle_T}(X_T). \tag{5.9}$$

The *adversarial loss* is applied to the output of both discriminator. During the aligner training phase, it is minimized, during discriminator training phase it is maximized (or its negative value minimized). First we define it for the target to source alignment where aligner F tries to map target domain to source domain, then for source to target alignment with aligner G. Let $h_{D_A} : X_S \rightarrow I, h_{D_B} : X_T \rightarrow I$ be two statistical model functions describing distance or classes of source versus aligned target and target versus aligned source. Let h_{D_A} be parameterized by θ_{D_A} and let $D_A(x_S, \theta_{D_A})$ be the parameter representation of discriminator A where D_A is the model function with parameters θ_{D_A} that outputs the prediction for $x_S \in X_S$. Let h_{D_B} be parameterized by θ_{D_B} and let $D_B(x_T, \theta_{D_B})$ be the parameter representation of discriminator B where D_B is the model function with parameters θ_{D_B} that outputs the prediction for $x_T \in X_T$. Then the **adversarial loss** first for source L_{adv_S} and second for target L_{adv_T} is defined based on a selected loss function $\mathbf{L}$:

$$\begin{aligned} L_{adv_S}(X_S, X_T) &:= L_{D_A,D_S}(X_S) - L_{D_A,D_T}(X_T) \\ &= \mathbf{L}_{x_S \sim D_S^X}\left(h_{D_A}\left(x_S\right)\right) - \mathbf{L}_{x_T \sim D_T^X}\left(h_{D_A}\left(h_F(x_T)\right)\right) \\ &= \mathbf{L}_{x_S \sim D_S^X}\left(D_A\left(x_S, \theta_{D_A}\right)\right) - \mathbf{L}_{x_T \sim D_T^X}\left(D_A\left(F(x_T, \theta_F), \theta_{D_A}\right)\right) \end{aligned} \tag{5.10}$$

$$\begin{aligned} L_{adv_T}(X_S, X_T) &:= L_{D_B,D_T}(X_T) - L_{D_B,D_S}(X_S) \\ &= \mathbf{L}_{x_T \sim D_T^X}\left(h_{D_B}\left(x_T\right)\right) - \mathbf{L}_{x_S \sim D_S^X}\left(h_{D_B}\left(h_G(x_S)\right)\right) \\ &= \mathbf{L}_{x_T \in D_T^X}\left(D_B\left(x_T, \theta_{D_B}\right)\right) - \mathbf{L}_{x_S \sim D_S^X}\left(D_B\left(G(x_S, \theta_G), \theta_{D_B}\right)\right) \end{aligned} \tag{5.11}$$

Two scenarios are further discussed where $\mathbf{L}$ is explicitly chosen based on the modeling task at hand:

- First for a classification modeling task: Let $I = \{0, 1\}$ and the discriminator a binary classification model with suitable/sigmoid output activation function. Then the adversarial loss is defined as

$$\mathcal{L}_{adv_S}\left(X_S, X_T\right) = \mathbb{E}_{x_S \sim D_S}\left[\log\left(h_{D_A}(x_S)\right)\right] + \mathbb{E}_{x_T \sim D_T}\left[\log\left(1 - h_{D_A}(F(x_T))\right)\right] \tag{5.12}$$

$$\mathcal{L}_{adv_T}\left(X_S, X_T\right) = \mathbb{E}_{x_T \sim D_T}\left[\log\left(h_{D_B}(x_T)\right)\right] + \mathbb{E}_{x_S \sim D_S}\left[\log\left(1 - h_{D_B}(G(x_S))\right)\right] \tag{5.13}$$

  This is the original loss function used in GAN [75] and in DANN [68] for classification.

- Second for a regression modeling task: Let $I \subset \mathbb{R}$ or $I = \mathbb{R}$ and the discriminator a regression model with linear output activation function. Then the adversarial loss is defined as

$$L_{adv_S}(X_S, X_T) = \mathbb{E}_{x_S \sim D_S}\left[D_A\left(x_S, \theta_{D_A}\right)\right] - \mathbb{E}_{x_T \sim D_T}[(D_A(F(x_T, \theta_F), \theta_{D_A}))], \tag{5.14}$$

$$L_{adv_T}(X_S, X_T) = \mathbb{E}_{x_T \sim D_T}\left[D_B\left(x_T, \theta_{D_B}\right)\right] - \mathbb{E}_{x_S \sim D_S}[(D_B(G(x_S, \theta_G), \theta_{D_B}))], \tag{5.15}$$

  where $\mathbb{E}$ defines the expected value and the loss is an approximation of the Wasserstein distance of two sampled distributions, for details see [78].

In cases where X_S and X_T share the same feature representation and dimensions, it can happen that very similar samples - meaning samples where important properties should not be changed during the adaptation process, [239] is mentioning color as an example in the field of computer vision - show up in the other domain. Hence it is recommended by [239] based on Taigman et al. [198] to add one more additional loss term namely the *identity loss*. The idea is that if almost identical samples occur, the aligner should perform close to an identity function. Hence the **identity loss** is defined as,

$$L_{id}(X_S) := L_{F,D_S}(X_S) = \mathbf{L}_{x_S \sim D_S^X}\left(F\left(x_S, \theta_F\right)\right) = \mathbf{L}_{x_S \sim D_S^X}\left(F(x_S), x_S\right)$$

$$L_{id}(X_T) := L_{G,D_T}(X_T) = \mathbf{L}_{x_T \sim D_T^X}\left(G\left(x_T, \theta_G\right)\right) = \mathbf{L}_{x_T \sim D_T^X}\left(G(x_T), x_T\right)$$

with $\mathbf{L}$ defining a selected loss function e.g L_1 loss. The so called identity loss is added to the overall loss function. In the scenario of heterogeneous domain adaptation hence in cases where source and target space are not identical due to different representations the identity loss is omitted.

The training itself happens in an adversarial setting with a two-player game approach. The adversarial training routine for DBACS is based on Algorithm 4.1 and adopted to parallel training of both aligner and both discriminator plus inclusion of the additional loss terms:

- The first competitor of the adversarial training is the discriminator D_A trained to distinguish between source and aligned target data meaning optimizing a classification loss. In parallel the discriminator D_B is trained to distinguish between target and aligned source data also meaning optimizing a classification loss. Then, the optimization of the discriminator A and discriminator B loss $L_{D_{total}}$ is defined as

$$\begin{aligned}\max_{\theta_{D_A},\theta_{D_B}} L_{D_{total}}(X_S, X_T) &= \max_{\theta_{D_A}} L_{adv_S}(X_S, X_T) + \max_{\theta_{D_B}} L_{adv_T}(X_S, X_T)\\ &= \max_{\theta_{D_A}} L_{D_A,D_S}(X_S, \theta_{D_A}) - L_{D_A,D_T}(F(X_T, \theta_F), \theta_{D_A})\\ &+ \max_{\theta_{D_B}} L_{D_B,D_T}(X_T, \theta_{D_B}) - L_{D_B,D_S}(G(X_S, \theta_G), \theta_{D_B})\\ &= \max_{\theta_{D_A}} \mathbf{L}_{x_S \in S^X}\left(D_A\left(x_S, \theta_{D_A}\right)\right) - \mathbf{L}_{x_T \in T^X}\left(D_A\left(F(x_T, \theta_F), \theta_{D_A}\right)\right)\\ &+ \max_{\theta_{D_B}} \mathbf{L}_{x_T \in T^X}\left(D_B\left(x_T, \theta_{D_B}\right)\right) - \mathbf{L}_{x_S \in S^X}\left(D_B\left(G(x_S, \theta_G), \theta_{D_B}\right)\right) \quad (5.16)\end{aligned}$$

  where S^X, T^X define the part of the drawn sample set S, T that corresponds to the input space X_S, X_T. Depending on additional loss terms chosen to improve discriminator stability and quality, more loss term can be added accordingly.

- The second competitor in the adversarial training is the aligner cycle. We define $L_{A_{total}}$

using the adversarial loss for both aligners, the cycle consistency loss and if possible the identity loss. In case of labeled target data the aligner F is also updated in order to optimize prediction loss L_P for aligned target data. The adversarial part of the aligner losses is set in opposite direction compared to the two discriminator. If training both discriminator involves a loss minimization with respect to the discriminator parameters, training both aligner means a loss maximization with respect to the update of both aligner parameters and vise versa. Based on equation (5.16):

$$\begin{aligned}
&\min_{\theta_F,\theta_G} L_{A_{total}}(X_S, X_T)\\
&= \min_{\theta_F} \left[\lambda_{adv_S} \cdot L_{adv_S}(X_S, X_T) + \lambda_{id_S} \cdot L_{id}(X_S) + \lambda_P \cdot L_P(F(X_T))\right]\\
&+ \min_{\theta_G} \left[\lambda_{adv_T} \cdot L_{adv_T}(X_S, X_T) + \lambda_{id_T} \cdot L_{id}(X_T)\right]\\
&= \min_{\theta_F} [-\lambda_{adv_S} \cdot \mathbf{L}_{x_T \in T^X} \left(D_A\left(F(x_T, \theta_F), \theta_{D_A}\right)\right)\\
&\qquad + \lambda_{id_S} \cdot \mathbf{L}_{x_S \in S^X} \left(F(x_S, \theta_F), x_S\right) + \lambda_P \cdot \mathbf{L}_{(x_T,y) \in TL}(P\left(F(x_T, \theta_F), \theta_P\right))]\\
&+ \min_{\theta_G} \; [-\lambda_{adv_T} \cdot \mathbf{L}_{x_S \in S^X} \left(D_B\left(G(x_S, \theta_G), \theta_{D_B}\right)\right)\\
&\qquad + \lambda_{id_T} \cdot \mathbf{L}_{x_T \in T^X} \left(G(x_T, \theta_G), x_T\right)]
\end{aligned} \tag{5.17}$$

where $\lambda_{(\cdot)}$ represents the weight assigned to each corresponding loss term. As it can be seen later in the use cases this is necessary especially if the overall scale of different error functions for different model parts differ; for example mean absolute error for predictor versus categorical cross entropy for discriminator: both are used in parallel to update the aligner weights, hence based on their range need to be weighted accordingly. For $\lambda_P = 0$ the training happens in an unsupervised setting where no target labels are available. A gradient penalty regularization term (for details see Chapter 3.2) is added when updating both aligners. The recommendations of [78] are followed during training of the two aligner.

For training of DBACS, the algorithm presented in Algorithm 4.1 is used as base and enriched in a way that both discriminator as well as both aligner get updated simultaneously. Aligner F (the one also existing in DBAM) has the predictor loss as well as a gradient penalty regularization included hence has a supervised training (if target labels are available) while aligner G is trained in an unsupervised manner only using discriminator loss, cycle loss (plus identity loss if applicable) and gradient penalty regularization. Use case specific losses can be added to improve prediction quality. Feature matching loss (FM) inspired by [72] and SSIM (see 7) are added and discussed in Chapter 10: Defect Classification (DC). The training itself happens in an adversarial setting with a two-player game approach: the pseudo-code of the approach is reported in Algorithm 4.1. The following step-by-step description describes the extended training procedure used for DBACS:

1. First a dedicated prediction model is trained on samples from the source domain. During training the parameters respective weights θ_P are fitted via minimization of the L_P loss. The source domain is selected under consideration of quality, stability, availability of data respective labels and best performance for the corresponding modeling task. For all the following training steps all predictor weights θ_P are kept frozen in order to preserve the high quality of the dedicated source prediction model.

2. The aligner F is mapping the target domain to the source domain and is set up in a way that after training it allows the direct comparison of the source domain with the aligned target domain and enables interpretability and further analysis of domain differences. The aligner G is mapping the source domain to the target domain and is set up in a way that after training it allows the direct comparison of the target domain with the aligned source domain and enables interpretability and further analysis of domain differences. The set up as well as its initialization and the possibility of a pre-training step depends highly on the underlying input structure of source and target. The parameters respective weights θ_F and θ_G are fitted via optimization of the $L_{A_{total}}$ loss.

3. The discriminator models are trained on samples from source as well as aligned target data respective target as well as aligned source data and $L_{D_{total}}$ is optimized in opposite direction compared to the aligner loss. The parameters respective weights θ_{D_A} and θ_{D_B} are fitted via optimization of the $L_{D_{total}}$ loss. In the adversarial setting θ_F, θ_G are fitted via minimization of the $L_{A_{total}}$ loss while for θ_{D_A}, θ_{D_B} the $L_{D_{total}}$ loss is maximized. The iterative training is set up with a predefined training ratio between aligner models and discriminator models update. Whenever one of the adversarial competitors is trained, the weights of the other competitor are short-term frozen.

Details, specifications and tuning of hyperparameter are data respective use case specific and therefore discussed in the dedicated application chapters.

5.2 Synthetic Data Example

To showcase the functionality of DBACS and to present a proof of concept, the synthetic data example presented at the end of Section 4 is revisit and adopted according to the extended scenario: It is again based on gaussian distributed features with shifted mean and a quadratic mapping as underlying target function.

The predictor $P : X_S \to Y$, parameterized by θ_p is trained on a sample set of source domain and then fixed by freezing all weights. In addition to a first aligner called aligner F with $F : X_T \to X_S$ and parametrization $F(x_T, \theta_F)$ that defines the predicted outputs for $x_T \in X_T$. - a second aligner $G : X_S \to X_T$ with $G(x_S, \theta_G)$ that outputs the prediction for $x_S \in X_S$ is defined. In addition to the previous existing discriminator - renamed to $D_A : X_S \to \mathbb{R}$ with $D_A(x_S, \theta_{D_A})$ that defines outputs for $x_S \in X_S$. - a second discriminator with identical architecture $D_B : X_T \to \mathbb{R}$ with $D_B(x_T, \theta_{D_B})$ that predicts outputs for $x_T \in X_T$ is introduced. All models are set up using ANN architecture with the following details:

- The ANN predictor model consists of 1 dense layers with dimension 4 and linear activation and an output layer with sigmoid activation. The model is trained by minimization of huber loss and default learning rate using Adam optimizer;
- the domain discriminators both have the same architecture 1 dense layer with dimension 4 with leaky ReLU activation function and linear activation for the output layer since Wasserstein loss is used;
- both aligner consist of 1 dense layers with dimension 4 and relu activation) and an output layer with sigmoid activation. For an improved initialization both aligners are pretrained in an autoencoder style hence trained to recreate their input. MSE as loss function is used for pretraining with ADAM optimizer and learning rate $lr = 1e-2$.

Algorithm 4.1 is modified so that both aligner are trained in parallel, aligner F using supervised training with adversarial loss in form of Wasserstein, prediction loss in form of MSE, cycle consistency loss using L_1 loss, and gradient penalty; aligner G using unsupervised training with adversarial loss, cycle consistency loss and gradient penalty. Discriminator D_A and D_B are trained using the adversarial loss in form of Wasserstein. The ratio 1:5 (aligner versus discriminator) is used.

The results of the supervised alignment corresponding to aligner F are presented in Figure 5.3 showing the alignment of target to source after 0, 10 and 20 epochs, the results of the alignment corresponding to aligner G are presented in Figure 5.4 showing the unsupervised alignment of source to target also after 0, 10 and 20 epochs. Results corresponding to the predictor P are shown in Figure 5.2 of both source and target before and after alignment.

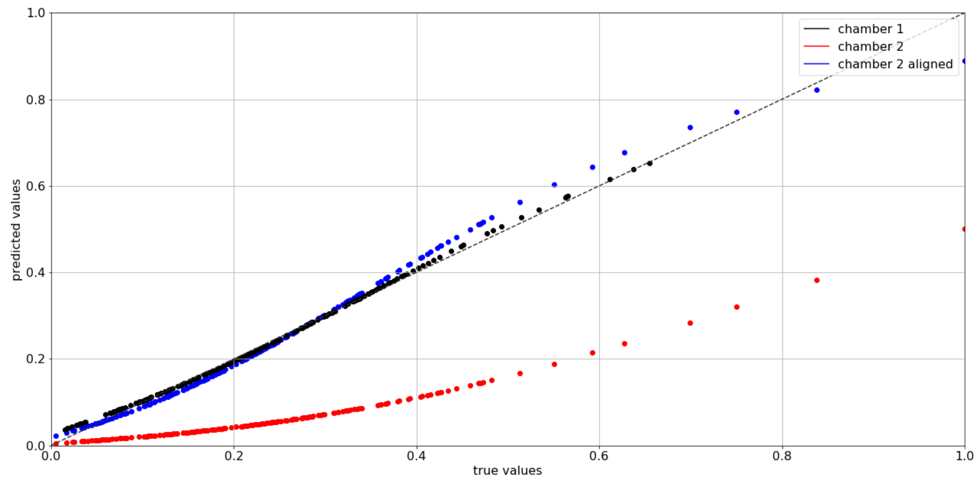


Figure 5.2: DBACS predictor scatter plot for synthetic data use case. Graphical representation of true vs. predicted Y for synthetic test data for X_s (black), X_t (red) and X_t after alignment (blue) using DBACS

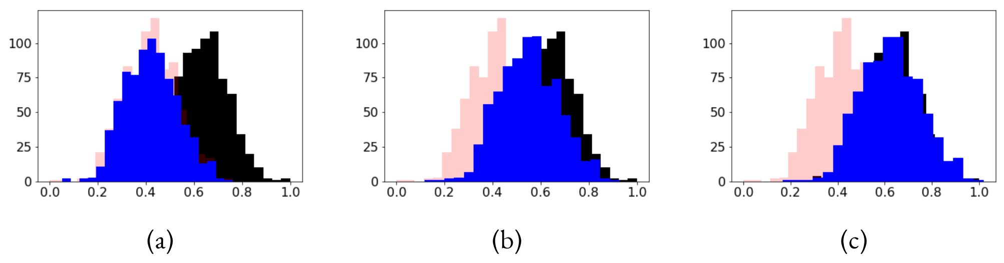


Figure 5.3: DBACS aligner F histogram for synthetic data use case. Stepwise shift (blue) during training procedure of the normal distributed training data from X_t (red) to X_s (black) after (a) 0, (b) 10 and (c) 20 iterations.

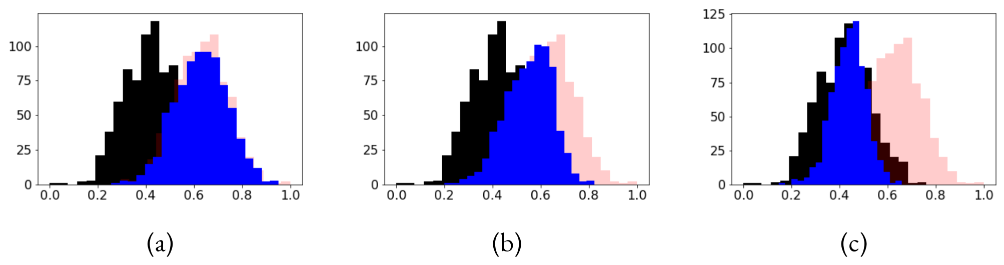


Figure 5.4: DBACS aligner G histogram for synthetic data use case. Stepwise shift (blue) during training procedure of the normal distributed training data from X_st (red) to X_t (black) after (a) 0, (b) 10 and (c) 20 iterations.

6 Benchmark Methods and Models

In the following chapter multiple benchmark models and statistical methods are described that are later used as challenger for DBAM and DBACS in Chapter 8, Chapter 9, Chapter 10. The methods are selected based on application specific literature, hence we apply use case specific benchmarking. For more details on relevant literature we refer to the corresponding use case chapters. Here, the methods are defined in a general way independent of the application they are later used for.

6.1 General Linear Regression Models

A regression model describes the relationship of one or more variables (called independent) with an output variable (called dependent). There exists different kind of regression techniques based on factors like number of variables and shape of the expected function. An introduction is given by following the explanations in [69].

The following notations are used throughout the whole section. Let $X \subset \mathbb{R}^N$ be the N-dimensional input space and $x \in \mathbb{R}^N$ one sample containing values for each feature and Y the target or output space with $y \in [0,1] \subset \mathbb{R}$ a scalar value.

Linear Regression and Ordinary Least Squared (OLS): For linear regression (LR) the modeling function is assumed to be linear. Hence, the Ordinary Least Squares (OLS) estimates $\{\beta_j\}_{j=1}^N$ are obtained by:

$$\hat{\beta} = \arg\min_{\beta} \sum_i \left(y - \sum_{j=1}^{N} x_{ij}\beta_j \right)^2 . \tag{6.1}$$

where $i \in \{1, ..., n\}$ with $n = |S|$ and S defining a sample set. Since the OLS objective function can yield to overfitting and computational issues for high-dimensional and collinear data, additional regularization terms are introduced in the considered approaches to assure better generalization performance and, in some cases, sparsity. Hence, Ridge Regression **RR**, Lasso Regression **LASSO** and Fused Lasso Regression **FL** are all based on the classical ordinary least squared paradigm. For their usage in VM we refer to [Park and Kim] and [191].

$$\hat{\beta}_{\text{RR}} = \arg\min_{\beta} \sum_i \left(y - \sum_{j=1}^{N} x_{ij}\beta_j \right)^2 + \lambda \cdot \sum_{j=1}^{N} {\beta_j}^2 \tag{6.2}$$

$$\hat{\beta}_{\text{LASSO}} = \arg\min_{\beta} \sum_i \left(y - \sum_{j=1}^{N} x_{ij}\beta_j \right)^2 + \lambda \cdot \sum_{j=1}^{N} |\beta_j| \tag{6.3}$$

$$\hat{\beta}_{\text{FL}} = \arg\min_{\beta} \sum_i \left(y - \sum_{j=1}^{N} x_{ij}\beta_j \right)^2 + \lambda_1 \cdot \sum_{j=1}^{N} |\beta_j| + \lambda_2 \cdot \sum_{j=1}^{N} |\beta_j - \beta_{j-1}| \tag{6.4}$$

where λ, λ_1, λ_2 are non negative penalty parameters (called regularization parameters): the L_2-penalty $\sum_j {\beta_j}^2$ typically helps in case of collinear data, the L_1-penalty $\sum_j |\beta_j|$ induces sparsity while the combined term

$$\lambda_1 \sum_{j=1}^{N} |\beta_j| + \lambda_2 \sum_{j=1}^{N} |\beta_j - \beta_{j-1}| \tag{6.5}$$

that favors neighboring features or timestamps and therefore allow to remove noise effects in the models and can be used for time-series data as in [Park and Kim]. More details are given by Friedman et al. [66]. OLS is used in Chapter 8: Virtual Metrology (VM) as benchmark prediction model.

6.2 Ensemble Learning and Decision Trees

Ensemble Learning uses a group of models to improve prediction performance. It consists of a finite group of models but is very flexible towards individual model structure. Decision trees are based on decisions and their consequences in a tree-like or flow-chart structure. Conditional probabilities are used to make predictions.

Random Forest (RF): Random Forest (RF) is a tree-based ensemble method that is well-known for its robustness, flexibility and high accuracy even when facing noise or nonlinearity. A random forest is based on a set of decision trees build up by a so-called recursive partitioning algorithm applied on different random sample sets of the training data. The prediction for an unseen input is then computed by averaging over the different predictions from the individual trees.

For a more detailed description of the method see Breiman [23]. For a visualization of RF see Figure 6.1. RF Regression (RFR) effectiveness in semiconductor process control with VM technologies was shown for example in [31]. RFR is used in Chapter 8: Virtual Metrology (VM) as benchmark prediction model.

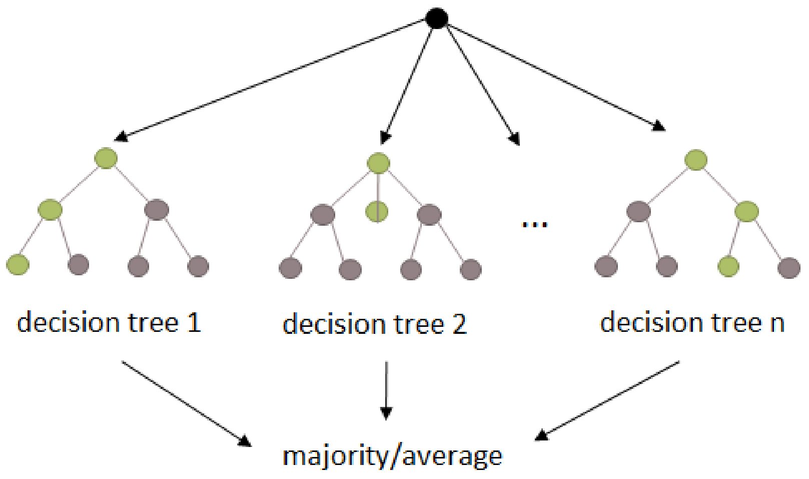


Figure 6.1: Graphical representation of a random forest model. For regression the average over all decision tree results are taken. For classification a majority vote leads to the final output

Gradient Tree Boosting (GTB) Gradient Tree Boosting (GTB) respective its XG-Boost implementation by Chen and Guestrin [34] is used to improve RF results. Boosting by Friedman [67] describes an ensemble method consisting of multiple (weaker) models that are combined into a single one. For GTB respective XGBoost, decision or regression trees are used as weaker models. The training happens in a sequential way such that each predictor or classifier outperforms its predecessor. The following summarizes the step-by-step training procedure in [34]:

Let $S = \{(x_i, y_i)\}_{i=1}^n$ with $x_i \in \mathbb{R}^N, y_i \in \mathbb{R}$ a labeled sample set with N features and sample size $|S| = n$, let K be the number of additive functions and $F = \{f(x) = \omega_{q(x)}\}, (q : \mathbb{R}^N \to T, \omega \in \mathbb{R}^T)$ be the space of prediction/classification functions, in our case regression trees, T stands for number of leaves in the tree and ω describes the scores on the leaves. Then the output is defined via model ensembling meaning the sum of K additive functions:

$$\hat{y}_i := \Phi(x_i) = \sum_{k=1}^{K} f_k(x_i), \qquad f_k \in F \tag{6.6}$$

To learn the functions $\{f_k\}_{k=1}^K$ the following objective is minimized:

$$L(\phi) = \sum_i \mathbf{L}(\hat{y}_i, y_i) + \sum_k \Omega(f_k), \tag{6.7}$$

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$$

where $\mathbf{L}$ is a selected well behaved (differentiable and convex) loss function, Ω penalizes the complexity of the model. Since (6.7) includes functions as parameter that cannot be solved by traditional optimization methods, we transform it to enable additive training. Let $\hat{y}_i^{(t)}$ be the i-th prediction, then the loss function at the t-th iteration can be defined as:

$$L^{(t)} = \sum_{i=1}^{n} \mathbf{L}(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t) \tag{6.8}$$

where the added component f_t is the tree that most improves the model according to (6.7). Second order Taylor approximation is applied:

$$L^{(t)} \simeq \sum_{i=1}^{n} \left[\mathbf{L}(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t) \tag{6.9}$$

where $g_i = \partial_{\hat{y}^{(t-1)}} \mathbf{L}(y_i, \hat{y_i}^{(t-1)})$ and $h_i = \partial^2_{\hat{y}^{(t-1)}} \mathbf{L}(y_i, \hat{y_i}^{(t-1)})$. Removing the constant terms and using the definition of Ω leads to:

$$\begin{aligned} \tilde{L}^(t) &= \sum_{i=1}^{n} \left[g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \gamma T + \frac{1}{2} \lambda \sum_{j=1}^{T} w_j^2 \\ &= \sum_{j=1}^{T} \left[\left(\sum_{i \in I_j} g_i \right) w_j + \frac{1}{2} \left(\sum_{i \in I_j} h_i + \lambda \right) w_j^2 \right] + \gamma T \end{aligned} \tag{6.10}$$

where $I = \{i | q(x_i) = j\}$ is the set of indeces of points associated to a leaf j.

$$w_j^* = \frac{\sum_{i \in I_j} g_i}{\sum_{i \in I_j} hi + \lambda} \qquad \tilde{L}^{(t)}(q) = -\frac{1}{2} \sum_{j=1}^{T} \frac{\left(\sum_{i \in I_j} g_i \right)^2}{\sum_{i \in I_j} hi + \lambda} + \gamma T \tag{6.11}$$

is used to compute the optimal weight w_j^* of leaf j for a fixed structure $q(x)$. $\tilde{L}^{(t)}$ can be used to estimate the goodness of a tree structure $q(x)$: the smaller the score, the better the structure. To speed up performance a greedy algorithm is used to to rank all possible tree structures. Loss reduction after splitting a leave into two branches is measured as:

$$L_{split} = \frac{1}{2} \left[\frac{\left(\sum_{i \in I_L} g_i \right)^2}{\sum_{i \in I_L} h_i + \lambda} + \frac{\left(\sum_{i \in I_R} g_i \right)^2}{\sum_{i \in I_R} hi + \lambda} + \frac{\left(\sum_{i \in I} g_i \right)^2}{\sum_{i \in I} h_i + \lambda} \right] - \gamma \tag{6.12}$$

This function is used to evaluate splits in practice.

To further improve performance and speed together with avoiding overfitting, XGBoost makes use of shrinkage and column subsampling. A short coming of XGBoost can be its high number of parameters leading to longer training times and tendency to overfit. XGBoost is used in Chapter 9: Predictive Maintenance (PdM) as benchmark prediction model.

6.3 Linear and Kernel Transformations

Linear and kernel transformations are based on mappings into a new feature space. The kernel trick - for an introduction to kernel for ML see Hofmann et al. [93], see Figure 6.2 for a visualization - describes an efficient way to transform nonlinear data by mapping it into a higher dimensional space where it is linearly separable. It is a key factor for successful and easy numerical usage.

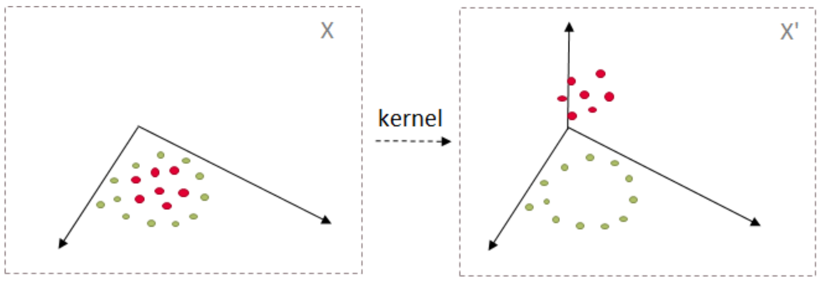


Figure 6.2: Graphical representation of kernel trick meaning a mapping into a higher dimensional space to make data linearly separable. X describes the input space,$\Phi : X \rightarrow X'$ describes the mapping with X' the new (higher dimensional) feature space . The example shows the kernel $\Phi : \mathbb{R}^2 \rightarrow \mathbb{R}^3, \Phi(x, y) = (x^2, y^2, \sqrt{2}xy)$.

The following notations are used throughout the whole section. Let X be considered as the input space and Y as the output space. It is recaptured that a distribution over $X \times Y$ is called a domain. A learning algorithm is provided with a data set S drawn i.i.d. from a domain D_S with $X_S \times Y_S$, $X_S \subset X$, $Y_S \subset Y$. For better differentiation D_S is called source domain.

Correlation and Covariance: Covariance evaluates the relationship of two variables/features, more precisely the level two variables/features vary with each other. Let C be the covariance of two random variables from X with sample index i, j (I stands for number of samples, j stands for number of features). Then the Pearson product-moment correlation coefficients R_{ij} are computed with

$$R_{ij} = \frac{C_{ij}}{\sqrt{C_{ii} * C_{jj}}} \tag{6.13}$$

Principal component analysis (PCA): Principal component analysis (PCA) is a linear transformation of a vector space respective its points/vectors. The projection is created in a way that highest occurring variance is represented by the first latent dimension - the so called first principle component - second highest variance by the second principle component and so on. For a detailed description including mathematical formula see Jolliffe [105]. There, the following steps are followed:

1. standardization of variables in order to have a standardized range hence equal contribution of each feature. Especially important if individual feature scales vary a lot;

2. computation of covariance matrix in order to understand pairwise correlation and redundant information;

3. compute the eigenvalues and eigenvectors of covariance matrix. The eigenvectors are sorted based on their corresponding eigenvalue that stands for the amount of variance they represent. Then, the first principle components are defined by the eigenvectors with the largest eigenvalue that represents the directions of the axes with the highest variance (most information). By ranking the eigenvectors in order of their eigenvalues, highest to lowest, you get the principal components in order of significance.

Let $\Psi : X \to X'$ define the nonlinear principle component transformation to be computed with X' defining the space the features are projected on. Then principle component transformation Ψ is formalized via

$$S' = \Psi(S) = \Gamma^T S \tag{6.14}$$

where $S' \in X'$ describes the transformed/projected input sample set with S being the sample set drawn i.i.d. from the domain defined over $X \times Y$. Γ consists of the eigenvectors and is computed via $\Lambda = \Gamma^T \Sigma \Gamma$, Λ is a diagonal matrix defined by the eigenvalues and Σ is the covariance matrix.

PCA can be used for dimensionality reduction of large input space. It is limited by its linearity and the fact that smaller variance can still be an important factor for significant misbehavior. Here it is used as benchmark transformation in Chapter 8: Virtual Metrology (VM) for the heterogeneous domain adaptation part.

(Kernel) Canonical Correlation Analysis (CCA/KCCA): Canonical Correlation Analysis (CCA) defines linear or in the more general case a kernel (KCCA) transformation for two sets of variables/features such that after the transformation the projected features are maximal correlated in the linear case and that the inner scalar product is maximized in the kernel case. For a general visualization of CCA see Figure 6.3. A summary of the descriptions is taken from Hardoon et al. [81]:

Let $S = \{x_S\} = \{x_{S_i}\}_{S_i=1}^{r} \sim \{D_S\}^r$ with $x_{S_i} \in X_S$ the source sample set of size $|S| = r$ drawn i.i.d. from D_S called source domain. For CCA a second data set is considered. Let $T = \{x_T\} = \{x_{T_j}\}_{T_j=1}^{l} \sim \{D_T\}^l$ with $x_{T_j} \in X_T$ and $X_T \subset X$ be the second set of samples of size $|T| = l$ drawn i.i.d. from D_T called target domain. Let $\Phi_S : X_S \to X_S^{'}$ and $\Phi_T : X_T \to X_T^{'}$ define two linear transformations. Let those transformation be defined by

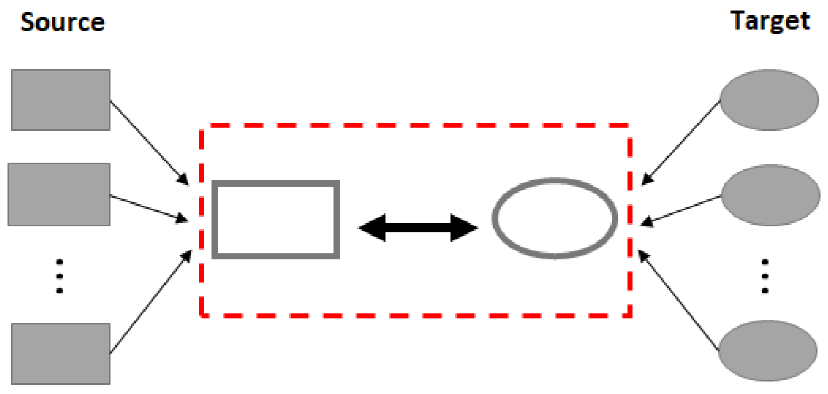


Figure 6.3: Visualization of Canonical Correlation Analysis [81]. Visualization of (Kernel) Canonical Correlation Analysis (CCA/KCCA). The canonical components of source and target are a weighted combination of corresponding input features. The correlation respective inner product of the canonical components within the red box is maximized. Similar to PCA the number of canonical components can be defined.

two projections into direction w_S, w_T via:

$$\Phi_S(S) = S^{'} = S_{x_S,w_S} = \langle w_S, x_S \rangle \tag{6.15}$$
$$\Phi_T(T) = T^{'} = T_{x_T,w_T} = \langle w_T, x_T \rangle. \tag{6.16}$$

The objective is to find w_S, w_T such that the correlation between the projected vectors is maximised, hence:

$$\rho = \max_{w_S,w_T} corr\left(S_{x_S,w_S}, T_{x_T,w_T}\right) = \max_{w_S,w_T} \frac{\langle S_{x_S,w_S}, T_{x_T,w_T} \rangle}{\|S_{x_S,w_S}\|\|T_{x_T,w_T}\|}. \tag{6.17}$$

This is reformulated by using properties of the inner product:

$$\rho = \max_{w_S,w_T} \frac{w_s' \mathbb{E}[x_S x_T'] w_T}{\sqrt{w_S' \mathbb{E}[x_S x_S'] w_S w_T' \mathbb{E}[x_T x_T'] w_T}} \tag{6.18}$$

with $\mathbb{E}$ denoting the discrete empirical expectation, $'$ denotes the transpose of a vector or a matrix. Using the covariance matrix with

$$C = C(x_S, x_T) = \mathbb{E}[x_S x_T] = \begin{bmatrix} C_{x_S x_S} & C_{x_T x_S} \\ C_{x_S x_T} & C_{x_T x_T} \end{bmatrix}$$

where C is a block matrix with the within-covariance $C_{x_S x_S}, C_{x_T x_T}$ and the between-covariance matrices $C_{x_S x_T}, C_{x_T x_S}$ as entries.

Finally the optimization objective can be formulated in the following way:

$$\rho = \max_{w_S, w_T} \frac{w_s' C_{x_S x_T} w_T}{\sqrt{w_S' C_{x_S x_S} w_S w_T' C_{x_T x_T} w_T}} \tag{6.19}$$

By checking that rescaling of w_S, w_T does not change the problem, (6.19) can be maximized subject to

$$w_S' C_{x_S x_S} w_S = 1, \tag{6.20}$$

$$w_T' C_{x_T x_T} w_T = 1. \tag{6.21}$$

Next, the formulation of the dual problem is used. Computing the corresponding Lagrangian L leads to

$$L(\lambda, w_S, w_T) = w_s' C_{x_S x_T} w_T - \frac{\lambda_S}{2}(w_S' C_{x_S x_S} w_S - 1) - \frac{\lambda_T}{2}(w_T' C_{x_T x_T} w_T - 1) \tag{6.22}$$

Then, the partial derivatives in direction of w_S, w_T are:

$$\frac{\partial L}{w_S} = C_{x_S x_T} w_T - \lambda_S C_{x_S x_S} w_S = 0, \tag{6.23}$$

$$\frac{\partial L}{w_T} = C_{x_T x_S} w_S - \lambda_T C_{x_T x_T} w_T = 0. \tag{6.24}$$

Subtracting w_S∗(6.24) from w_T∗(6.23), set $\lambda = \lambda_S = \lambda_T$, assuming $C_{x_T x_T}$ is invertible, rearrange the equation and usage of the partial derivative leaves to

$$C_{x_S x_T} C_{x_T x_T}^{-1} C_{x_T x_S} w_S = \lambda^2 C_{x_S x_S} w_S \tag{6.25}$$

that is equivalent to a generalised eigenproblem of the form $Ax = \lambda Bx$. By using Cholesky decomposition, this can be further simplified to a symmetric eigenvalue problem: $Ax = \lambda x$.

To avoid the restrictions from the linearity of CCA, KCCA first maps the data to a higher dimensional space before following the steps described above.

$$\tilde{\Phi}_S : S \to \tilde{\Phi}_S(S), \tilde{\Phi}_S(x_S) = (\tilde{\Phi}_{S_1}(x_S), \ldots, \tilde{\Phi}_{S_R}(x_S)), R > dim(X_S)$$
$$\tilde{\Phi}_T : T \to \tilde{\Phi}_T(T), \tilde{\Phi}_T(x_T) = (\tilde{\Phi}_{T_1}(x_T), \ldots, \tilde{\Phi}_{T_L}(x_T)), L > dim(X_T).$$

The kernel trick [93], meaning exchanging one kernel representation with another respective mapping data into a higher dimensional space where it can be linearly separated, is used. A visualization of the kernel trick is given in Figure 6.2. If then S and T are the data matrices, $C_{x_S x_S} = S'S$, $C_{x_S x_T} = S'T$, $w_S = S'\alpha$ and $w_T = T'\beta$ (with α, β given the direction of the mapping), it is possible to show that by using again the dual form and applying the same transformations, the problem can be again characterized as generalized eigenvalue problem $Ax = \lambda x$.

For a detailed discussion on computational aspects, trivial solutions and numerical complexity see [81]. CCA is used as domain adaptation benchmark in Chapter 8: Virtual Metrology (VM) in the heterogeneous DA part.

6.4 Deep Transformation and Domain Adaptation

The section is split up into pseudo-labeling based methods using neural networks followed by feature-based and instance-based domain adaptation methods that both can be applied on a broad range of statistical modeling methods. All methods are suitable for un- and/or semi-supervised learning assuming labeled source but partly labeled or unlabeled target data. For an overview of different learning paradigm see Chapter 3.1. Notation are taken from chapter 3.3, chapter 4, chapter 5. For an introduction into domain adaptation theory see Chapter 3.3. Figure 6.4 gives an overview of methods and the use cases they are later applied to.

6.4.1 Pseudo-Labeling

Pseudo-labeling by Lee et al. [123] itself is a semi-supervised learning method and consists in producing artificial labels for not (yet) manually classified images and in training a model to predict artificial labels when fed with unseen and unlabeled images as input. It uses the model class prediction as a label to train again, if it is above a certain probability threshold. Hence a classification model is trained on labeled data by minimizing a supervised loss func-

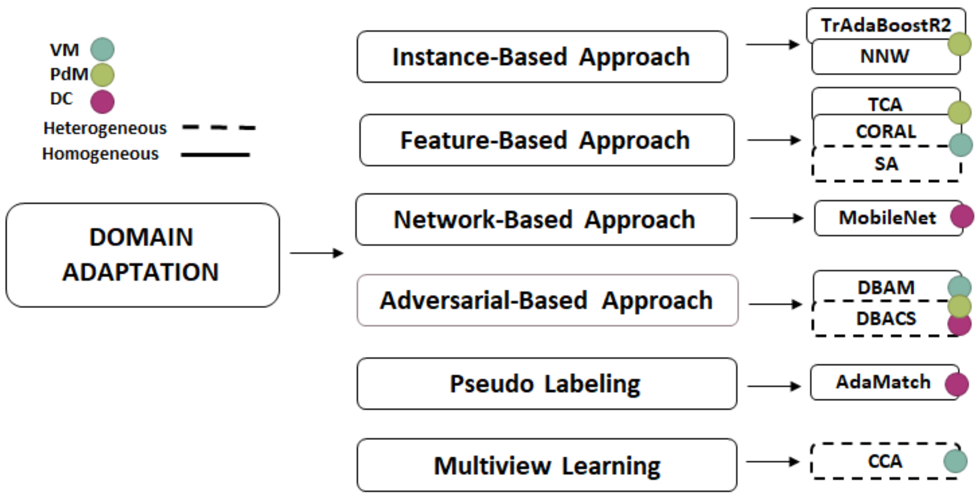


Figure 6.4: Visualization of DL-based TF categories, methods and corresponding use cases. Visualization and illustration to identify the relationship of applications to the corresponding methodologies and methods presented in this chapter. The colored use cases are presented in Part III: Applications and Case Studies.

tion usable for classification. Then unlabeled data is used to enable more general usage hence further improve accuracy for source but especially for unlabeled target data.

The following notations are used throughout the whole section. Let X be considered as the input space and Y as the output space. It is recaptured that a distribution over $X \times Y$ is called a *domain*. Let $\mathbf{L} : Y \times Y \rightarrow \mathbb{R}$ describe a loss function. A learning algorithm is provided with a data set S drawn i.i.d. from a domain D_S with $X_S \times Y_S, X_S \subset X, Y_S \subset Y$. For better differentiation D_S is called *source domain*. In the SSL setting, it is distinguished between labeled and unlabeled data and defined $S := SL \cup SU$ where SU stands for the unlabeled sample subset and SL for the labeled one. Without loss of generality for UDA and SSL, $SU = \emptyset$ sine the source domain is assumed to be labeled. Hence

$$S = \{X_S, Y_S\} = \{x_S^i, y_S^i\}_{i=1}^n \sim \{D_S\}^n, \tag{6.26}$$

with $|S| = n$ being the number of drawn samples (all labeled) and therefore $X_S = X_{SL} \subset X, Y_S = Y_{SL} \subset Y$.

For DA, it is assumed to have two data sets available. Let $T = TL \cup TU$ be the second data set T drawn i.i.d. from a domain D_T called *target domain* with a distribution over $X_T \times Y_T$,

$X_T \subset X, Y_T \subset Y$, and consisting of unlabeled TU and/or labeled TL samples.

$$TL = \{X_{TL}, Y_{TL}\} = \{x_T^j, y_T^j\}_{j=1}^{m-l} \sim \{D_T\}^{m-l}; \tag{6.27}$$

$$TU = \{X_{TU}, Y_{TU}\} = \{x_T^j\}_{j=m-l+1}^{m} \sim \{D_T^X\}^l; \tag{6.28}$$

with $|T| = m$ being the number of drawn target samples, therefore $X_{TL} \subset X_T \subset X$, $Y_{TL} \subset Y_T \subset Y$ and $X_{TU} \subset X_T \subset X$. Let D_S^X, D_S^Y and D_T^X, D_T^Y be the marginal distributions of D_S and D_T over X_S, Y_S and X_T, Y_T respectively.

Pseudo-Labels: It is assumed softmax output unit, categorical cross entropy is selected as loss function $\mathbf{L} = \mathbf{L}_{CE}$. The goal is to build a statistical model $h : X \rightarrow Y$ so that the classification error $\mathbf{L}$ is minimized. For labeled source data the following optimization problem is given:

$$\begin{aligned} h(\cdot) &= \min_{\tilde{h}} \mathbf{L}(\tilde{h}(X_S), Y_S) \\ &= \min_{\tilde{h}} \sum_{\left(x_S^i, y_S^i\right) \in S} \mathbf{L}\left(\tilde{h}(x_S^i), y_S^i\right). \end{aligned} \tag{6.29}$$

where (x_S^i, y_S^i) is any paired source (input, output) sample drawn i.i.d from D_S.

Pseudo-labels $\hat{y}_T$ are classes assigned to unlabeled target samples by using a classifier trained on $S = SL$ and selecting the class with maximum predicted probability:

$$\hat{y}_T^j = \operatorname{argmax} h(x_T^j) \tag{6.30}$$

The process of generating pseudo-labels can be accomplished in a so called *online* and an *offline* manner:

- **Offline pseudo-labeling**: A classifier is trained with the available labeled data for a fixed amount of iterations. A probability threshold is defined. Then the classifier is used to generate pseudo-labels for unlabeled target data. Pseudo-labels with a softmax output higher than the threshold are selected, added to the labeled data and used for the next training phase to further improve the accuracy of the classifier model especially for target data samples.

- **Online pseudo-labeling**: The threshold setting and pseudo-labeling is being performed in a temporary manner on each mini-batch during the classifier training. Sample weightage based on the number of training steps are introduced to account for the lower classification accuracy especially at the beginning of the training. Slowly increasing the weightage is recommended as precaution measure to avoid local minima. The loss function $\mathcal{L}$ is defined as:

$$\mathcal{L} = \mathbf{L}(h(X_S), Y_S) + \alpha(t)\mathbf{L}(h(X_T), \hat{Y}_T) \tag{6.31}$$

where $\alpha(t)$ is the weightage function respective schedule based on total number of steps T defined as:

$$\alpha(t) = \begin{cases} 0 & \text{if } t < T_1 \\ \frac{t-T_1}{T_2-T_1}\alpha_h & \text{if } T_1 \leq t < T_2 \\ \alpha_h & \text{if } T_2 \leq t \end{cases} \tag{6.32}$$

with T_1, T_2 and α_h are constants that shall be oriented on the percentile of the total number of training steps T and the problem at hand. The overall ratio of the unlabeled data with respect to the labeled data is influential but mostly dependent on the availability of data.

AdaMatch: AdaMatch by Berthelot et al. [19] is suited for both un- and semi-supervised learning and domain adaptation. It exploits pseudo-labeling and enriches it with consistency regularization, addressing the distribution shift between source and target domains present in the *batch norm statistics*, flexible pseudo-label confidence threshold and modified version of distribution alignment. Let $h_{ada} : X \rightarrow Z := R^C$ be a classifier model with X image input and Z logit output for each of the C classes. Then, AdaMatch follows the following pipeline:

- **Image Augmentation**: Each minibatch X_{SL}, X_{TU} are augmented both with a with a *weak* and a *strong* augmentation, respectively $X_{SL}^{aug} = \{X_{SL,w}, X_{SL,s}\}$, $X_{TU}^{aug} = \{X_{TU,w}, X_{TU,s}\}$. The logits of the augmented images are computed as:

$$\{Z'_{SL}, Z_{TU}\} = h_{ada}(X_{SL}^{aug}, X_{TU}^{aug}) \tag{6.33}$$
$$Z''_{SL} = h_{ada}(X_{SL}^{aug}), \tag{6.34}$$

where Z'_{SL} and Z''_{SL} could be different because of the batch normalization statistics, since they are obtained from differently assembled batches (respectively with and without the target domain images).

- **Random Logit Interpolation**: The logits Z'_{SL} and Z''_{SL} are interpolated:

$$Z_{SL} = \lambda Z'_{SL} + (1 - \lambda) Z''_{SL}, \tag{6.35}$$

where λ is sampled from a uniform distribution with range $(0, 1)$ to weight each logit.

- **Distribution Alignment**: The idea is to encourage the target unlabeled distribution of pseudo-labels to follow the distribution of the source labeled data assuming similar source and target label distributions. AdaMatch estimates the source label distribution from the output of the model on the labeled source data. Given the logits for weakly augmented source $Z_{SL,w}$ and target $Z_{TU,w}$ samples, the pseudo-labels are computed:

$$\hat{Y}_{SL,w} = \text{softmax}(Z_{SL,w}), \qquad \hat{Y}_{TU,w} = \text{softmax}(Z_{TU,w}). \tag{6.36}$$

Distribution alignment is applied by multiplying the target unlabeled pseudo-labels by the ratio between the expected value of the weakly augmented source pseudo-labels $\mathbb{E}[\hat{Y}_{SL,w}]$ and the expected value of the weakly augmented target pseudo-labels $\mathbb{E}[\hat{Y}_{TU,w}]$:

$$\tilde{Y}_{TU,w} = \text{normalize}\left(\hat{Y}_{TU,w}\frac{\mathbb{E}[\hat{Y}_{SL,w}]}{\mathbb{E}[\hat{Y}_{TU,w}]}\right), \tag{6.37}$$

where normalize makes the distribution sum to 1 again.

- **Relative Confidence Threshold**: The threshold c_τ, applied to decide which target pseudo-labels are used for calculating the categorical cross entropy loss during training, is computed on the fly by taking the average of the weakly augmented source pseudo-labels on the highest predicted class, multiplied by a constant parameter τ. Let $n_S L$ define the number of available labeled source samples. Then:

$$c_\tau = \frac{\tau}{n_{SL}} \sum_{i=1}^{n_{SL}} \max_{j \in [1,\ldots,k]} \left(\hat{Y}^{(i,j)}_{SL,w}\right). \tag{6.38}$$

c_τ is then used to compute a mask:

$$mask^{(i)} = \max_{j \in [1..k]} \left(\hat{Y}^{(i,j)}_{TU,w}\right) \geq c_\tau. \tag{6.39}$$

- **Loss Function**: The loss for the source labeled, both weakly and strongly augmented data is defined as:

$$\mathcal{L}_{source} = \frac{\tau}{n_{SL}} \sum_{i=1}^{n_{SL}} \mathcal{L}\left(Y_{SL}^{(i)}, Z_{SL,w}^{(i)}\right) + \\ + \frac{\tau}{n_{SL}} \sum_{i=1}^{n_{SL}} \mathcal{L}\left(Y_{SL}^{(i)}, Z_{SL,s}^{(i)}\right) \tag{6.40}$$

The loss for the unlabeled, masked target data is defined as:

$$\mathcal{L}_{target} = \frac{\tau}{n_{TU}} \sum_{i=1}^{n_{TU}} \mathcal{L}\left(\text{stop\_grad}(\tilde{Y}_{TU,w}^{(i)}), Z_{TU,s}^{(i)}\right) mask^{(i)} \tag{6.41}$$

where $L = L_{CE}$ is the categorical cross entropy loss and stop_grad is a function that prevents gradient back-propagation on pseudo-labels, which is a standard practice in SSL that improves training.

The final loss $\mathcal{L}_{final}$ is then given by

$$\mathcal{L}_{final} = \mathcal{L}_{source} + \mu(t)\mathcal{L}_{target} \tag{6.42}$$

where $\mu(t)$ is a warm-up function that schedules the weight to be assigned to the unlabeled loss term similar to the weightage function within the online pseudo-labeling. [19] proposes to use

$$\mu(t) = \frac{1}{2} - \frac{\cos\left(\min(\pi, \frac{2\pi t}{T})\right)}{2}, \tag{6.43}$$

where T again is the total number of training steps. In practice, this function goes from 0 to 1 in the first half of the training and stays at 1 for the second half.

Online and Offline pseudo-labeling as well as AdaMatch is used as benchmark domain adaptation method in Chapter 10: Defect Classification (DC).

6.4.2 Feature- and Instance-based Domain Adaptation

Domain adaptation is a field of transfer learning where data driven methods are used to enable usage of models for multiple data sets following different distributions. Different paradigm of TL are defined with feature-based and instance-based two important categories especially useful in the field of ML.

The following notations are used throughout the whole section. Let X define the input space and Y the label or target space with $Y = \{0, 1\}$ for classification and $Y \subseteq \mathbb{R}$ for regression task. In the DA setting it is assumed that the training sample S is drawn according to a source distribution D_S, while target data is drawn according to a target distribution D_T that may differ. Hence let $S = \{x_i, y_i\}_{i=1}^{n} \sim \{D_S\}^n$ with $x_i \in X_S \subset X$ is the input and $y_i \in Y$ is the corresponding output of the source domain. Similar we denote $T = \{x_i, y_i\}_{j=1}^{m} \sim \{D_T\}^m$ with $x_j \in X_T \subset X$ is the input and $y_j \in Y$ is the corresponding output of the target domain (predicting those target labels is the goal, here we assume that labels are also available but only used for testing not for training). Let $f : X \rightarrow Y$ be the (unknown) labeling/target function. Let $\mathbf{L} : Y \times Y \rightarrow \mathbb{R}$ describe a loss function over pairs of labels and let $L_D(f, g) = \mathbb{E}_D|\mathbf{L}(f, g)| = \mathbb{E}_{x \sim D}|\mathbf{L}\left(f(x), g(x)\right)|$ describe the loss over two functions $f, g : X \rightarrow Y$ and any distribution or domain D over X. Let a hypothesis class $\mathcal{H}$ be a set of all possible modeling functions h that are considered. Then the domain adaptation task consists of finding $h \in \mathcal{H}$ such that the corresponding loss $L_{D_{S,T}}(h, f) = \mathbb{E}_{D_{S,T}}|\mathbf{L}(h, f)|$ is minimized according to source D_S as well as target domain D_T.

Feature-based models

The goal of feature-based methods is to find features for source and target such that they show in best case identical behavior and distribution in the sense of being not domain specific and domain invariant. In general, features are projected into a new feature representation space. Then a task model is trained on the new input features. Learning an optimal projection able to still solve the task without keeping domain specific information in the data at hand is the overall goal. For a visualization see Figure 6.5.

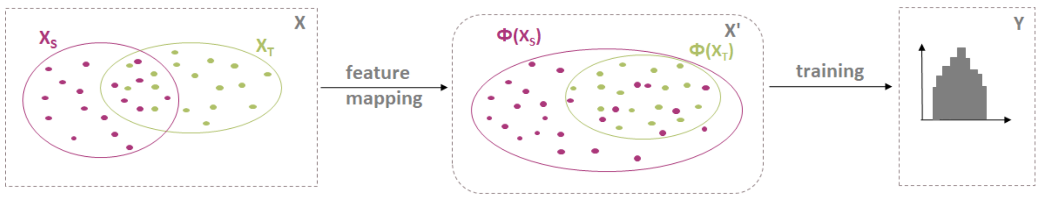


Figure 6.5: Visualization of feature-based DA methodology. Graphical representation of feature-based domain adaptation methods. X describes the input space,$\Phi : X \rightarrow X'$ describes the mapping with X' the new (lower dimensional) feature space and Y the output space.

Correlation Alignment (CORAL) Correlation Alignment (CORAL) by Sun et al. [188] is an unsupervised domain adaptation method that aligns second order statistics of source and target domain. It aligns the original input spaces hence does not work with lower dimensional latent representations. It is simple hence fast and easily applicable to larger data sets with a high number of samples. First, CORAL that works with linear transformation of the source features is described. Therefore, the mathematical formula and the algorithm for CORAL for unsupervised DA given in [188] is summarized.

Since Coral is unsupervised no target labels are used except for testing. Let $\Upsilon : X_S \rightarrow X_T$ with $\Upsilon(X_S) = X_S^* A$ describe the feature transformation of the source domain. Let μ_S, μ_T be the feature mean and C_S, C_T the covariance matrices. Then, the distance between the covariance matrices (assuming normalized features with zero mean) is minimized by:

$$\min_A \|C_{\hat{S}} - C_T\|_F^2 = \min_A \left\|A^T C_S A - C_T\right\|_F^2 \tag{6.44}$$

where A defines the linear transformation that is applied to the source, $C_{\hat{S}}$ describes the covariance of the transformed source features and $\|\cdot\|_F^2$ denoting the squared Frobenius norm selected as distance metric. The minimization objective is called CORAL loss. In order to solve equation (6.44) Algorithm 1 in [188] is followed: First, the covariance matrices is computed, followed by whitening the source and then recoloring it with the target covariance. After the source features are aligned, another prediction model can be trained on the adjusted features. For a comparison of CORAL with other related methods we refer to the corresponding section in [188] and [189].

CORAL can be extended to a DL-based version called DeepCoral as presented in detail by Sun and Saenko [189]. It uses nonlinear transformations by correlating activation layer. Therefore a DL network architecture is introduced that is trained with the typical classification or regression loss using labeled source data plus the above described CORAL loss. Source and target batch covariance are used to compute CORAL loss while it can be defined on which and on how many layers the loss is computed. Using the notation introduced in [189] for a single feature layer, CORAL loss is defined in case of a d-dimensional activation layer output of source respective target input (where the corresponding weights needs to be learned) via.

$$L_{CORAL} = \frac{1}{4d^2} \|C_S - C_T\|_F^2$$

where the covariance matrices C_S, C_T are defined as batch covariances. The gradient for backpropagation can be computed using the chain rule and network parameters are shared for source and target (in the sense of copying the network and use them in parallel, one for source and one for target input). The final loss is defined as

$$L_{DeepCORAL} = \mathbf{L} + \sum_{i=1}^{t} \lambda_i \cdot L_{CORAL}$$

with t defining the number of CORAL layers used, λ_i the corresponding loss weight and $\mathbf{L}$ a classifier or regression loss of choice for the modeling task at hand.

Strong points - besides its simplicity- are avoidance of overfitting by simply using classification/regression loss and avoidance of over simplistic features without task specific information. CORAL is applied in Chapter 8: Virtual Metrology (VM) and Chapter 9: Predictive Maintenance (PdM).

Transfer component analysis (TCA) Transfer component analysis (TCA) by Pan et al. [156] is an unsupervised domain adaptation method that tries to learn so called *transfer components* in a reproducing kernel Hilbert space (RKHS) using maximum mean discrepancy (MMD). The idea is to project the original features into a latent feature space such that the transferred data distributions are close to each other. The new representations can then be used to train a machine learning model dedicated to the labeled source domain that is also applicable to the target domain.

Let D_S^X and D_T^X be the marginal distributions of X_S and X_T. Let H be a universal RKHS and $\Theta : X \rightarrow H$ the nonlinear transformation to be found. Let $X_S^{'} = \{x_{S_i}^{'}\} = \{\Theta(x_{S_i})\}$, $X_T^{'} = \{x_{T_i}^{'}\} = \{\Theta(x_{T_i})\}$ and $X^{'} = X_S^{'} \cup X_T^{'}$ be the transformed and combined input sets from the source and target domains. Then the distance between two distributions in H is defined using the empirical estimate, two sample sets S, T with $|S| = n, |T| = m$ and MMD:

$$DIST(X_S^{'}, X_T^{'}) = \left\| \frac{1}{n} \sum_{i=1}^{n} \Theta(x_{S_i}) - \frac{1}{m} \sum_{j=1}^{m} \Theta(x_{T_j}) \right\|_H^2 . \qquad (6.45)$$

Θ can be found by minimizing (6.45). This is done by transforming it into a kernel learning problem. Using the kernel trick with $k(x,x_j) = \Theta(x_i)^T\Theta(x_j)$ we get:

$$DIST(X_S^{'}, X_T^{'}) = tr(KL); K = \begin{bmatrix} K_{S,S} & K_{S,T} \\ K_{T,S} & K_{T,T} \end{bmatrix} \tag{6.46}$$

K is a $(n+m) \times (n+m)$ kernel matrix, $K_{S,S}$, $K_{T,T}$ and $K_{S,T}$ respectively $K_{T,S}$ are the kernel matrices defined by the kernel k on the data in the source domain, target domain, and cross domains; and $L = [L_{ij}] \geq 0$ with $L_{ij} = \frac{1}{n}$ if $x_i, xj \in X_S$ respective $L_{ij} = \frac{1}{m}$ if $x_i, xj \in X_T$ and otherwise $L_{ij} = -\frac{1}{mn}$.

Learning the kernel function k can be solved by learning the matrix K. This is done in [156] the following way: Let $\tilde{W}$ be a $(n+m) \times l$ dimensional transformation matrix mapping the feature space into a l-dimensional latent space, K is decomposed using the empirical kernel map and $W = K^{-1/2}\tilde{W} \in \mathbb{R}^{(n+m)\times l}$. Then the new kernel matrix respective kernel function is defined as:

$$\tilde{K} = (KK^{-1/2}\tilde{W})(\tilde{W}^T K^{-1/2} K) = KWW^TK$$
$$\tilde{k}((x_i, x_j) = k_{x_i}^T WW^T k_{x_j}.$$

Finally the distance between the transformed input spaces can be reformulated:

$$DIST(X_S^{'}, X_T^{'}) = tr(KL) = tr((KWW^TK)L) = tr(W^TKLKW). \tag{6.47}$$

The final kernel learning problem then has the following formalism:

$$\min_W tr(W^TW) + \mu\, tr(W^TKLKW) \tag{6.48}$$
$$s.t. W^TKHKW = I \tag{6.49}$$

where the side condition is introduced to avoid the trivial solution $W = 0$, $tr(W^TW)$ is a regularization term, μ is a trade off parameter, I the identity matrix, $H = \mathrm{I}_{n+m} - \frac{1}{n+m}11^T$ is the centering matrix with $1 \in \mathbb{R}^{n+m}$.

For further details, use cases and performance discussion see [156]. TCA is applied as domain adaptation benchmark in Chapter 9: Predictive Maintenance (PdM).

Subspace Alignment (SA) Subspace Alignment (SA) presented by Fernando et al. [64] linearly aligns subspaces generated by PCA; for details on PCA see the description further above based on [105]. First each source and target domain is mapped to its according subspace, then a linear transformation is learned in order to adapt the projected source to the projected target.

Let $S' = \{x'_{S_i}\}, T' = \{x'_{T_i}\}$ be the PCA projected input sets from the mapped source X'_S and target X'_T domains. Let $\phi : X'_S \to X'_T$ be the linear transformation to be found with $\phi(X'_S) := X'_S{}^* M$ and matrix M. Then:

$$M^* = \arg\min_M \|S'M - T'\|_F^2$$

with $\|\cdot\|_F$ describing the Frobenius norm. $M* = S'^* T'$ holds since the Frobenius norm is invariant towards orthogonal transformation.

For further instructions on unique hyperparameter tuning for the number of principle components it is referred to [189]. SA was introduced as unsupervised DA method for classification task. Overall benefits lie in the simplicity hence speed of the method while presenting comparative high accuracy. SA is used as benchmark domain adaptation model in Chapter 8: Virtual Metrology (VM) in the heterogeneous DA part as well as in Chapter 9: Predictive Maintenance (PdM).

6.4.3 Instance-based models

The idea behind instance-based methods is a weighting of instances respective samples such that the difference between source and target distribution is minimized. Different distance measures are chosen depending on the task at hand and are characteristic for each method. The reweighted (source) samples (in the unsupervised case where no labeled target data is available) are used to train a new model that solves the task for both domains. For a visualization see Figure 6.6.

Nearest Neighbors Weighting (NNW) Based on Dudani [58] with the idea of classifying unlabeled data by using their nearest neighbors, Nearest Neighbors Weighting NNW by Loog [136] transfers this idea to a DA setting. In more detail, it addresses a covariate shift, meaning the conditional probabilities of the output values y given the input vector x from

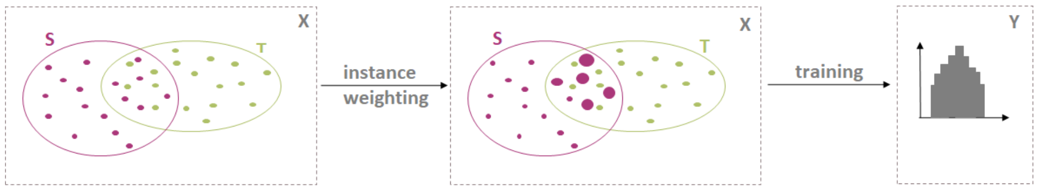


Figure 6.6: Visualization of instance-based DA methodology. Graphical representation of instance-based domain adaptation methods. X describes the input space and Y the output space. The reweighted samples are visualized via decreased or increased sample size.

source respective target are equal but input distribution D_S and D_T respective source and target domain vary a lot from each other. [136] recaps that the expected loss can be estimated as

$$\mathbb{E}_{D_S}\left[(w(x)\mathbf{L}(h(x),y)\right] - \mathbb{E}_{D_T}\left[\mathbf{L}(h(x),y)\right] \tag{6.50}$$

with $x \in X, \mathbf{L}(h(x),y)$ the loss function with $h \in \mathcal{H}$ ($\mathcal{H}$ hypothesis class meaning class of all possible modeling functions) and

$$w(x) = \frac{D_T(x)}{D_S(x)} \tag{6.51}$$

The following two step approach is then applied to minimize the loss of the target domain based on the source domain and best modeling function h:

1. determine an estimate $\hat{w}$ for w based on training samples
2. minimize the empirical loss using the weighted samples with respect to $h \in \mathcal{H}$.

Due to its simplicity the approach is very easy to implement and very efficient since no optimization needs to be solved. First a lattice structure is defined on the input feature space via Voronoi or Dirichlet tessellation. A Voronoi cell or region is defined via a center and all samples of the space that are closer than to any other center point. The source domain is used as centers and target data is associated to a specific cell. Let x_{S_i} be a center point from source and x_{T_i} the target samples associated to it. Then the weights of each source sample is defined as number of target points associated to its Voronoi region via

$$\hat{w}(x_{S_i}) = |V_{S_i} \cap \{x_{T_j}\}_j| \tag{6.52}$$

with $||\cdot$ defining the cardinality of a set.In case of too many empty Voronoi cells, Laplace smoothing is applied where every cell is expected to have already one count:

$$\hat{w}(x_{S_i}) = |V_{S_i} \cap \{x_{T_j}\}_j| + 1. \tag{6.53}$$

[136] shows its advantages towards KLIEP (another importance weighting method presented by Sugiyama et al. [187]). NNW is applied as domain adaptation benchmark in Chapter 9: Predictive Maintenance (PdM).

Two Stage Transfer AdaBoost for Regression (TrAdaBoostR2) Two Stage Transfer AdaBoost for Regression (TrAdaBoostR2) presented by Pardoe and Stone [158] is based on the adaptive boosting algorithm by Freund and Schapire [65] and adopted to a DA setting. Boosting in general refers to the idea that a ensemble of weaker models can outperform a strong one. TrAdaBoostR2 is a supervised DA method for regression and makes use of the available labeled target data to train together with the labeled source data an regression model and use the errors then to update source as well as target sample weights to improve transfer. [158] presents the following algorithm:

1. select initial weight vector $\tilde{w}$ (size fits the number of available labeled data samples from source $|S| = n_S$ and target $|T| = n_T$ combined namely n_S+n_T) and normalize it via

$$w_i = \frac{\tilde{w}_i}{\sum_{j=1}^{n_S+n_T} \tilde{w}_i^j}. \tag{6.54}$$

2. train a model with weighted labeled data samples from source and target and get a hypothesis $h : X \to Y$

3. evaluate h and get back the prediction error from labeled source and target data using $\mathbf{L}$ as loss function

$$\epsilon_S = \mathbf{L}(h(S), Y_S) \tag{6.55}$$
$$\epsilon_T = \mathbf{L}(h(T), Y_T). \tag{6.56}$$

4. Normalize error vectors and compute weighted error E of target samples with w_T the part that belongs to the labeled target samples

$$E_T = \frac{1}{n_T} w_T^T \epsilon_{T,norm}) \tag{6.57}$$

5. Define with N number of overall iterations

$$\beta_T = \frac{E_T}{1 - E_T} \tag{6.58}$$

$$\beta = \frac{1}{1 + \sqrt{2 \ln n_S / N}} \tag{6.59}$$

and use this to update the sample weights

$$w_S = w_S \beta^{\epsilon_{S,norm}} \tag{6.60}$$

$$w_T = w_T \beta_T^{-\epsilon_{T,norm}} \tag{6.61}$$

The final predictions are generated by the weighted median of the $N/2$ model functions h.

TrAdaBoostR2 is applied as domain adaptation benchmark in Chapter 9: Predictive Maintenance (PdM).

7 Metrics and Losses

Selection of correct, meaningful and strong metrics is essential for trustworthy evaluation of models. Losses are used to quantify how good a model is performing (also during training) and are the basis for optimization strategies like gradient decent.

To measure the success of the alignment besides overall performance of different models, two groups of losses are introduced namely **Distribution-based loss** and **Performance-based loss**.

Let X be defined as the input space and Y as the output space. Let $\mathbf{L} : Y \times Y \to \mathbb{R}$ define a loss function. Let $h_P : X \to Y$ define a a statistical task model and let h_P be parameterized by θ_P and $P(x, \theta_P)$ be the representation of our statistical model from input to output where P is the model function with parameters θ_P that outputs the predicted target value for $x \in X$.

7.1 Performance-based Loss

Let $y^{true} = \{y_i^{true}\}_{i=1}^n \sim \{D^Y\}^n$ the labels from a sample set from source or target and $y^{pred} = \{y_i^{pred}\}_{i=1}^n = \{P(x_i)\}_{i=1}^n$ the predicted values computed from $\{x_i\}_{i=1}^n \sim \{D^X\}^n$. Let $\bar{y}^{true}$ stands for the mean value and Var for variance, the square of the standard deviation.
Then the following performance-based loss function are defined:

Mean absolute error (MAE) Mean absolute error **(MAE)** is one of the most common loss functions and defined as

$$\text{MAE}(y^{true}, y^{pred}) = \sum_{i=1}^{n} \left| y_i^{true} - y_i^{pred} \right| \tag{7.1}$$

Root mean square error (RMSE) Root mean square error **(RMSE)** is a common metric often used in literature for PdM thanks to its robustness and interpretabiliy.

$$\text{RSME}(y^{true}, y^{pred}) = \sqrt{\sum_{i=0}^{n} \frac{(y_i^{pred} - y_i^{true})^2}{n}}. \tag{7.2}$$

Maximal residual error (ME) Maximal residual error **(ME)** is defined as

$$\text{ME}(y^{true}, y^{pred}) = \max_i \left| y_i^{true} - y_i^{pred} \right| \tag{7.3}$$

R-square (R2) R-square **(R2)** is the coefficient of determination also called regression score function and defined as

$$\begin{aligned} \text{R2}(y^{true}, y^{pred}) &= 1 - \frac{\sum_{i=1}^{n} \left(y_i^{true} - y_i^{pred} \right)^2}{\sum_{i=1}^{n} (y_i^{true} - \bar{y}^{true})^2} \\ &= 1 - \frac{\sum_{i=1}^{n} \left(y_i^{true} + y_i^{pred} \right)^2}{Var(y^{true})} \end{aligned} \tag{7.4}$$

Explained variance regression score (EV) Explained variance regression score **(EV)** is defined as

$$\text{EV}(y^{true}, y^{pred}) = 1 - \frac{Var(y^{true} - y^{pred})}{Var(y^{true})} \tag{7.5}$$

R2 and EV are equal if the mean error is equal to 0.

7.2 Distribution-based Loss

Kolmogorov–Smirnov (KS) The Kolmogorov–Smirnov **(KS)** test is a nonparametric test to compare two one dimensional data distributions. Either a sample set is compared to a given data distribution (called one-sided KS-test) or two sample sets are tested against each other (called two-sided KS test). For our multidimensional input space we use it for a univariate feature comparison and compute the average over all univariate test statistics. The resulting statistics are the so called p-value - the probability that one achieves similar or more extreme values under the null hypothesis - and the ks-score - the absolute maximum distance between the two domains. The smaller the p-value and the higher the ks-score the more likely it is to reject the null hypothesis.

T-distributed Stochastic Neighbor Embedding (t-SNE) T-distributed Stochastic Neighbor Embedding **(t-SNE)** is a machine learning algorithm developed for data visualization and especially suitable if the data contains nonlinear dependencies [206]: t-SNE is based on gradient-descent minimization of the Kullback-Leibler **(KL)** divergence of the two domains under investigation. The KL divergence itself is presented as metric for comparison of inner domain distance - meaning the divergence between training and test set of one domain - with outer domain distance - meaning the divergence between training respective test sets of source versus target domain.

Frechet inception distance (FID) Frechet inception distance **(FID)** is well known in the field of GAN to evaluate the quality of generated data ([87]). A general introduction to the Frechet distance between multivariate functions is given in [56]. The FID metric is defined as the squared Wasserstein metric between two multidimensional gaussian distributions. Let $\mathcal{N}(\mu_1, \Sigma_1)$ and $\mathcal{N}(\mu_2, \Sigma_2)$ two multivariate gaussian distributions with μ_1, μ_2 the corresponding mean and Σ_1, Σ_2 the corresponding covariance matrices. Then FID is defined as:

$$FID = ||\mu_1 - \mu_2||^2 + tr\left(\Sigma_1 + \Sigma_2 - 2 * (\Sigma_1\Sigma_2)^{1/2}\right). \tag{7.6}$$

Structural Similarity Index (SSIM) Structural Similarity Index **(SSIM)** is a metric that takes structural information of data presented as images into account. The main features are contrast, structure and luminance and the loss value is computed based on those feature. Inspired by segmentation task, one application to a computer vision related domain adaptation case is given by Gokaslan et al. [73]. Let x,y be two non negative images signals with discrete representation, then the mean intensity in x is defined as

$$\mu_x = \frac{1}{N}\sum_{i=1}^{N} x_i. \tag{7.7}$$

For luminance comparison we then define the function l as

$$l(x,y) = \frac{2\mu_x\mu_y + C_1}{\mu_x^2 + \mu_y^2 + C_1}. \tag{7.8}$$

with $C_1 = (K_1 L)^2$ where $K_1 << 1$ is a small constant and L the dynamic range of pixel values. Contrast is given by an unbiased estimate of the standard deviation of the signal contrast as

$$\sigma_x = \left(\frac{1}{N-1}\sum_{i=1}^{N} (x_i - \mu_x)^2\right)^{1/2}. \tag{7.9}$$

The contrast comparison $c(x,y)$ between x and y is then given as

$$c(x,y) = \frac{2\sigma_x\sigma_y + C_2}{\sigma_x^2 + \sigma_y^2 + C_2}. \tag{7.10}$$

with $C_2 = (K_2 L)^2$ where $K_2 << 1$ is a small constant and L the dynamic range of pixel values. For structural comparison we first define a correlation coefficient σ_{xy} as

$$\sigma_{xy} = \frac{1}{N}\sum_{i=1}^{N} (x_i - \mu_x)(y_i - \mu_y). \tag{7.11}$$

Then the structure comparison function is defined (using again a third constant C_3 as before) as

$$s(x,y) = \frac{\sigma_{xy} + C_3}{\sigma_x\sigma_y + C_3}. \tag{7.12}$$

Overall the Structural Similarity (SSIM) index between image signals x, y is given by

$$\begin{aligned} SSIM(x,y) &= [l(x,y)]^\alpha + [c(x,y)]^\beta + [s(x,y)]^\gamma \\ &\overset{*}{=} \frac{(2\mu_x\mu_y + C_1)(2\sigma_x\sigma_y + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)} \end{aligned} \tag{7.13}$$

with $\alpha, \beta, \gamma > 0$ and set to 1 for $\overset{*}{=}$ and $C_3 = C_2/2$. $SSIM \in [-1, 1]$ hence the index is a value between -1 and 1 where values close to 1 predict a high similarity and values close to -1 indicates significant differences.

SSIM is designed to fulfill the properties of a similarity measure meaning symmetry, boundedness and unique maximum. It is locally convex that makes it a good candidate for optimization. In order to apply the computation above to time series data, we expand its dimension. Hence the last axis of the time series is multiplied and filled with copies of the corresponding single value. The SSIM can be directly applied to the expanded time series data.

Part III

Applications and Case Studies

8 Virtual Metrology (VM)

Process monitoring and process control are key factors for a highest of standard production. A detailed view on technologies, process, production and yield including monitoring, modeling, diagnosis, control and analysis is presented in May and Spanos [146]. The goal of metrology technologies is to estimated quantities that are important for quality/control purposes but are costly or even impossible to measure. Such important quantities are typically measured in so-called metrology stations where multiple product-related measurements are taken after the process is finished. Metrology is monitored via univariate or multivariate control charts where upper control limits (UPC) and lower control limits (LCL) are defined depending on application and applied (statistical) methods. Due to cost reduction and time restriction, only a subset of processed wafers are physically measured after a process.

Virtual metrology (VM) (other terms like soft sensing are used outside of semiconductor manufacturing) modules exploit the availability of data already collected by equipment or information systems (with no additional costs) to make use of statistics to estimate virtual measures for wafers where no real and physical metrology is performed. VM technologies have a long history in being applied in production with many beneficial impacts, namely:

- *costs* metrology measures are performed for a subset of the produced products/process iterations hence supporting cost and process time reduction
- *quality* - the availability of estimations for important process quantities allow to increased process monitoring capabilities;

- *control* - the usage of VM estimations in combination with real metrology can be used to update control modules (typically in Run-to-Run controllers) minimizing drifts, shifts or variance of the production [186];
- *data-driven sampling optimization* - it has been shown how VM estimation-based decision making in production sampling selection can lead to increased production defect detectability [63].

Therefore, VM stands for methods that predict wafer properties based on sensor measurements taken during the process within the process equipment. Hence the missing values of the non tested products can be replaced by predicted values and used for analysis, control and quality assessment. A general introduction to process control including metrology is given in Chapter 2: Elements of Semiconductor Manufacturing.

The results presented in the first part of this chapter are covering the work published in Gentner et al. [70] and Gentner et al. [69] about homogeneous domain adaptation for a VM regression task. The extended benchmarking with focus on time series suitable DL architectures are obtained in collaboration with University of Padua, Information Engineering Department, Prof. Dr. Alessandro Beghi, Prof. Dr. Gian Antonio Susto and Filippo Dalla Zuanna. A manuscript submitted for publication is not yet published. The second part about heterogeneous DA is created under the supervision of University of Padua, Information Engineering Department, Prof. Dr. Alessandro Beghi and Prof. Dr. Gian Antonio Susto. A manuscript submitted for publication is not yet published.

8.1 Introduction

In this chapter the focus lies on two virtual metrology use cases related to advanced process control (APC) and statistical process control (SPC). APC describes the equipment and the process itself in the production line and measures, describes and monitors functionality and properties on the product itself - properties are referred to so called wafer state measurements - during as well as after one or multiple process steps as well as health status of process equipment. The process analysed here, is a plasma etching process. The plasma etching process runs on an equipment from a specific vendor, see Figure 8.1.

SPC stands for the part of metrology that is monitored via univariate or multivariate control charts where upper control limits (UPC) and lower control limits (LCL) are defined

(a) (b)

Figure 8.1: LAM Research plasma etching tools. Two etching tools from the semiconductor fabrication supplier Lam Research: (a) Lam Alliance series, (b) Lam 2300 series.

depending on application and applied (statistical) methods. Therefore, SPC contains optical or electrical measurements - often also referred to as inline measurements. Two examples of optical metrology measurements for etching process are layer thickness (LT) and critical dimension (CD). A visual graphic of those are presented in Figure 8.2.

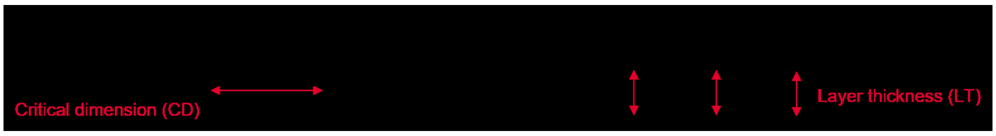


Figure 8.2: Visualization of metrology/inline measurements. Graphical visualization of two metrology measurements done in semiconductor manufacturing: on the left side it shows a gate width as example for critical dimension (CD), on the right a remaining layer thickness (LT).

The following sections are organized into different parts: First a literature deep-dive is presented with a focus on process control and virtual metrology with a focus on semiconductor manufacturing applications. Then, the first VM use case for identical-in.design equipment is introduced. The first use case is split into 2 parts: First, modeling approaches presented in [69] including linear regression, lasso regression, random forest and artificial neural networks are applied and discussed on the one hand for the VM prediction task and on the other hand for DBAM as method for domain adaptation. Second, the benchmarking is enriched by focusing on time series suitable DL architectures like LSTM, TCN and 1DCNN and their utility for domain adaptation with DBAM.

Next, the second VM use case for equipment with heterogeneous data representation is presented. Here, the domain adaptation method DBACS is applied and benchmarked. Benchmark models include correlation analysis, principle component analysis and multi-view learning using CCA besides CORAL as domain adaptation approach. In the last part, (equipment) matching enabled by DBACS is discussed before the chapter is closed with a summary.

8.2 Literature

Due to the high complexity and necessity of early fault detection in semiconductor manufacturing, metrology related research has a long history with Chen et al. [33] being one of the first papers presenting the basic idea behind VM and its advantages when it comes to process control (see also Su et al. [186]) and quality while technical requirements related to product as well as production are constantly increasing. Following [70], [69] we group the literature by data type, give a short overview of content and innovation there and have a separate look at current ML-based methods and domain adaptation related approaches.

Fault Detection and Classification (FDC) data

Most of the work related to semiconductor process and equipment are based on feature input data meaning descriptive statistics computed on the raw time dependent sensor measurements from the equipment. It is known under Fault Detection and Classification (FDC) data, also called keynumbers. Since those features are either way computed and collected thanks to standard process control mechanism, data is easy to access and methods are straight forward to apply. Based on the huge amount of available features, it is recommended to apply feature selection. While a broad range of feature selection techniques exists and in general are recommended as part of the data preprocessing (for a general review on feature selection techniques including filters see Saeys et al. [170]), literature presents more advanced and VM specific feature selection methods for example by Kang et al. [109]. It shows very promising results related to accuracy but also computational time. For the prediction itself based on FDC data, Lynn et al. [141] presents one of the first semiconductor VM example for an etching process using multiple linear regression and ANN compared with different preprocessing steps (PCA, correlation, stepwise analysis). A set of very common regression algorithm for VM prediction tasks are presented by Susto and Beghi [191]. Kernel based least square optimization like Ridge Regression, Lasso and Efficient Net are also successfully applied to a VM task by **(author?)** [Park and Kim]. It shows their potential as benchmarks against raw sensor based methods. Tree-based methods including some optimization steps applied on FDC for a VM task on a deposition process is presented by Chen et al. [31] and tested against different regression based - kernel, support vector and gaussian process - methods as well as ANN models to showcase their superiority.

Time series data

(author?) [Park and Kim] presents a Fussed-Lasso based approach using raw sensor data and proves it advantage against standard feature based approaches in VM. The usage of raw sensor measurements without preprocessing that resorts to descriptive statistics, prevents information loss. To avoid such information gaps on one side, but also keep the simplicity of a stationary data representation on the other side, Susto et al. [194] present an alternative supervised feature extraction approach based on regularization called SAFE.

Deep learning based methods

Like in many other industrial areas and applications (for a chemical manufacturing example including time series data see Guo et al. [79]) DL-based methods are successfully applied to VM in semiconductor manufacturing. While in early stages shallow Ml-based approaches prove to be superior (comparison of typical VM regression models are presented in Lynn et al. [142] and [210] - with Gaussian process as winner - for example), current research shows the superiority of DL-based methods as shown for example in Shang et al. [180] for soft sensing in general. A very valuable but cost-intense input for VM prediction task is optical emission spectroscopy (OES) data: Maggipinto et al. [144] presents a 1DCNN architecture suitable for time series data like OES. Other papers using the power of convolutional neural networks for VM prediction are Lee and Kim [125] using a "recurrent feature incorporated approach", Maggipinto et al. [145] and Yuan et al. [231] for automated feature extraction - the former based on OES images, latter for another industrial application - and Wu et al. [219] combining convolutional with gaussian processes to tackle a multi step problem. VM/soft sensor model based on LSTM is presented by Yuan et al. [232] for a pharmaceutical use case.

Matching and adaptation related methods

ML- and especially DL has been successfully researched and applied to a broad range of semiconductor applications. Models have to be made fast and flexible while being highly accurate and reliable especially under complex production environment - including up to thousand different product steps and constantly changing technology portfolios. But to address the effort of bringing those applications into non-standardized production, *scalability* of models and methods has to be discussed: Following the explanations in [70], [69], it is distinguish between two approaches that are often discussed in literature:

Matching: Matching - often done with production equipment hence referred to as equipment matching (EM) - is a highly researched topic in semiconductor manufacturing based on data as well as domain expert input to assure consistent processes and therefore quality and output. As presented in Chouichi et al. [42] EM aims at finding and eliminating setting and performance differences for increased control based on statistical methods applied on sensor data. It often relies on the explainability of methods to detect most important features and enable a root cause analysis. While [42] successfully applies traditional and well explainable statistical methods like partial least square, discriminant analysis and correlation, some patents like Heng et al. [86] addresses the matching approach with ANN. Matching is resulting in aligned and equalized equipment or process properties hence production standardization.

Domain Adaptation Domain adaptation (DA) is a purely data driven approach applied to deal and handle but not to necessarily eliminate existing differences in production; meaning occurring differences in collected data samples and hence different data distributions.

DA is based on domain adaptation theory introduced in Chapter 3.3. Introduced methods are developed to deal with situations where multiple data sets with different data distributions occur for identical or similar modeling tasks. Details about the related DL-based approach called domain adversarial neural network (DANN) used for domain adaptation are presented in 3.3 including application independent, theory focused literature. DA is a sub field of transfer learning (TL) developed to transfer a well performing statistical model trained on a specific data distribution to a second similar but non-identical data distribution for the same or a similar modeling task.

Two examples for transfer learning tested on VM in semiconductor manufacturing are presented in Kang [108] and Tsutsui and Matsuzawa [204]. Former focuses on transfer learning where previous gained knowledge is exploited to improve initialization hence accuracy and efficiency for a second dedicated model; more recently active learning with DANN applied to such a cold start problem is presented in Shim and Kang [183]. Latter uses optical emission spectroscopy (OES) data and convolutional neural network (CNN) plus additional preprocessing able to handle time series data with different length to create chamber invariant features. Semi-supervised approaches for transferable and invariant feature extraction methods are presented in Farahani et al. [61] and Li et al. [126]. Unsupervised domain adaptation (UDA) for VM is presented in Shim and Kang [183] addressing the VM critical aspect of time and cost savings by using DANN to achieve a good target model initialization hence improving wafer selection for active learning and therefore leading to a significantly shorter training time. For a domain adaptation method for fault diagnosis using MMD (see Chapter 3.2 for more details on probability distance metrics) to assess distribution similarity see Azamfar et al. [7].
While domain adaptation is a hot-topic for computer vision (see the survey by Wang and Deng [212]) as well as optimal transport tasks (see for example Courty et al. [45]), still only a limited amount of semiconductor manufacturing specific applications is available so far. The same goes for heterogeneous DA where overall a limited amount of publications are available: Yang et al. [224] with a classification task for heterogeneous information networks and Tsai et al. [203] as well as Fang et al. [60] with an - in this area most common - image to text transfer use case. Alipour and Tahmoresnezhad [3] combines distribution alignment via subspace mapping under consideration of aligned second order statistics with pseudo-labeling.
Another research field strongly related to heterogeneous DA is multi-view learning (MVL) (also known as data fusion or data integration); for python specifications see Perry et al. [161]. While covariance based analysis is heavily used in various fields as presented in the surveys by Sun [190] and Xu et al. [222], it is underrepresented for semiconductor related literature with Chen et al. [36] applying Canonical Correlation Analysis (CCA) for fault detection purposes and Yu et al. [230] applying correlation and CCA on evaluation of manufacturing performance systems being rare exceptions. A review of MVL in the DL setting is given by Yan et al. [223]; a similar collection called *CCA zoo* is provided by Chapman and Wang [30].

8.3 Virtual Metrology (VM) for Identical-in-design Equipment

We present an use case based on a plasma etching process during front-end production as described in Chapter 2. The data is taken from an single-wafer etching equipment with 4 identical-in-design and parallel running chambers presented in Figure 8.1 part a). The task is to create a virtual metrology model using raw sensor measurements and predict a specific metrology measurement taken from a metrology equipment on the wafer after the etching process is finished.

The data consists of physical sensor measurements like pressure, temperature, gas flows beside others taken during the etching process in a half second range depending on the running process. An optical end point detection is installed and applied. Layer thickness (LT) is selected as target inline measurement, it is visualized in Figure 8.2. Roughly 10 percent of the processed wafers get measured after the process due to high cost and time effort hence those can be used as labeled data. For the unmeasured wafer having a virtual prediction leads to advantages in process control. The prediction model is trained on a dedicated chamber.

Since a clear data shift is visible between the chambers, a direct model transfer usage is not possible. Hence as presented in Gentner et al. [69], the functionality of DBAM is applied to keep the high accuracy of a dedicated model and to showcase an explainable and scalable domain adaptation approach. DBAM is compared to other methods presented in literature. First the results presented in [69] are summarized. Second, the research is enriched with more time series specific benchmarking using different DL architectures and combinations of those.

Data Preparation

The etching chambers present overall 33 installed sensors where physical measurements in form of time series are collected during the process. The data under consideration for this use case is restricted to one recipe and one product group running parallel on two different identical-in-design etching chambers. We consider a time period of around 2 years where no statistical significant data shifts besides chamber differences occur. Multiple inline measurements are taken for sampled wafers after the process, we select layer thickness (LT) as target variable since it shows high dependency on the etching itself and there is no direct need to consider multiple related process steps before; like deposition and lithography for example where its influence on resulting metrology can be significant, see Chouichi et al. [42] for a discussion on that including ANOVA as method to select relevant operational data.

The source sample set respective samples from chamber 1 consists of around 1200 samples and the target from chamber 2 around 1100 samples, both including labels. To enable broad benchmarking, two data formats are considered as presented in [69]:

(1) Raw sensor measurements in form of time series where below described preprocessing steps are applied;

(2) Computed features based on descriptive statistics of the raw sensor measurements used in (1).

Set (2) is based on set (1) and no additional data cleaning, feature selection or feature engineering is applied to assure comparability of the different methods independent of the input data respective used information.

The following preprocessing steps are applied to the raw sensor measurements:

- removal of constant features and outliers (based on interquartile range (IQR))
- data normalization with min-max scaler (see sklearn python module by Pedregosa et al. [160])
- equal-distributed upsampling of timestamps to generate time series with equal length of size 1024.
- for (2): computation of average value per step before normalizing with min-max scaler.

After preprocessing is finished, data set (2) consists of 252 stationary features with the same amount of samples for both data sets. The layer thickness (LT) as target data for both chambers shows no significant distribution shift - meaning no target shift can be detected; see Figure 8.3 for a boxplot graph of one of the folds used later in cross validation.

8.3.1 Experimental Design: Part 1

Part 1 corresponds to results presented in [69] and the experiment is split in two phases:

In the first phase of the experiment we evaluate benchmark models for VM prediction task suitable for stationary data inspired by VM related literature. Selected methods suitable for stationary feature data are RR, LASSO, RF and ANN. Modeling details are presented in the following section. ANN is also used as prediction model for the alignment with DBAM in order to showcase its usability for stationary when it comes to scalability and generalization.

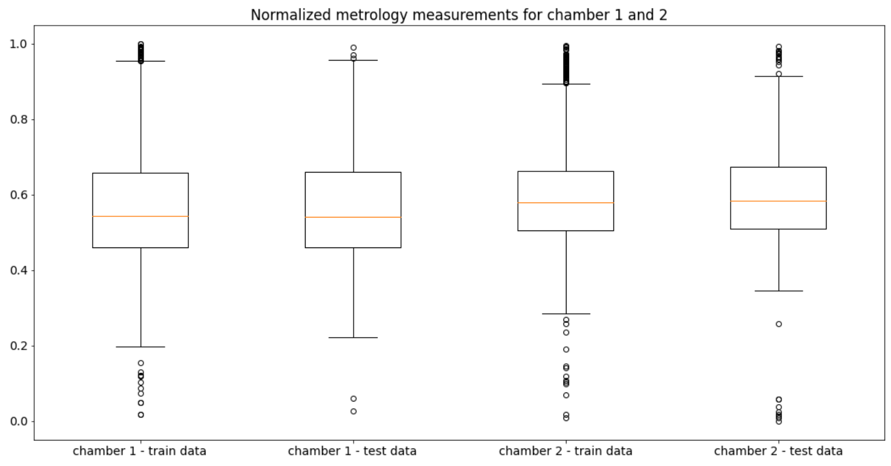


Figure 8.3: Boxplot of normalized layer thickness [70]. Boxplot graphs of normalized metrology measurements namely layer thickness from both chambers as presented in [70].

In the second phase of the experiment we evaluate two benchmark models presented in literature suitable for time series date: FL and dedicated 1DCNN model. Architectural details see below. Similar to the stationary data in order to prove scalability and generalization, first the dedicated 1DCNN model is trained as separate prediction benchmark and then used also as baseline predictor for a time series suitable DBAM. Since stationary and time series data are created based on the same raw sensor measurements, we analyse performance related to captured information in different data types.

Benchmark models

The benchmark model selection is based on pertinent literature, paper use case recommendations and suitability. For the VM prediction task, (ridge regression (RR), LASSO and fused lasso (FL) - all based on the classical OLS paradigm - are analyzed. Random forest (RF), a tree-based ensemble method, is also applied and evaluated as model for VM prediction. An introduction into the methodology of the selected benchmark models is given in Chapter 6.

ANN and 1DCNN are used on the one hand in phase 1 as VM prediction models but are also used for domain adaptation within DBAM in phase 2. For more details about neural network architectures see Chapter 3.1.

Metrics and losses

To measure the success of the alignment besides overall performance of different models we use two groups of losses; for detailed descriptions see Chapter 7:

For overall performance comparison, the following performance-based losses are applied:

- Mean absolute error (MAE), maximal residual error (ME), R-square/R2 and explained variance regression score (EV) are used.

For distribution comparison and alignment evaluation, the following distribution-based losses are applied:

- T-distributed Stochastic Neighbor Embedding (t-SNE) is used for visualization of distribution alignment;
- Kullback-Leibler (KL) divergence is used for comparison of inner domain distance - meaning the divergence between training and test set of one domain - with outer domain distance - meaning the divergence between training respective test sets of source versus target domain;
- Kolmogorov–Smirnov (KS) test is applied for a univariate feature comparison by computing the average over all univariate test statistics. Inner domain equality - meaning train sample set vs test sample set of one domain - is tested and compared to outer domain equality - meaning train respective test sample set from source versus target.

Models and Hyperparameters

5 fold cross validation is applied hence both data sets are split in 5 subsets and 4 merged sets are used as train and 1 as test set per fold. Architectures of all models stays fixed for all 5 folds. The presented results are performance averages over all 5 folds.

For selected benchmark models, the following available python implementations are used:

- *scikit-learn* by [160] plus available and in the python module included model specific hyperparameter optimization based on cross validation.

For the neural network based models, the following architectures are used for prediction:

- The ANN predictor model consists of 5 fully connected dense layers each with rectified linear unit (ReLU) activation function followed by batch normalization layer and a 10 percent dropout layer. The ANN is trained using stationary features by minimization of MAE using Adam optimizer including exponential learning rate decrease after 50 epochs and early stopping with learning rate $lr_P = 0.0001$, $min_\delta = 0.000001$ and $patience = 20$. For training ANN-based DBAM the previously on source trained ANN predictor is reused with fixed weights.

- The 1DCNN predictor model consists of two convolutional layers with dimension 16 and 8, kernel size 17 and stride of 2, exponential linear unit (elu) activation function, one max pooling layer, batch normalization layer in between, The last two layers after the flattening has dimension 16 and 1 and the output activation function is sigmoid. The 1DCNN is trained under usage of preprocessed sensor data by minimization of MAE using Adam optimizer with including exponential learning rate decrease after 50 epochs and early stopping with learning rate $lr_P = 0.00001$, $min_\delta = 0.00001$ and $patience = 20$. For training 1DCNN-based DBAM the previously on source trained 1DCNN predictor is reused with fixed weights.

All DBAM models are trained using Algorithm 4.1 presented in Chapter 4. The predictor models are reused and trained weights are frozen. For the other DBAM parts the following architectures are used:

- The domain discriminator for the ANN has 4 fully connected dense layers with Leaky ReLU activation function and linear output function. The aligner in form of an autoencoder consist of 3 dense layers plus ReLU activation function in the encoder part as well as in the decoder part plus batch normalization in between. The output activation function is linear. The aligner is pretrained to mirror the target data using also Adam optimizer with MAE as loss function to ensure a good initialization. For training ANN-based DBAM the previously on source trained ANN predictor is reused with fixed weights.

- The domain discriminator for the 1DCNN has 2 convolutional layers of dimension 32 and 16, kernel size 17, valid padding and leaky ReLU activation function, max pooling of size 4 between the two layers followed by a flattening layer and 6 dense layers again with leaky ReLU activation and linear output function. The aligner in form of an autoencoder consist of 2 convolutional layers with filter size 32 and 16, kernel size 7 in the encoder part and 2 upsampling plus convolutional layers (dimension 32 and input shape, kernel size 21 and 33) in the decoder part, plus ReLU activation function, plus 15 percent dropout in between. The output activation function is linear. For a good initialization the aligner is pretrained to mirror the target data using also Adam optimizer with MAE as loss function. For training 1DCNN-based DBAM the previously on source trained 1DCNN predictor is reused with fixed weights.

Adversarial Training

Here, the training approach that makes use of changing a maximization to a minimization task by reversing the sign of the function to be optimized plus artificial domain labels and linear output function of the discriminator models combined with Wasserstein loss including gradient penalty regularization is used.

For training details including hyperparameter setting see Algorithm 8.1 that is use case specifically adopted from Algorithm 4.1 as presented in [69], for details about loss and algorithm see Section 4. As stated in [69] the source domain is selected under consideration

Algorithm 8.1 DBAM training with gradient penalty for VM.
$\lambda = 10,$ ratio=20, $\beta_1 = 0.5,\ \beta_2 = 0.9,\ lr_{disc} = 0.0001,\ lr_{align} = 1e^{-5}$

```
Require: The gradient penalty coefficient $\lambda$, the $ratio$ between discriminator and aligner
    update, batch size $m = 32$, Adam optimizer [113] hyperparameters $lr$ (learning rate),
    $\beta_1, \beta_2$
Require: pretrained and frozen predictor parameter $\theta_P$, initial discriminator parameter
    $\theta_{D_0}$, pretrained aligner parameter $\theta_{A_0}$
 1: while $\theta_A$ and $\theta_D$ have not converged
 2:     for $t = 1...ratio$
 3:         unfreeze $\theta_D$, freeze $\theta_A$
 4:         sample $S_m = \{x\}_{i=1}^m \sim D_S, T_m = \{z\}_{i=1}^m \sim \{D_T\}^m$
 5:         $L_{D_{total}} = L_{D,D_S}(S_m, \theta_D) - L_{D,D_T}(A(T_m, \theta_A), \theta_D)$
 6:         $\theta_D \leftarrow \text{Adam}(-L_{D_{total}}, lr)$
 7:     end for
 8:     freeze $\theta_D$, unfreeze $\theta_A$
 9:     sample $S_m = \{x\}_{i=1}^m \sim D_S, T_m = \{z\}_{i=1}^m \sim \{D_T\}^m$
10:     $L_{D_{total}} = L_{D,D_S}(S_m, \theta_D) - L_{D,D_T}(A(T_m, \theta_A), \theta_D)$
11:     $L_A = L_{D_{total}} + \lambda L_P(A(T_m, \theta_A), \theta_P)$
12:     $\theta_A \leftarrow \text{Adam}(L_A, lr)$
13: end while=0
```

of quality, stability, availability of data respective labels and best performance for the corresponding regression task. First a baseline prediction model is trained on source via minimization of L_P defined as mean absolute error to fit the parameters θ_P. The weights are frozen after the training hence the high quality accuracy of a dedicated model is preserved. The aligner is then pretrained in form of an autoencoder using target data for best possible initialization. Residual loss is selected for parameter updates θ_A comparing input and recreated signals as output via mean absolute error. After pretraining is finished the DBAM training starts as explained in Algorithm 8.1. The iterative training in form of two player min-max game between discriminator and aligner is done with a ratio of 20.

Results and Discussion: Part 1

In this section, the training and benchmarking done in [69] is presented and a summary of the use case results is given. First the necessity of the adaptation approach is declared by checking distribution based statistics and metrics presented in Section METRICS AND LOSSES: First, the KL-divergence is presented in Table 8.1 between source and target for train as well as test data. Inner-domain distance is significant smaller than outer domain distance for train as well as test set. For similarity comparison of input data respective sensor

KL-divergence of inner vs. outer domain distance

		Source		Target	
Domain		Train	Test	Train	Test
Source	Train	-	23.4	368.0	-
	Test	-22.4	-	-	342.7
Target	Train	367.2	-	22.9	-
	Test	-	347.4	-	-22.0

Table 8.1: KL-divergence of inner (train-test) versus outer (train-train, test - test) domain distance. The results are taken from [69].

measurements taken from source and target domain, univariate KS-test is applied. KS-test is used to confirm the null hypothesis meaning estimate the probability that two sample sets are drawn from the same distribution. The results are very consistent through all 5 folds hence average values over all folds as well as all features are reported. For inner domain feature comparison - train versus test of the same domain - a very high p value with 57 percent for source and 64 percent for target as well as a very low ks score with 0.051 for source and 0.049 for target is presented. For outer domain feature comparison of source and target domain for both train and test data, a very low p value with 9.8 percent for train and 12.8 percent for test and a very high ks score with 0.51 for train and test is registered. Hence the null hypothesis is not rejected for inner domain comparison of train and test data in all 5 cases, but the null hypothesis for comparison of train respective test data of the two different domains is always rejected. For visual comparison based on KL-divergence a T-SNE plot before (a) and after (b) the alignment is created. As stated in [69] and as it can be seen in Figure 8.4, a clear

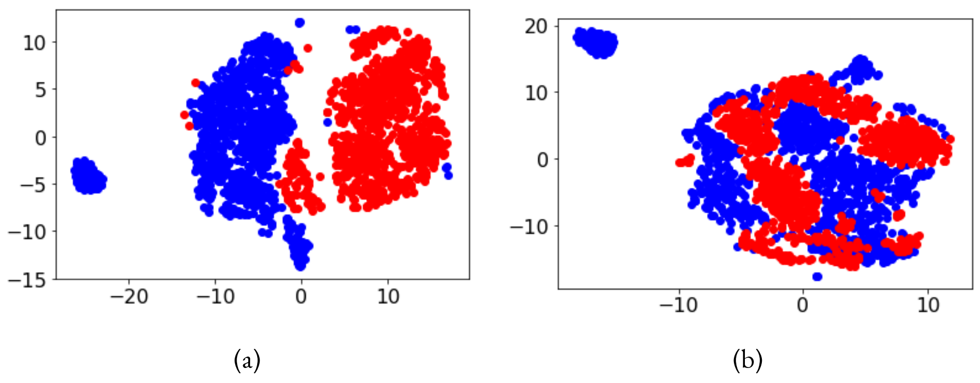


Figure 8.4: T-SNE visualization before and after alignment with DBAM [69]. Graphical t-SNE representation from [69] of source and target domain before (see (a)) and after (see (b)) the alignment via DBAM. The source is colored in blue and contains data from the reference chamber, the target is colored red and contains data from the second chamber. The axes are dimensionless. The effect of the adaptation of the input features after DBAM is applied during training. The adaptation brings the distributions of the two domains closer.

correlation between domain separation and domain overlap is visible before and after the alignment.

Having confirmed a significant difference in input between source and target equipment data, we compare generalization and scaling capacities of VM models presented in literature. Following two learning strategies as presented in [69]:

(i) train and test each model on source domain and directly apply this model on the target domain;

(ii) freeze the prediction models (in case of neural network based architectures), apply the DBAM method and train the aligner with data from target (aligner input) and source (aligner output) and then reuse the already existing model also on the aligned target data.

A third strategy including training a model on both source and target to achieve a generalized model is omitted in favor of showcasing the transfer power of a dedicated model with very high accuracy needed for productive deployment. The focus is to study the behavior of models when context shift occurs and possible solutions for such a scenario.

Table 8.2 taken from [69] shows average results for all models when used on the source test set. Table 8.3 taken from [69] shows average results for all models when used on the target test set. While RF is the most robust model, none of the benchmark model shows acceptable results when applied directly to the target data hence no benchmark is suitable

Source test score average over 5fold cross validation

	X: Feature data					X: Time-series data		
	RR	Lasso	RF	ANN	DBAM	FL	1DCNN	DBAM
MAE	0.06	0.06	0.07	0.06	0.06	0.07	0.07	0.07
ME	0.30	**0.29**	0.34	0.32	0.32	0.33	0.37	0.37
R2	0.88	**0.88**	0.85	0.87	0.87	0.83	0.83	0.83
EV	0.87	**0.88**	0.85	0.87	0.87	0.83	0.84	0.84

Table 8.2: Source test score average over 5fold cross validation. Regression models are trained only on source data, ANN and 1DCNN show the scores where no additional adaptation is applied. DBAM uses the already trained ANN and 1DCNN predictors with fixed weights as well as labeled target training data during training.

Target test score average over 5fold cross validation

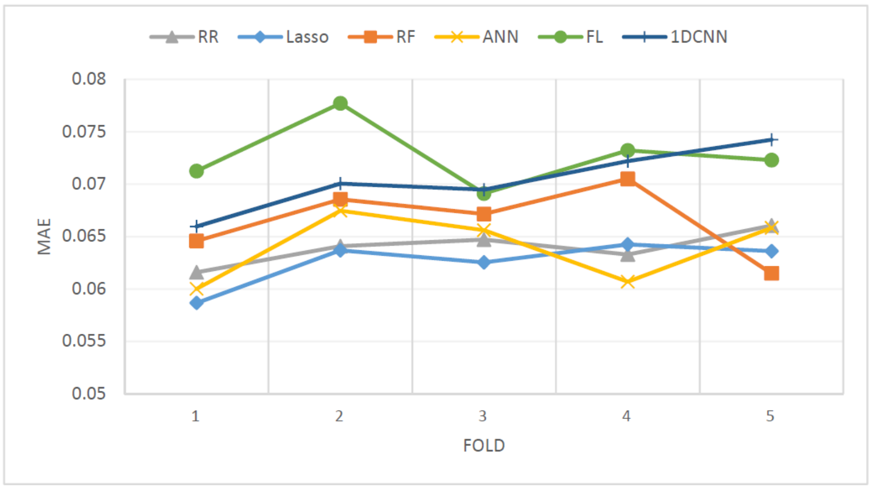
	X: Feature data					X: Time-series data		
	RR	Lasso	RF	ANN	DBAM	FL	1DCNN	DBAM
MAE	1.31	0.48	0.11	0.20	0.10	0.14	0.26	**0.09**
ME	2.01	1.27	0.45	0.58	0.37	0.42	0.58	**0.34**
R2	-0.55	-0.62	0.64	0.25	0.69	**0.70**	0.63	0.67
EV	-48.77	-8.47	0.41	-0.54	0.57	0.24	-1.26	**0.65**

Table 8.3: Target test score average over 5fold cross validation. Regression models are trained only on source data, ANN and 1DCNN show the scores where no additional adaptation is applied. DBAM uses the already trained ANN and 1DCNN predictors with fixed weights as well as labeled target training data during training.

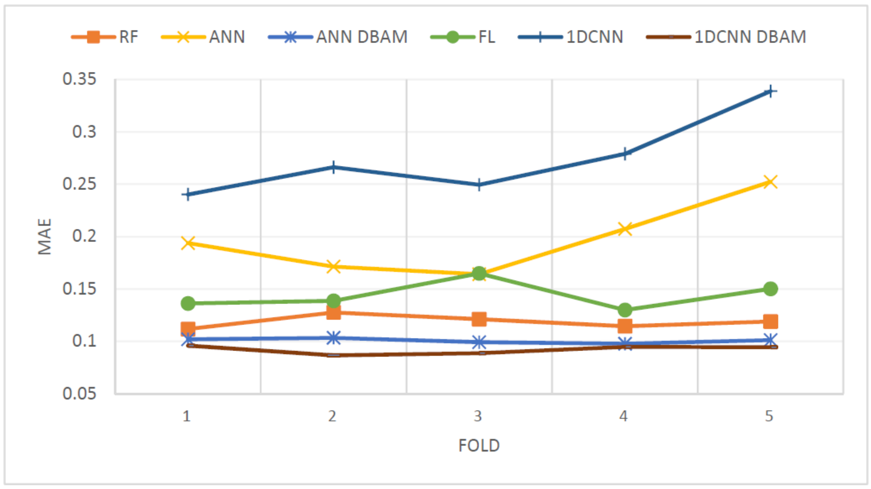
for direct transfer. Overall best results are achieved with 1DCNN DBAM applied to time series data respective sensor measurements. A summary including average prediction errors is presented in Table 8.2 and Table 8.3. Detailed information about evaluation of prediction scores of all 5 cross folds for (a) test source as well as (b) test target are summarized in Figure 8.5.

RF is the most robust VM benchmark model and therefore also presented in Figure 8.6. Nevertheless the plots there show that RF is not able to achieve continuous predictions also visible in the higher ME score presented in table 8.3 when compared to source.

Figure 8.6 from [69] shows true vs predicted values of [a] 1DCNN of source and target domain before alignment, [b] 1DCNN of source and target domain after alignment and [c] source and target domain of best performing benchmark model (RF). The visualization supports the results presented in Table 8.3: clear improvement for [b] compared to [a] and overall good results for RF but with visible discontinuities confirmed by the higher ME score.

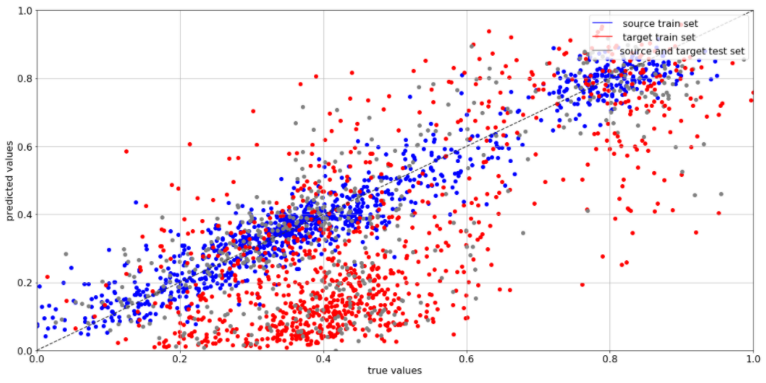


(a)

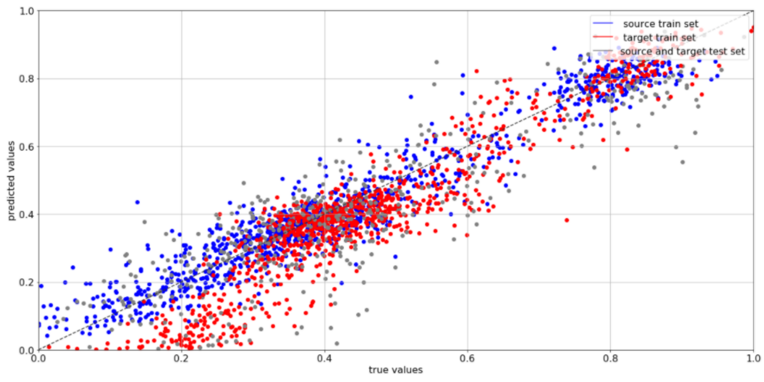


(b)

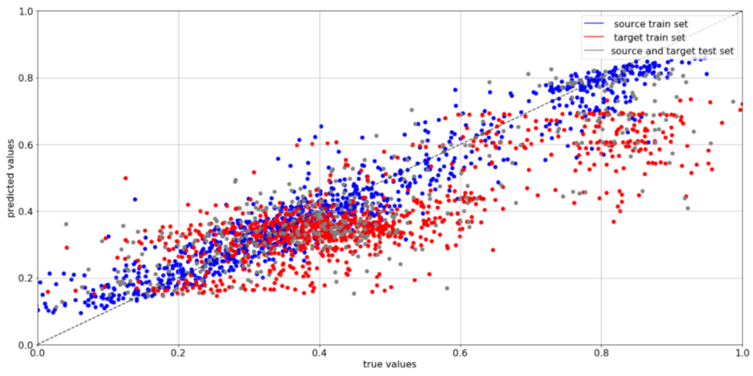
Figure 8.5: 5fold cross validation model error for prediction including DBAM after alignment[69]. 5fold cross validation results taken from [69] to present inner model variance and overall stability of each model. Panel (a) shows test results of the source domain and panel (b) the test results of the target domain (for DBAM after alignment). In panel [b] RR and LASSO are omitted due to their poor performance.



(a)



(b)



(c)

Figure 8.6: True vs. predicted scatter plot for DBAM before and after alignment and best benchmark [69]. Predicted versus true labels from source and target domain of best performing benchmark model (based on MAE, RF, see panel (c)) and 1DCNN DBAM (example plot of one selected fold, see panel (a) - before alignment and panel (b) - after alignment). All prediction plots are taken from [69]. The train source data is colored in blue, the target as well as the aligned target data is colored in red and the test target is colored grey for both. The effect of the adaptation for the predicted values in the target domain is visible comparing (a) and (b). The adaptation brings the distributions of the two domains closer and improves predictions to a high degree. RF shows a comparable low MAE but a high ME.

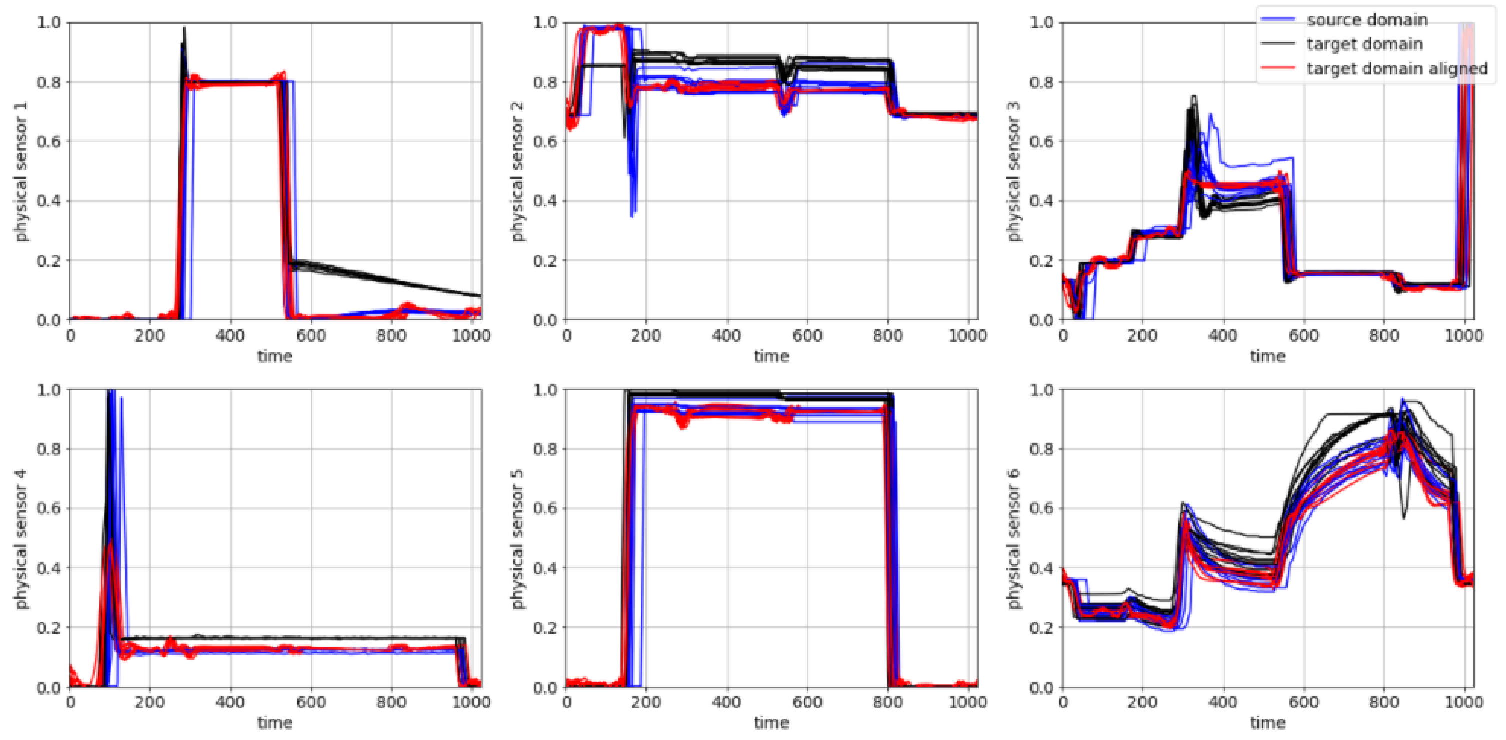


Figure 8.7: Aligner visualization with 6 raw sensor measurements of both source and target before and after the alignment [69]. Graphical representation of 6 raw sensor measurements of both source and target and before and after the alignment with different degrees of univariate misfit like a clear vertical shift for sensor 2 and 6, clear shift but only for a certain step for sensor 1 and 3. The source is again marked blue, the target black and the aligned target red. The graph is taken from [69].

A visualization of the aligner output is presented in Figure 8.7. The aligner set up enables interpretability as well as a comparison between source and aligned target sensor measurements. The presented sensors are taken from [69] and are selected to showcase different kinds of mismatch or shifts that occur naturally in the data.

8.3.2 Experimental Design: Part 2

Part 2 enriches the published work [69] by considering more time series suitable deep learning architectures for the VM use case for the plasma etching process. It applies LSTM, TCN, 1DCNN and combinations of those - all prove to be successful based on published literature - to the different DBAM parts. Therefore, data including preprocessing, the explanations about Metrics and Losses as well as the Adversarial Training are identical to Part 1.

Benchmark models

The benchmarking starts with the best configurations of the first part phase 2, hence with DBAM using 1DCNN architecture. Each part of DBAM is successively changed, trained and evaluate using time series suitable DL architecture.

First the predictor architecture is studied. Two different configurations are examined using TCN architecture. For a description of TCN see Chapter 3. The first configuration uses the canonical architecture and consists only of TCN layers. The second configuration is inspired by Thill et al. [202] and used TCN layers to extract latent features followed by 1DCNN layers for further dimension reduction. The following parameter search space is defined and hyperparameter tuning using grid search is applied to determine the final architecture:

- Activation function: ReLU, Leaky ReLU;
- Number of TCN layers: 1, 2;
- Number of dilated convolution blocks: 1, 2, 4;
- Number of filters: 16, 32, 64, 128;
- Kernel size: 3, 15, 50;
- Dilated convolution depth: 4, 6.

For the LSTM based predictor architecture considers 3 different configurations inspired by Zhang et al. [235] and by Mutegeki and Han [150]. For a description of LSTM see Chapter 3. The first configuration uses purely LSTM layers to create the prediction model. The second configuration uses first stacked LSTM layers to create time sequences of features, followed by 1DCNN layers for dimension reduction. The final two layers are fully connected dense layers. The third configuration uses first 1DCNN layers for feature extraction and dimension reduction, model lightening and simplification, followed by LSTM layers to extract time dependencies. The hyperparameter search space is defined as:

- Number of LSTM layers: 1, 2, 3;
- Number of units per layer: 25, 50, 100;
- Dropout rate: 0, 0.1, 0.25.

Again, hyperparameter tuning using grid search is applied to determine the final architecture.

Next, domain adaptation using DBAM is checked under consideration of different DL time series suitable architectures. The starting point is the 1DCNN DBAM architecture discussed in the first phase part 2. The following step-by-step approach is applied:

1. The predictor architecture is changed by testing the above selected TCN and LSTM predictor models. The adversarial elements are kept unchanged. The performance is analyzed and the best performing predictor model is used for further architecture evaluation with changed aligner and discriminator architectures;
2. the aligner architecture is changed and tested using LSTM and TCN architecture. For better initialization a warm-up training of the aligner is done identical to the pretraining described for the 1DCNN benchmark and as recommended in [69].
3. Based on the previous performance result, alternative architectures using TCN and LSTM are discussed for the discriminator model within DBAM.

The following aligner architecture are studied: The following parameter search space is defined for TCN aligner again based on [202] and hyperparameter tuning using grid search is applied to determine the final architecture:

- Activation function: ReLU;

- Number of TCN layers: 1, 2;
- Number of dilated convolution blocks: 1, 2, 4;
- Number of filters: 8, 16, 32;
- Kernel size: 15, 50;
- Dilated convolution depth: 4, 6.
- Type of compression: Average pooling, 1DCNN pooling.

A series of TCN layers are used to extract features that subsequently are compressed, decompressed and finally passed through another series of TCN layers to reconstruct the signals. The first stack of TCN layers uses causal convolution while the second stack uses valid padding. After upsampling with an upsampling layer of 16, TCN layers for the reconstruction uses a dilated convolution depth that is inverted.

The LSTM aligner follows the recommendations by Sagheer and Kotb [171] and Li et al. [127]. LSTM layers are stacked together to produce the features. They are compressed, decompressed and finally used to reconstruct the original signals with a new series of LSTM layers. The following parameter search space is tested for the LSTM aligner architecture:

1. Number of LSTM layers: 1, 2;
2. Number of units: 25, 50;
3. Dropout rate: 0, 0.1, 0.25;
4. Type of compression: Average pooling, 1DCNN pooling.

As for the TCN aligner, the last parameter owns 16 as the compression factor. Instead of an upsampling layer, features are repeated N times to restore the original dimension.

Based on the results of the aligner study, TCN discriminator is skipped and only one LSTM based discriminator is tested under consideration of training time, dimensionality, computational complexity and literature recommendations. Hence, the LSTM discriminator uses both, LSTM as well as 1DCNN layers.

Further test are done for the adversarial training approach, hence on the training ratio and the adversarial loss weights:

- Adversarial loss weight (α): chosen from the set $\{0.1, 0.25, 0.5, 1, 5\}$;
- Discriminator training extra steps: chosen from the set $\{5, 10, 20\}$.

Models and Hyperparameters

Hyperparameter tuning is done on a validation set. 5 fold cross validation is finally applied using the same data split as before. The following architectures are used for prediction:

- The TCN predictor model consists of one TCN layer with 1 dilated convolution block, 64 filters, kernel size 50 and dilated convolution depth 6; one 1DCNN layer with kernel size and stride of 5 reducing by 5 the time dimension; a fully connected layer with 16 units applied to the flattened version using a flattening layer of the features extracted previously; finally an output layer with 1 fully connected unit, the 1DCNN layer and the fully connected layer have LeakyReLU activation function, the final output activation is linear. The used hyperparameter are marked in the section above. Other hyperparameters are selected based on recommendations by Cao et al. [24].
- The LSTM predictor model has one 1DCNN layer with kernel size and stride of 5 reducing by 5 the time dimension; one LSTM layer with 50 units per layer and 0 dropout rate; one fully connected layer with 16 units applied after a flattening layer and an output layer with 1 fully connected unit. The 1DCNN layer and the fully connected layer have LeakyReLU activation function, the dense output layer has a linear output activation function. Other hyperparameter are taken from [Reimers and Gurevych]. The selected hyperparameter are marked in the section above.

All DBAM models are trained using Algorithm 4.1 presented in Chapter 4. The predictor models are reused and trained weights are frozen. For the adversarial training using DBAM the following training parameters are kept fixed:

- **Available data**: same data used with 32 as batch size;
- **Training epochs and early stopping criterion**: fixed at 300 without early stopping;
- **Optimizer**: Adam;
- **Learning rates and their schedule**: fixed for the aligner at 0.00001 and the discriminator at 0.0001, without decreasing during training;

- **gradient penalty regularization by [78]**: with a loss weight fixed to 10.

For the aligner and discriminator DBAM parts the following architectures are used:

- The TCN aligner structure takes inspiration from [202]. None of the tested hyperparameter settings give acceptable performance results since too much noise is introduced and reconstructed signals are not identifiable.
- The LSTM aligner consists of one LSTM layer with 50 units and no dropout; one average pooling layer with 16 as compression factor; one repeat layer of 16 to reconstruct the length of the signal; finally one LSTM layer with 33 units and no dropout to reconstruct the signal.
- The LSTM discriminator has one 1DCNN layer with 50 filters, kernel size 5 and stride 5; one 1DCNN layer with 50 filters, kernel size 3 and stride 3; one LSTM layer with 16 units and no dropout; a fully connected layer with 512 units applied to the flattened version of the features using a flattening layer; a series of fully connected layer with 256, 128, 64, 32 units respectively; one output layer with 1 fully connected unit with linear activation function. 1DCNN and the fully connected layers use Leaky ReLU as activation function.

Results and Discussion: Part 2

First the evaluation of the predictor models is presented. The hyperparameter tuning is done on a first division of the data using a validation set. The presented results are averages over all 5 folds. The results for the TCN based predictor are given in Table 8.4. The results for

Evaluation of TCN predictor for VM

	MAE	ME	EV	R2
Source Validation	0.0531	0.185	-	-
Source Test	0.0560	0.164	0.895	0.893
Target Test	0.129	0.357	0.720	0.377

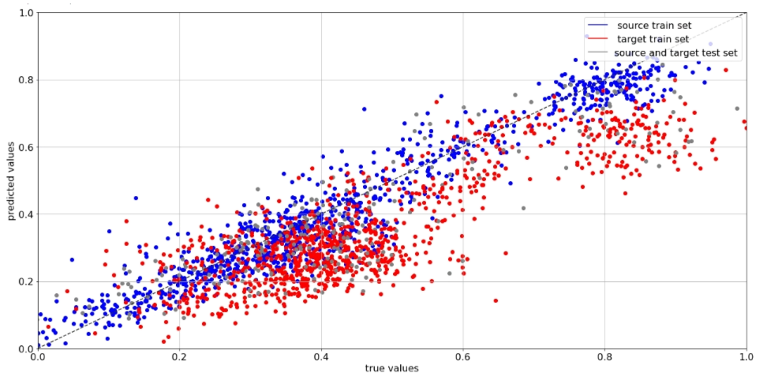
Table 8.4: TCN predictor: VM performance evaluation. Source and target test score average over 5fold cross validation. Regression models are trained only on source data.

the LSTM based predictor are given in Table 8.5.

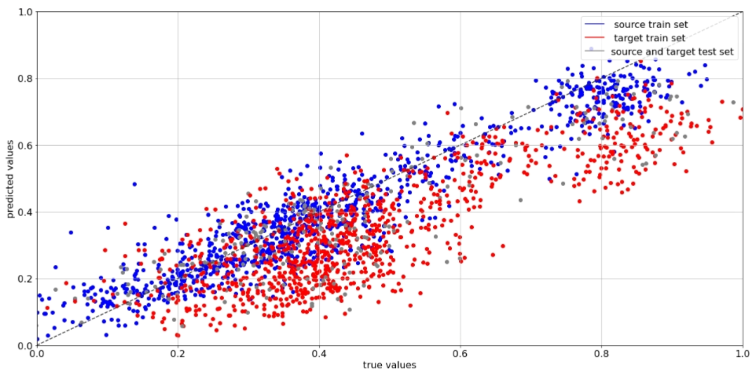
Evaluation of LSTM predictor for VM

	MAE	ME	EV	R2
Source Validation	0.0535	0.177	-	-
Source Test	0.0624	0.165	0.900	0.
Target Test	0.141	0.338	0.689	0.302

Table 8.5: LSTM predictor: VM performance evaluation. Source and target test score average over 5fold cross validation. Regression models are trained only on source data.



(a)



(b)

Figure 8.8: True vs. predicted scatter plot for TCN and LSTM predictor. Predicted versus true labels from source and target domain of TCN and LSTM predictor. The train source data is colored in blue, the target is colored in red and the test target is colored grey for both. A better generalization capability is visible for both.

The results of both TCN and LSTM predictor show improved results in regards to the source accuracy but also to the generalization capabilities towards the target accuracy. A scatter plot of predicted versus true labels from source and target domain of TCN and LSTM predictor are shown in Figure 8.8. The higher complexity of the models leads to better results on the dedicated task as well as the transfer task under a stable training. Thanks to the 1dCNN plus LSTM architecture set up, the LSTM predictor model size is kept small and problems with the limited receptive field are avoided. The same goes for the TCN model.

Next, the DBAM results are presented and the architectural changes are progressively analyzed. The first check includes changing the predictor model architectures while keeping the aligner and discriminator part unchanged. First, the TCN predictor model is used and DBAM is trained using $\alpha = 0.1$ after parameter optimization and a training ratio of 5. Then the LSTM predictor model is used and DBAM training is done using $\alpha = 0.25$ and same training ratio. The results for both TCN and LSTM predictor for source and target test data after DBAM alignment is presented in Table 8.6. Both, TCN as well as LSTM predictors

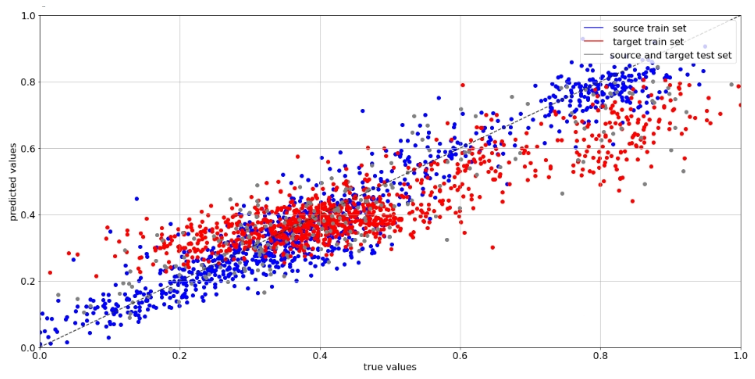
DBAM Alignment with TCN and LSTM predictor for VM

	MAE	ME	EV	R2
TCN predictor	0.0881	0.387	0.687	0.646
LSTM predictor	**0.0822**	**0.320**	**0.713**	**0.707**

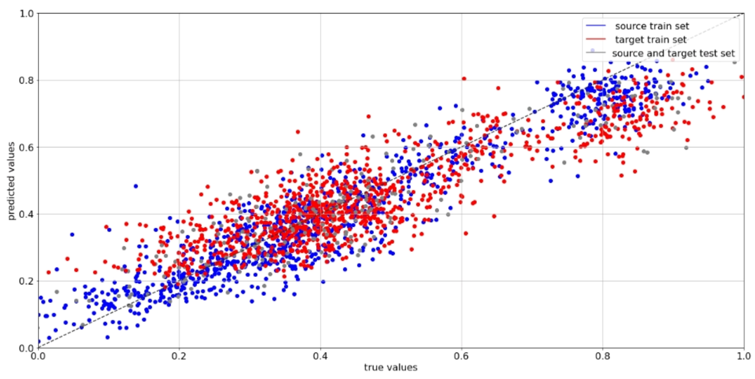
Table 8.6: DBAM Alignment with TCN and LSTM predictor: VM performance evaluation. Target test score average over 5fold cross validation. Prediction/regression models are trained only on source data. DBAM is used to align the target data and the performance of the dedicated model is checked on the mapped target data.

also improve the alignment performance of DBAM. A scatter plot of predicted versus true labels from source and target domain of TCN and LSTM predictor after alignment with DBAM are shown in Figure 8.9. The LSTM predictor shows best performance compared to 1DCNN and TCN for both the regression task itself as well as the DBAM alignment. For visual comparison based on KL-divergence, a T-SNE plot before (a) and after (b) the alignment is created. It can be seen in Figure 8.10, a clear correlation between domain separation and domain overlap is visible before and after the alignment.

The second check includes changing the aligner model architectures while selecting the corresponding predictor model and keeping the discriminator part unchanged. The TCN aligner (and all tested variations in the parameter search space) introduces noise to the reconstructed signals, hence make it not suitable for the DBAM model that is no longer able to rely on its functionality to recreate therefore visualize and interpret the aligned signals. Examples of the noisy signals are presented in Figure 8.11. It is obvious that none of the possible



(a)



(b)

Figure 8.9: True vs. predicted scatter plot for TCN and LSTM predictor in DBAM. Predicted versus true labels from source and target domain of (a) TCN and (b) LSTM predictor after alignment with DBAM. The train source data is colored in blue, the aligned target is colored in red and the test target is colored grey for both. A significant alignment and improved prediction accuracy is visible for the aligned target domain.

TCN aligners can improve previously achieved alignment results, hence further evaluation is skipped.

The LSTM aligner is trained using $\alpha = 0.25$ and a training ratio of 20. The performance results are given in Table 8.7 (compared to the LSTM predictor plus 1DCNN aligner). The results are comparable but slightly lower in accuracy. A visualization of the aligned signals is given in Figure 8.12. The LSTM aligned signals are more regular and smooth that leads to an overall good fit except for the peaks where the recreated signals are smoothed compared to the original ones. The LSTM aligner weights are kept frozen and due to comparable prediction accuracy using LSTM aligner, no significant changes in the prediction plots are visible in Figure 8.13.

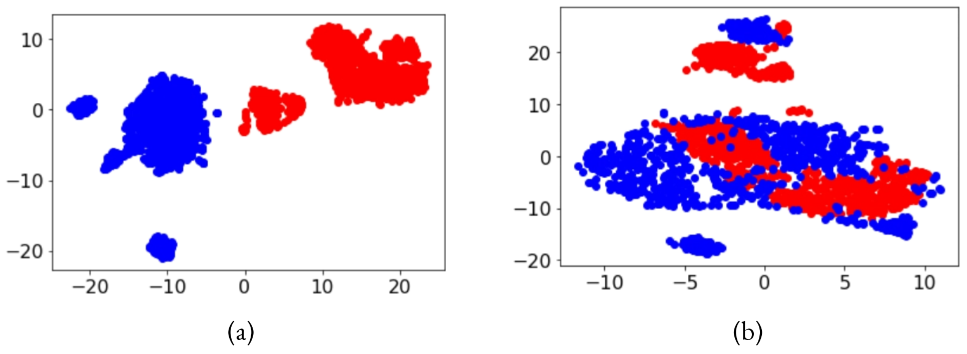


Figure 8.10: T-SNE visualization before and after alignment with DBAM using LSTM predictor Graphical t-SNE representation of source and target domain before (see (a)) and after (see (b)) the alignment via DBAM using a LSTM predictor model. The source is colored in blue and contains data from the reference chamber, the target is colored red and contains data from the second chamber. The axes are dimensionless. The effect of the adaptation of the input features after DBAM is applied during training. The adaptation brings the distributions of the two domains closer.

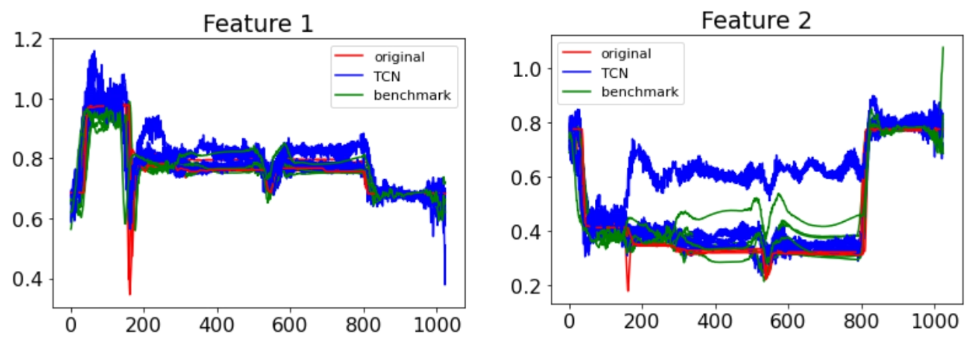


Figure 8.11: TCN aligner visualization of raw sensor measurements of both source and target after the alignment. Graphical representation of 2 raw sensor measurements of both source and target after the alignment using TCN predictor and TCN aligner model. The source is marked red, the aligned target blue and the aligned target via benchmark model is marked green.

DBAM alignment with LSTM predictor and LSTM aligner

	MAE	ME	EV	R2
LSTM predictor	**0.0822**	**0.320**	**0.713**	**0.707**
LSTM aligner	0.0856	0.342	0.693	0.691

Table 8.7: DBAM Alignment with LSTM predictor and LSTM aligner: VM performance evaluation. Target test score average over 5fold cross validation. Prediction/regression LSTM based model is trained only on source data. DBAM using LSTM predictor and LSTM aligner is used to align the target data and the performance of the dedicated model is checked on the mapped target data.

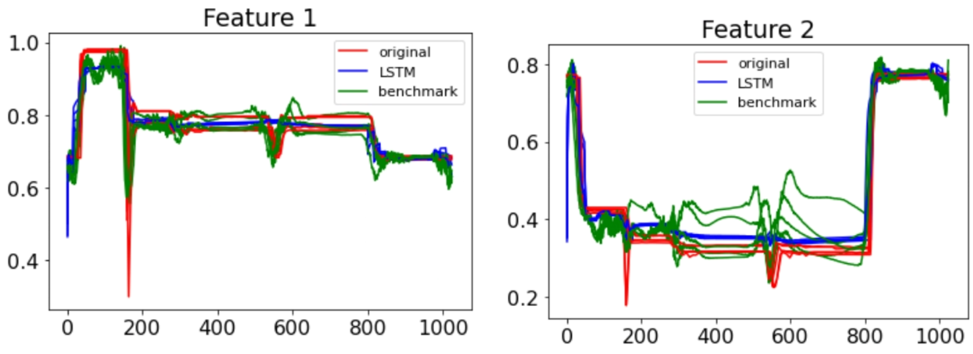


Figure 8.12: LSTM aligner visualization of raw sensor measurements of both source and target after the alignment. Graphical representation of 2 raw sensor measurements of both source and target after the alignment using LSTM predictor and LSTM aligner model. The source is marked red, the aligned target blue and the aligned target via benchmark model is marked green.

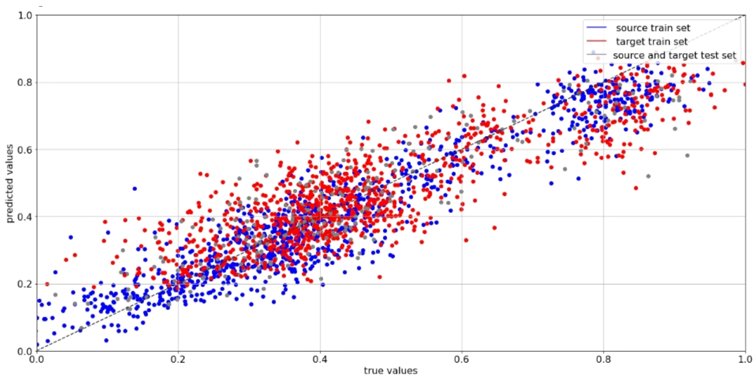


Figure 8.13: True vs. predicted scatter plot for LSTM predictor and LSTM aligner in DBAM. Predicted versus true labels from source and target domain of LSTM predictor after alignment with DBAM including LSTM alinger model. The train source data is colored in blue, the aligned target is colored in red and the test target is colored grey for both. A significant alignment and good prediction accuracy is visible for the aligned target domain.

Last check is the LSTM discriminator in the DBAM model. Again $\alpha = 0.25$ and a training ratio of 5 is chosen. The performance results are given in Table 8.8. The 1DCNN benchmark is improved again, but DBAM using LSTM predictor plus 1DCNN aligner and discriminator performs the best.

Overall, changing the predictor leads to the biggest improvement related to generalization of the predictor model itself as well as the transfer capabilities of DBAM. Since some architecture set ups have a larger dimensionality and computational complexity, it can be rec-

DBAM alignment with LSTM based DBAM

	MAE	ME	EV	R2
LSTM predictor	**0.0822**	**0.320**	**0.713**	**0.707**
LSTM aligner	0.0856	0.342	0.693	0.691
LSTM discriminator	0.0917	0.324	0.676	0.663

Table 8.8: DBAM alignment with LSTM predictor, LSTM aligner and LSTM discriminator: VM performance evaluation. Target test score average over 5fold cross validation. Prediction/regression LSTM based model is trained only on source data. DBAM using LSTM predictor, LSTM aligner and LSTM discriminator is used to align the target data and the performance of the dedicated model is checked on the mapped target data.

ommended to use the most simple architecture when facing small number of data samples (if comparable performance results can be achieved). Nevertheless, the analysis presented here showcase the power of LSTM for time series based regression tasks and confirms the promising results presented in literature.

8.4 Virtual Metrology (VM) for Equipment with Heterogeneous Data Representation

One of the biggest advantages of DBAM and its extended version DBACS is that it is applicable to data sets with different input representations. Research on multi-view learning by Xu et al. [222] deals with a very similar problem by comparing and analysing multiple representations called views of a problem and its collected input data. This is rarely discussed in semiconductor related literature so far but can happen in production when an already established process is transferred to another nonidentical equipment due to for example availability or new equipment generations. Here, data from an etching process is presented that runs on two different equipment types. Corresponding examples of both equipment types are shown in Figure 8.1. The process was first set up on on the equipment presented in (a) and then transferred to (b) by a domain expert in order to have identical processes including identical process results. Hence it can be assumed that the underlying physical process and therewith the physical information in the data is identical.

Data Preparation

The data comes from two different equipment types from the same vendor. The two equipment types are presented in Figure 8.1. The data set is restricted to a specific etching recipe that runs on both equipment. Metrology measurements are taken after the process independently of the process equipment. A time period of tree years is considered. The equipment type 1 in Figure 8.1 a) is selected as source since it consists most of the data, around 10 000 samples, the newer equipment type presented in Figure 8.1 b) is defined as the target having almost 6000 data samples. Only raw sensor measurements in form of time series data that is measured with preinstalled physical sensors on each equipment is considered and preprocessed with the following steps:

- removal of constant features;
- removal of features that show small fluctuation that can be detect as noise (variations smaller than 0.01) and a constant behavior underneath the noise;
- removal of samples showing label outliers based on IQR;
- removal of samples where the length of the time series lies below or above 25 percent respective 75 percent quantile of time series length;
- equal-distributed upsampling of timestamps and feature values to generate time series with equal length.

Since the available sensors are different - considering number of available sensors as well as taken measurement - the two equipment are preprocessed separately. 32 features for equipment type 1 respective 49 for equipment type 2 are finally selected and used as input for the VM modeling task. No significant label shift is detected within metrology measurements from different equipment types as presented in Figure 8.14.

Experimental Design

In the first part of the experiment we show the applicability of the DBACS model to a VM use case involving the two different equipment types for source and target hence heterogeneous data representations - different number of features as well as feature content.

In the second part of the experiment we test covariance based - meaning linear respective kernel based - approaches in order to deal with heterogeneous input data inspired by [81], [64] and [3]. The idea is to map both source and target via individual projections into a common subspace in order to create feature representations that enable a common VM prediction model. The following multi-step modeling approach is applied:

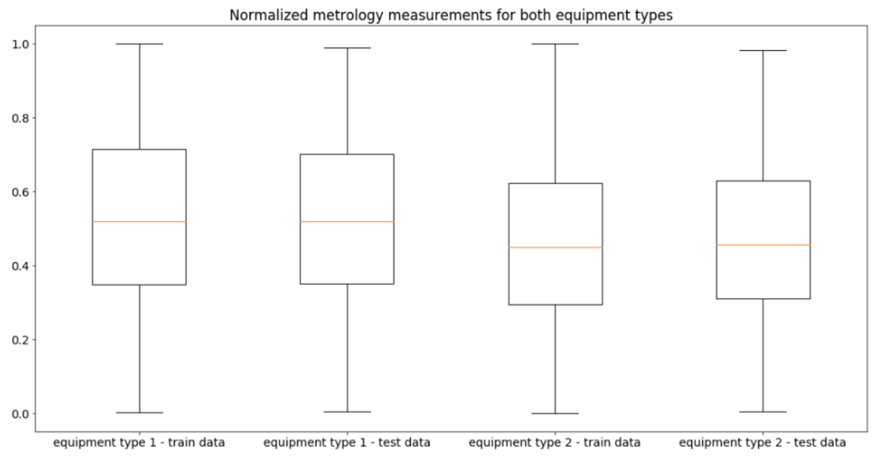


Figure 8.14: Boxplot graphs of normalized metrology/inline measurements from both equipment types considered in the analysis.

1. generate common (latent) feature representation;
2. compare feature as well as domain similarity and inner versus outer domain difference after the feature mapping;
3. optional refinement by applying additional domain adaptation methods on latent feature space;
4. train prediction model on aligned features.

In general, identical distributions of the latent source and target feature spaces cannot be directly expected even if the applied method explicitly promotes an in-between relationship of the representations. The optional application of additional domain adaptation methods means to first test the similarity of the data distributions and depending on the results either directly train one common model - if no significant difference can be detected anymore - or discuss application of known transfer learning and domain adaptation approaches to further improve the alignment before training the final prediction model.

Benchmark Models

DBACS architecture choice is made based on the findings in Section 8.3 that compares different prediction methods for the homogeneous VM task. Hence, 1DCNN as simple but good performing architecture is selected and used for all involved model parts: the prediction model trained on time series data from equipment type 1 selected as source domain, the aligner parts that are trained with data from both equipment types with the goal to create a bijective mapping between the two domains and discriminator parts trained as domain classifiers. For more details on the 1DCNN parts we refer to Chapter 3.1.

For the linear respective kernel based methods to treat heterogeneous data distributions with the same underlying physical information, the multi-step approach described above is followed and the following methods are compared against each other:

(A.) find best matching feature pairs by correlating analysis using Pearson's r;

(B.) create common (latent) input features space by applying Principle Component Analysis (PCA) to both domains and select same number of principle components;

(C.) create common (latent) input features space by applying Canonical Correlation Analysis (CCA) respective Kernel Canonical Correlation Analysis (KCCA) to both domains to generate highly correlated respective related feature representation.

For an introduction of applied benchmark models we refer to Chapter 6.
First (A.) correlation analysis is applied to match sensors from two equipment types and select best matching pairs. The distribution similarity of matched features is tested. This method is best suited if similarity between equipment types hence installed sensors and sensor readings are expected e.g. same vendor.

If no identical sensors are expected, latent features are needed to create identical input spaces that hold task relevant information while being equipment type hence domain invariant. Secondly, (B.) PCA as linear approach is used and same number of latent features is selected for both, covering high amount of variance hence relevant contribution for both equipment types. PCA does not need labels hence is unsupervised. Soure and target domains are treated completely separately and cross domain correlation or covariance are not considered.

Inspired by research in the field of semantic representations respective kernel representations [81], (C.) Canonical Correlation Analysis (CCA) is applied. Its purpose is to analyze the relationship between two sets of variables considering outer domain correlations while PCA focus on relationship of variables within on set.

Selecting suitable domain adaptation approaches as benchmarks besides the DL based ones is limited since the approach either needs to be a) suited for time series data or b) the method needs to be unsupervised if there is the need to reshape time series data by concatenating and treat subsequent timestamps as separate samples. Hence for the optional refinement by applying additional domain adaptation methods and for simplicity, the following approach known from domain adaptation related publications is applied:

- Correlation Alignment (CORAL)

For an introduction and further details see Chapter 6. For PCA as well as CCA, a test for distribution similarity is conducted as well as relationship and distance of latent feature respective inner and outer domain distance are tested. Inspired by subspace alignment SA [64], in a second step we combine PCA as well as CCA with CORAL and repeat evaluation.

Metrics and Losses

To measure the success of the alignment besides overall performance of different models, two groups of losses are reused as already introduced for the first VM use case in Section 8.3.

For **Distribution-based loss** we apply both (1) T-distributed Stochastic Neighbor Embedding **(t-SNE)** - a machine learning algorithm developed for data visualization and especially suitable if the data contains nonlinear dependencies [206] - as well as (2) Kullback-Leibler **(KL)** divergence - used in t-SNE that is based on gradient-descent minimization of the **(KL)** divergence. For the purpose of comparing inner domain distance - the distance between for example train and test set of one domain - and inter respective outer domain distance - the divergence between data selected from source and data selected from target domain - we use Frechet inception distance (FID). The Kolmogorov–Smirnov **(KS)** test is used for a univariate latent feature comparison and compute the average over all projected univariate test statistics. Here, we test a special kind of outer domain equality - meaning train sample set vs test sample set of both latent feature spaces meaning mapped train respective test sample set from source versus target.

For **Performance-based loss** and overall performance comparison, we apply a second time mean absolute error **(MAE)**, maximal residual error **(ME)**, coefficient of determination also called regression score function **(R-square/R2)** and explained variance regression score **(EV)** in case the mean error is not equal to 0. For a more detailed definition see Section 7.

For heterogeneous domain adaptation we additionally apply **Structural Similarity Index (SSIM)**. In order to apply SSIM, originally developed for image data, to time series data, its dimension are expanded. Hence the last axis of the time series is multiplied and filled with copies of the corresponding single value. The SSIM can be directly applied to the expanded time series data.

Models and Hyperparameter

5 fold cross validation is applied, hence both data sets are split into 5 subsets each and using 4 merged sets as train and 1 as test set per fold. Architectures of all models stay fixed for all 5 folds. For selected benchmark models we use available python implementations. For the correlation analysis we use the function implementation available in python module *numpy* [82] and for PCA and CCA we use existing function implementation in the python module *scikit-learn* [160]. More versions of CCA meaning KCCA and DCCA are implemented in the python module *mvlearn* [161]. For CORAL we use the implementations from the python module *transfertools* [?]. For further explanations concerning neural network architecture we refer to [74]. DBACS models are trained using the extended version of Algorithm 4.1 where for the Min-Max two player game all available aligner are updated together as well as all available discriminators. For the neural network based models (including the prediction model used in the benchmarking approaches) we use the following architectures:

- The 1DCNN predictor model consists of three convolutional layers with dimension 32, 16 and 8, kernel size 53,33 and 33, Leaky ReLU activation function and Batch Normalization after each 1DCNN layer. It is followed by one max pooling layer and a flattening layer for reducing overall output dimension from 3 to 2. The last two layers after the flattening has dimension 16 and 1 and the output activation function is sigmoid. The 1DCNN is trained under usage of preprocessed sensor data from equipment type 1 by minimization of MAE using Adam optimizer and learning rate $r = 0.00005$.

- The domain discriminators both have the same architecture besides the respective input shape: three convolutional layers of dimension 24, 16 and 8, kernel size 17, causal padding and leaky ReLU activation function, max pooling of size 4 and 2 times 2. Then we have one flattening layer and six dense layers size 512, 256, 128, 64, 32, 1 with Leaky ReLU activation and linear output function. Both aligner consist of 6 convolutional layers, first 5 followed by Leaky ReLU activation function, the final output is kept linear.

- The aligner that maps target domain to source domain has filter size 48, 42, 36, 32, 32 and final filter size is set to number of features of the source domain. Kernel size is 37 four times, then 57 and 7. Upsampling with size 3 and 2 is done after 4th and 5th layer block. The aligner that maps source domain to target domain has filter size 32, 36, 42, 46, 48 and final filter size is set to number of features of the target domain. Kernel size is 37 four times, then 57 and 7. Upsampling with size 3 and 2 is done after 4th and 5th layer block. For an improved initialization both aligners are pretrained: therefore we select sample pairs from source and target based on closest label value. Hence we map samples based on their label distance. Adam optimizer with SSIM as loss function is used for pretraining. For training 1DCNN-based DBACS the previously on source trained 1DCNN predictor is reused with fixed weights.

Results and Discussion

First, the necessity of the adaptation approach from method as well as expert perspective is recaptured. Therefore, it is stressed again that a direct comparison of features or distribution before the alignment is not plausible due to different data representations. The different number of available features do neither allow a direct transfer of a dedicated model nor the usage of a commonly trained one. Having complete separate models would not only lead to increased maintenance effort but also to limited possibilities to compare results and transfer troubleshooting, correction of defects and hence process improvements. A comparison is only possible on process result level but not on process monitoring level.

For the first part, the results for the heterogeneous DA done with DBACS are shown and discussed. For visual comparison of the two domains based on KL-divergence we present T-

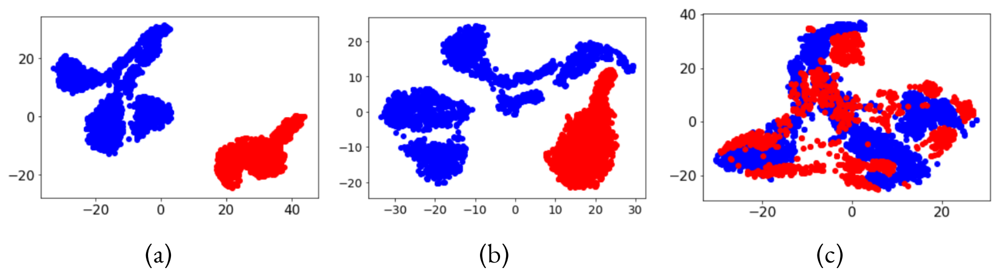


Figure 8.15: T-SNE visualization before and after alignment with DBACS. Graphical t-SNE representation of source and target domain in different stages of the alignment process: (a) shows features mapped by a randomly initialized aligner, (b) after the pretraining of the aligner and (c) after DA with DBACS is done. The source is colored in blue and contains data from equipment type 1, the target is colored red and contains data from the equipment type 2. The axes are dimensionless. The effect of the adaptation of the input features after DBACS is applied during training. The adaptation brings the distributions of the target domain closer and finally target overlaps source domain.

SNE plots in Figure 8.15. All plots are done using the aligner to map all features to the source feature space, but in different stages of the alignment process: (a) shows features mapped by a randomly initialized aligner, (b) after the pretraining of the aligner and (c) after the DA with DBACS is done. An increasing correlation between domain separation and domain overlap is visible.

First, lower and upper error bound are discussed and later (if possible) compared to the error achieved by DBACS. The lower bound represents the error when using dedicated models individually trained and individually used for each equipment type. It is the best model performance that can be expected hence the goal of a transfer learning approach is to come as close as possible to this lower bound. Nevertheless, dedicated models have some disadvantages: 1. higher effort in maintaining multiple models including evaluation, analyzing results, defining monitoring kpis, updating the models; 2. only applicable when enough data and especially labels are available; 3. no direct comparison and knowledge transfer between equipment types and generated prediction models is possible. The upper bound is the direct transfer of a trained model to unseen data coming from different equipment not used for training. For heterogeneous data representation an upper bound cannot be directly computed due to fixed model input shape. In the second part of this section alternative methods to deal with heterogeneous input spaces are evaluated. Hence for now only lower bounds are presented in Table 8.9.

Lower bounds for DA using dedicated ML algorithms

	Source lower bound		Target Lower bound	
	Train	Test	Train	Test
MAE	0.08	0.09	0.10	0.13
ME	0.60	0.53	0.63	0.62
R2	0.82	0.75	0.66	0.50
EV	0.66	0.69	0.64	0.47

Table 8.9: Evaluation of dedicated ML algorithms for VM predictions. Models are trained only on source respective target training data and afterwards evaluated with corresponding source respective target test data. All scores are mean values for 5 fold CV. Based on the different input space dimensions and data representations, dedicated models cannot be directly transferred.

Next we state generalization and scaling results of DBACS for VM prediction model used on heterogeneous domains. Following the learning strategy presented in Section 8.3: 1. freeze the prediction model trained on source data, 2. pretrain the aligners on samples paired based on closest labels, 3. apply the DBACS method and extended version of Algorithmus 8.1 to train the aligners and discriminators. Finally the already existing prediction model is reused for the aligned target data (target data gets mapped to source domain using aligner F). Table 8.10 shows the average 5 fold CV results for DBACS.

DBACS performance for source and aligned target

	Source domain		Target domain	
	Train	Test	Train	Test
MAE	0.08	0.09	0.10	0.13
ME	0.60	0.53	0.59	0.61
R2	0.82	0.75	0.74	0.55
EV	0.66	0.69	0.63	0.47

Table 8.10: Source and aligned target data training and test scores average over 5 fold CV. VM prediction models are trained only on source data and evaluated on test data. Target data is mapped to source domain using trained aligner F from DBACS and evaluated after the mapping using the VM prediction model trained on source.

The numbers given in Table 8.10 confirm the visual convergence seen in the t-SNE plot in Figure 8.15. The target data is successfully mapped to the source space and the dedicated source prediction model shows comparable errors for aligned target data. Detailed information about mean absolute error (MAE) of VM prediction model for all 5 fold CV for prediction of train and test source data as well as prediction of aligned train and test target data are summarized in Figure 8.16. DBACS shows overall a very good stability and reproducibility. Next, Figure 8.17 shows true vs predicted values of (a) target data mapped to source space by a randomly initialized aligner, (b) target data mapped to source space by the aligner after the pretraining and (c) target data mapped to source space after the DA with DBACS is done. Again, the visualization supports the results presented in Table 8.10: Enabling using of dedicated source model to mapped target data with clear improvement for (c) compared to (a) and (b). A visualization of both aligners output is presented in Figure 8.18. The cyclic aligner set up enables a bijective mapping between both domains hence interpretability as well as a comparison between source and aligned target sensor measurements as well as target and aligned source sensor measurements.

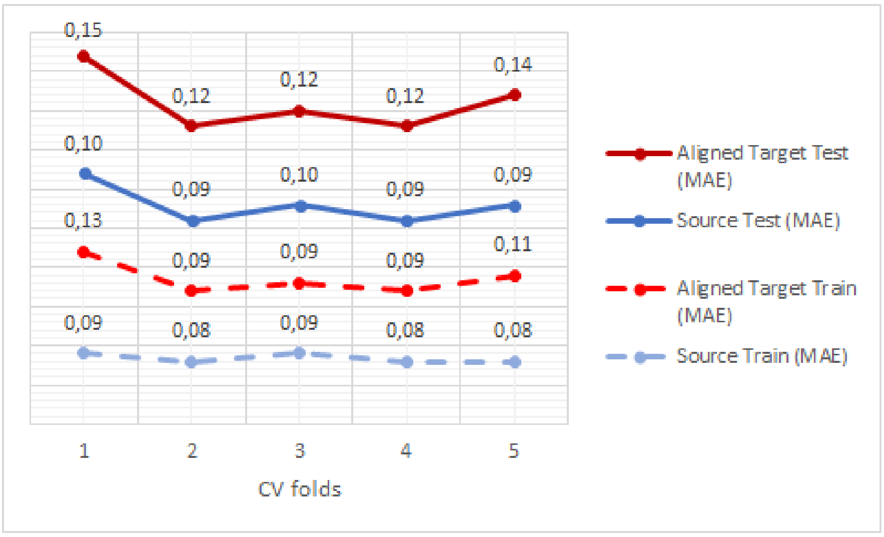
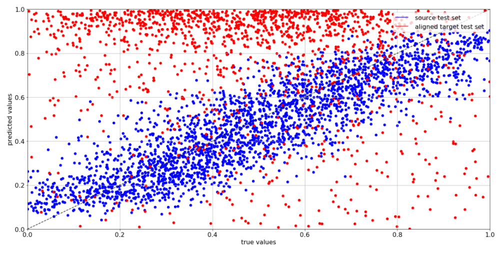
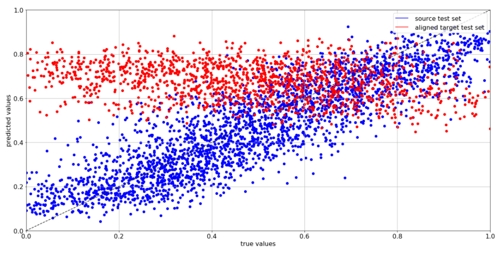


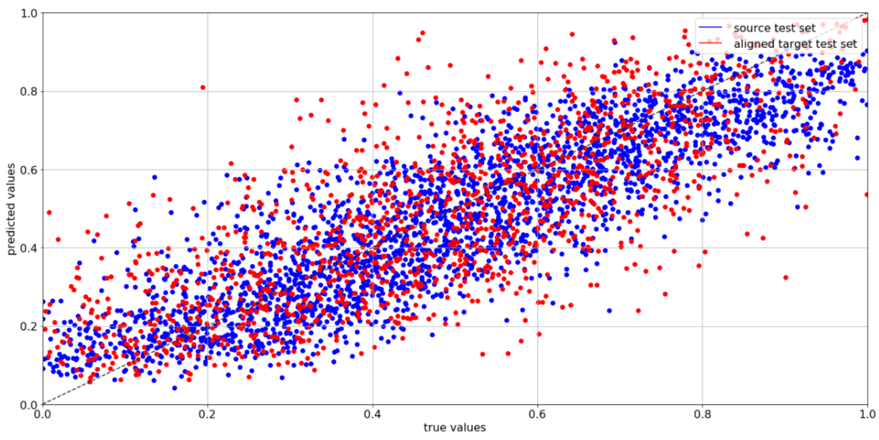
Figure 8.16: 5-fold cross validation model error for prediction including DBACS after alignment. 5fold cross validation mean absolute error (MAE) to present inner model variance and overall stability of DBACS model for DA. Panel shows VM prediction train and test results of the source domain and the VM prediction train and test results of the target domain after alignment with DBACS.



(a)

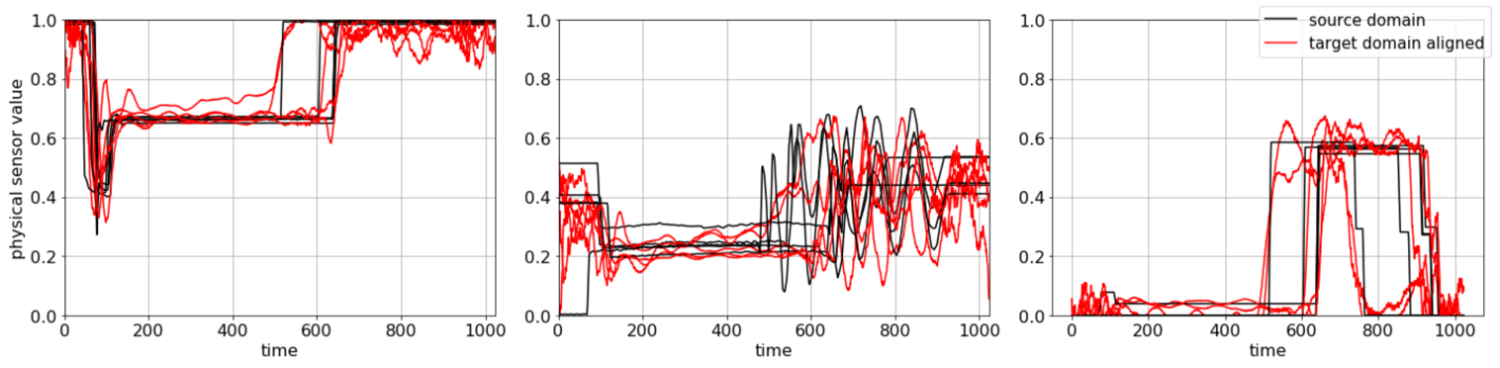


(b)

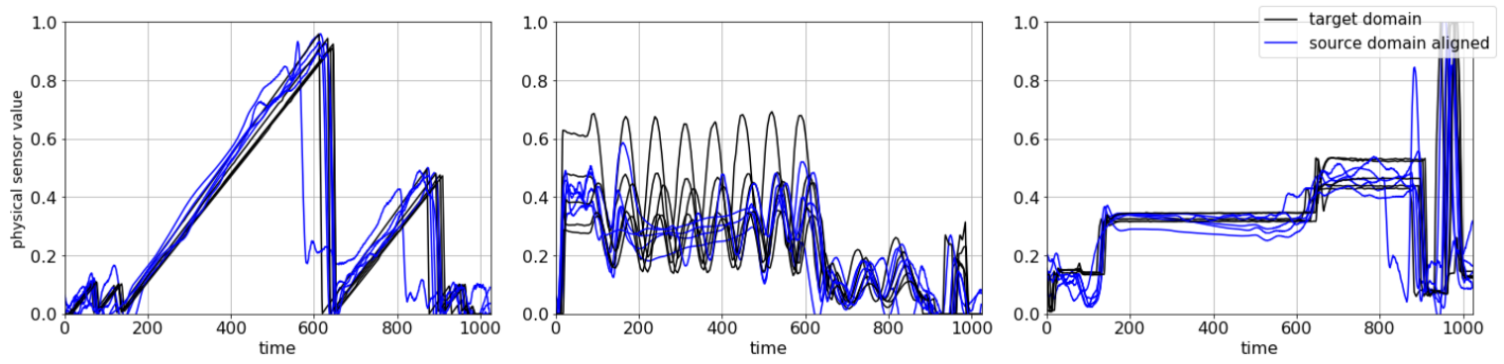


(c)

Figure 8.17: True vs. predicted scatter plot for DBACS before and after alignment. Predicted versus true labels from source and aligned target domain.(a) shows predictions of aligned target data after mapped to source space by a randomly initialized aligner, (b) predictions of aligned target data mapped to source space after the pretraining of the aligner is done and (c) predictions of aligned target data after DA training with DBACS is done. Only test data is presented as a matter of form. The test source data is colored in blue, the the aligned target test data is colored in red. The effect of the adaptation for the predicted values in the target domain is visible comparing (a),(b) and (c). The adaptation brings the distributions of the two domains to a common representation space hence enabling high quality predictions for both domains.



(a)



(b)

Figure 8.18: Aligner F and G visualizations of 2 times 3 raw sensor measurements of both equipment types before and after the corresponding alignment. Graphical representation of two times 3 raw sensor measurements of both equipment types before and after the corresponding alignment using the corresponding aligner part. (a) shows results for trained aligner F and mapped sensor signals from target to source domain in red and compares it to corresponding original source sensor signals plotted in black. A good alignment is visible beside some small noise. (b) shows results from trained aligner G and mapped sensor signals from source to target domain in blue and compares it to corresponding original target sensor signals plotted in black. A good alignment is visible as well. The x axis shows the timestamps of the sensor signals, y axis the sensor measurement values.

To conclude the first part of the result section, we stress the fact that there exists multiple sensors from both equipment types that show almost identical behavior. This is expected since the equipment types come from the same vendor and are suitable for similar process types, nevertheless it shows that the alignment works also without the usage of any expert knowledge hence purely data driven. See Figure 8.19 for features showing very similar behavior. A detailed analysis of feature pairing follows below.

For the second part, first results for (A.) correlation analysis are presented including feature pairing and distribution comparison using FID. In order to speed up and reduce data size we first compute over time average value for each sample and feature and use those as stationary input for correlation analysis. Since this do not give any significant correlated features we do not present any results here. Hence we keep the original data and just reshape the 3 dimensional sample into a two dimensional one by concatenating and subsequent reshaping where each value at each time step as separate sample:

$$(samples, timesteps, features) \rightarrow (samples \times timesteps, features).$$

We only consider significant correlations with $r > 0.7$, if a feature correlates over that threshold with multiple other features from the other equipment, the feature pair with the highest correlation value is chosen. Selected feature pairs and its average correlation value is given in Table 8.11. Sensor pairs (10,4), (3,12) and (15,38) are presented in Figure 8.19.

Feature pairs and corresponding Pearson's r

Source feature index	Target feature index	Pearson's r
0	0	0.97
3	12	0.78
5	22	0.72
7	6	0.79
10	4	0.89
11	3	0.73
15	38	0.77
23	19	0.75
26	21	0.75

Table 8.11: Selected feature pairs with correlation scores averaged over 5 fold CV.

FID between features that are matched from source and target by correlation is tested and compared to FID of train and test also with reduced features from only one domain. FID of original versus aligned features created by DBACS is tested as well. Inner domain FID - meaning FID between train and test set of one domain - is very close to 0 with $2e^{-16}$ for equipment type 1 and $7e^{-16}$ for equipment type 2. Outer domain distance for paired features based on correlation analysis is 0.25, in comparison to outer domain FID after DBACS alignment with 0.01. It hints that DBACS with its complexity is superior towards correlation analysis and that chosen subset of features do not contain all necessary information. To confirm, we train the 1DCNN prediction model again with reduced number of features based on detected pairings of source and target inputs. The performance is presented in Table 8.12. Using only a reduced number of features leads to overall lower accuracy and higher overall error for dedicated models as well as a model trained on both domains - using a common model is enabled by the feature pairing. Since overall MAE is significant higher for reduced input also for complete supervised training on both domains, only limited improvement and knowledge gain is expected by further modeling and analysis.

VM prediction model performance for paired features

	Source domain		Target domain	
	Train MAE	Test MAE	Train MAE	Test MAE
Train on source	0.15	0.16	0.52	0.52
Train on target	0.24	0.25	0.46	0.46
Train on both	0.15	0.16	0.15	0.16

Table 8.12: Results for VM prediction models that are trained with reduced number of features that are paired and selected via correlation analysis.

Second, results for (B.) PCA analysis are presented. Principal components (PCs) are computed for each equipment type separately and then compared. PCA expects a two dimensional input. Again, the 3 dimensional input is reshaped into a two dimensional one by treating each value at each time step as separate sample as done for the correlation analysis. For each equipment type we select the first 10 PCs in order to cover around 95% variance and to create same dimensional input space. For equipment type 1 we cover 97% of the variance and for equipment type 2 we cover 94%. We test FID between PCs of equipment type 1 (source) and equipment type 2 (target) and compare this to the FID between train and test data from one equipment type only (source respective target only). Inner domain FID - meaning FID between PCs of train and test set of one domain - is very close to 0 with $7e^{-16}$ for equipment type 1 and $7e^{-16}$ for equipment type 2. Outer domain distance for features mapped onto the PCs is 0.03, that is slightly higher in comparison to outer domain FID after extended DBAM alignment with 0.01 but significant lower compared to correlation bases paired features with 0.25. We also do observe correlation between the first with $r_1 = 0.68$ and second PC with $r_2 = 0.82$ from source and target. Further PCs do not show significant correlations. Again, we train the 1DCNN prediction model with reduced number of features based on PCA of source and target domain. In order to use time series model we reshape the samples into its original form after PCA is applied. The performance is presented in Table 8.13

VM prediction model performance for PCA based principle components

	Source domain		Target domain	
	Train MAE	Test MAE	Train MAE	Test MAE
Train on source	0.09	0.09	0.32	0.33
Train on target	0.47	0.47	0.12	0.13
Train on both	0.10	0.14	0.09	0.14

Table 8.13: Results for VM prediction models that are trained with reduced number of latent features that are created via PCA.

Using a reduced number of latent features leads to comparable high accuracy and small error for the test sets when used for dedicated models and a slightly higher error for source as well as target test set with one model trained on both domains - using a common model is again enabled by the usage of same dimensional latent feature space. Nevertheless, a direct transfer of a dedicated model is still not recommendable, neither from source to target nor target to source.

For creating PCs, only inner domain relations are considered so far. Ks test on PCs confirms existing differences since the average p value is almost 0 and the ks score is 0.17 in average. Hence the null hypothesis is rejected and an optional DA on the PCs is applied. As already discussed, electing suitable domain adaptation approaches as benchmarks besides the DL based ones is limited. We first apply CORAL and optional TCA. TCA is a strong but computationally expensive approach. Hence we can only select a small sample set (1% of the original sample size) for training. We test for number of components set to 2,5 and 10 and test linear as well as radial basis functional (rbf) kernel. Nevertheless no sufficient adaptation results could be generated and we skip result presentation. The FID score for outer domain distance after CORAL on the latent features generated by PCA is significant lower than before with 0.0001 for train and 0.001 for test. The adapted samples are used to bring it back in its original time series format. The back-shaped date is used to train a prediction model with the previously selected 1DCNN architecture.

VM prediction model performance for PCA plus domain adaptation

	Source domain		Target domain	
	Train MAE	Test MAE	Train MAE	Test MAE
PCA	0.10	0.14	0.09	0.14
PCA + CORAL	0.08	0.9	0.12	0.13

Table 8.14: Results for VM prediction models that are trained in a multi step approach. 1. reshape data by concatenating into 2 dimensions, 2. create reduced number of latent features via PCA, 3. optional apply CORAL and TCA for domain adaptation 3. reshape back into original time series format, 4. train 1DCNN VM prediction model with aligned features. All models are trained on aligned labeled data from both source and target.

Finally, the results for (C.) (K)CCA analysis are presented. With PCA only a limited amount of significant correlations between principle components from source and target can be observed. CCA takes explicit care of this relationship when computing canonical components. Due to computational complexity leading to memory issues, KCCA can only be applied using 1% of available samples. Unfortunately no high enough accuracy and usable results could be achieved with this very limited amount of data. Hence, only the results for CCA based on linear transformations and correlations are presented in the following.

CCA expects a two dimensional input. Again, the original data is kept and reshaped from the 3 dimensional sample into a two dimensional one by treating each value at each time step as separate sample as we do for the correlation analysis. First, linear transformation with CCA is applied. For each equipment type, 27 canonical components (CCs) are selected since it shows the most stable results. FID is tested between CCs of equipment type 1 (source) and equipment type 2 (target) and the resuts are compare to the FID between train and test data from one equipment type only (source respective target only). Inner domain FID - meaning FID between CCs of train and test set of one domain - is very close to 0 with $5e^{-9}$ for equipment type 1 and $2e^{-9}$ for equipment type 2. Outer domain distance for features mapped onto the CCs is very close to 0 with $1e^{-9}$, that is significant lower in comparison to outer domain FID after DBACS alignment, after PCA as well as significant lower compared to correlation bases paired features. We also do observe correlation between the first five CCs higher than $r = 0.5$. First ten CCs show correlations higher than $r = 0.15$. Again, the 1DCNN prediction model is trained with 27 mapped features based on CCA of source and target domain. In order to use time series model the samples are shaped back into its original form after CCA is applied. The performance is presented in Table 8.15

VM prediction model performance for CCA based canonical components

	Source domain		Target domain	
	Train MAE	Test MAE	Train MAE	Test MAE
Train on source	0.12	0.13	0.29	0.29
Train on target	0.29	0.29	0.10	0.14
Train on both	0.10	0.13	0.12	0.14

Table 8.15: Results for VM prediction models that are trained with latent features that are created via CCA.

Using a CCA based latent features show very similar behavior compared to PCA based latent features. It leads to comparable high accuracy and small error for the test sets when used for dedicated models and a slightly higher error for source as well as target test set with one model trained on both domains - using a common model is again enabled by the usage of same dimensional latent feature space. Nevertheless, a direct transfer of a dedicated model is still not recommendable, neither from source to target nor target to source.

The cross-modal retrieval method is tested as presented in [81] where the inner product of the latent features respective CCs are used for so called mate-based retrieval meaning finding the sample pairs with the highest inner product and use the labels of the corresponding paired labeled sample for solving the task. In the paper, cross-modal retrieval is used for classification, here it is transformed for the VM regression task at hand. Selecting only one sample based on maximized inner product leads to errors with MAE of 0.24 comparable to what is achieved using correlation analysis. Selecting 10 respective 30 closest samples based on inner product and using the average label value both gives MAE of 0.19. This goes in hand with the paper results mentioning that this approach is most successful for tasks where the relevant information can be stored in a low number of CCs.

Ks test on CCs confirms existing differences since the average p value is almost 0 and the ks score is 0.07 in average. Hence the null hypothesis is rejected and an optional DA on the CCs is applied. Again we apply CORAL and skip TCA due to computational limitations. The FID score for outer domain distance after CORAL on the latent features generated by CCA stays very close to 0. We use the adapted samples to bring back in its original time series format and use them to train again a prediction model with the previously selected 1DCNN architecture.

VM prediction model performance for CCA plus domain adaptation

	Source domain		Target domain	
	Train MAE	Test MAE	Train MAE	Test MAE
CCA	0.10	0.13	0.12	0.14
CCA + CORAL	0.07	0.8	0.13	0.13

Table 8.16: Results for VM prediction models that are trained in a multi step approach. 1. reshape data by concatenating into 2 dimensions, 2. create reduced number of latent features via CCA, 3. optional apply CORAL and TCA for domain adaptation 3. reshape back into original time series format, 4. train 1DCNN VM prediction model with aligned features. All models are trained on aligned labeled data from both source and target.

Optional DA on top shows slightly improved results if model is trained on data from both domains (see Figure 8.16) but does not provide usable features for transfer of a dedicated model since accuracy does not significantly improve for the domain not used for training.

8.5 Enabling Equipment Matching

Having with DBACS a methodology that allows parallel training and transfer in both directions - source to target but also target to source - mis- or abnormal behavior detected for aligned data can not only be compared to abnormal as well as normal data from source but also can be mapped back into its original space and analyzed there. These kind of comparison enables equipment matching for nonidentical equipment as it was described above namely finding and eliminating setting and performance differences for increased control based on statistical methods applied on sensor data. First, source signals are compared with its cycled signals on the signal shape itself as well target signals with its cycled target signals. Examples from both are presented in Figure 8.20.

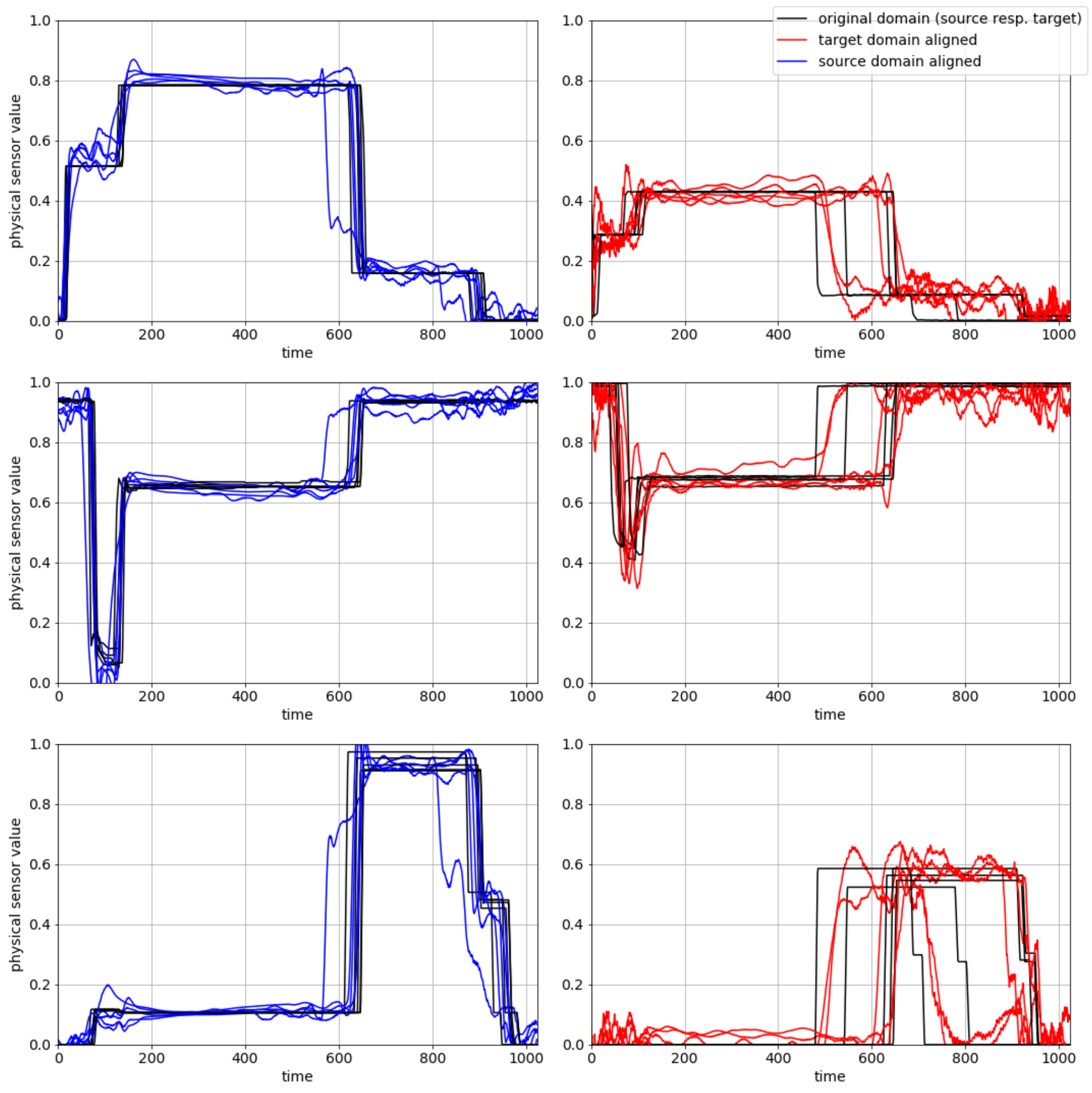
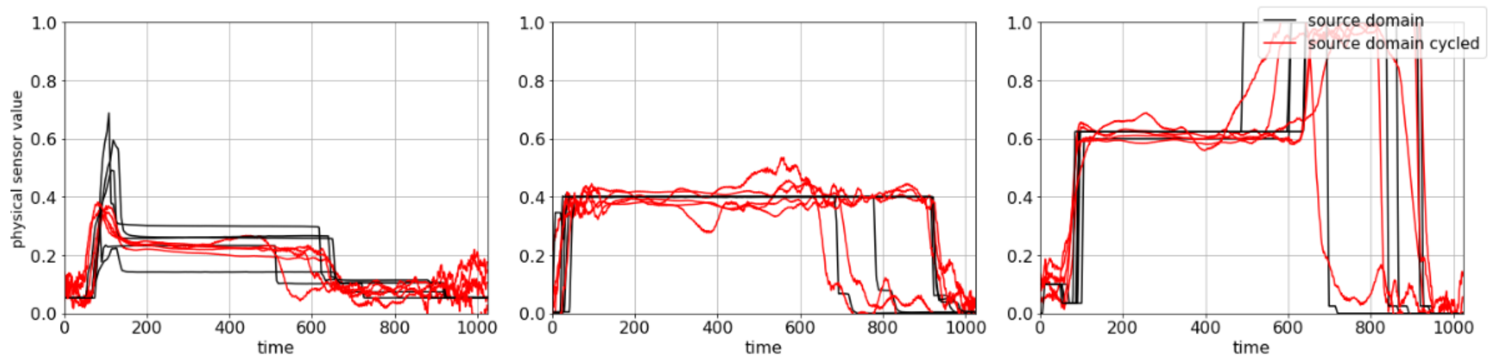
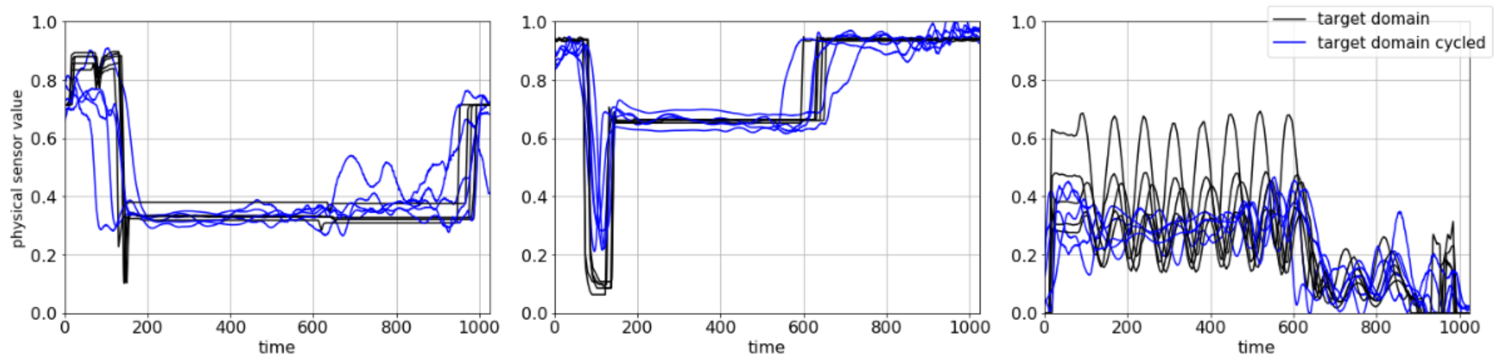


Figure 8.19: Aligner F and G visualizations of 3 times 2 paired raw sensor measurements of both equipment types before and after the corresponding alignment. Graphical representation of 3 times 2 raw sensor measurements of both equipment types before and after the corresponding alignment using the corresponding aligner part. Results for trained aligner F and mapped sensor signals from target to source domain are marked in red and corresponding original source sensor signals plotted in black (right column). Results from trained aligner G and mapped sensor signals from source to target domain are marked in blue and corresponding original target sensor signals are also plotted in black (left column). A good alignment is visible as well and visual inspection shows identical sensors exist on both equipment types.



(a)



(b)

Figure 8.20: Aligner F and G visualizations of 2 times 3 cycled raw sensor measurements of both equipment types in its original form as well as after its bijective mapping. Graphical representation of 2 times 3 raw sensor measurements of both equipment types in its original form as well as after its bijective mapping. (a) shows results for source signals and cycled source signals from source to target to source domain. The cycled signals are plotted in red and compared to its original source sensor signals plotted in black. A good alignment is visible beside some small noise. (b) shows results for target signals and cycled target signals from target to source back to target domain. The cycled signals are plotted in blue and compared to its original target sensor signals plotted in black. A good alignment is visible as well. The x axis shows the timestamps of the sensor signals, y axis the sensor measurement values.

The differences within source domain of samples having a high, middle and low prediction value are checked. The middle prediction is the preferred and targeted one. Figure 8.21 shows euclidean barycenter averages of tree example signals from source domain for low, middle and high label values. Sensor offsets for deviating metrology measurements are clearly visible for some of the signals. For final equipment matching, there is a comparison

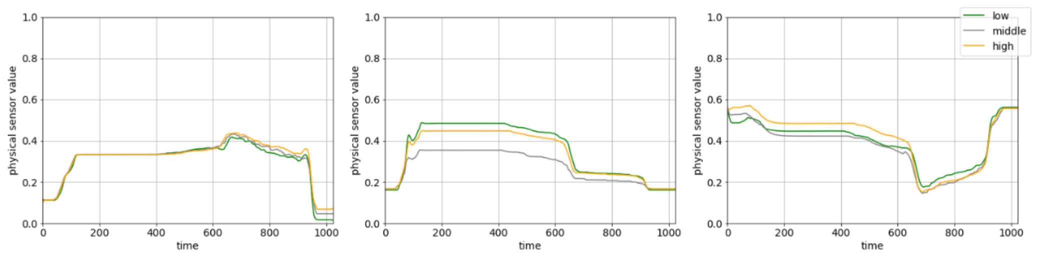


Figure 8.21: Comparison of raw source sensor measurements via barycenter average grouped into low, middle high label values. Graphical representation of euclidean barycenter averages for 3 example sensors of the source domain. The x axis shows the timestamps of the sensor signals, y axis the sensor measurement values. Example sensor measurements of samples corresponding to low label values - meaning values smaller 0.1 - are plotted in green, example sensor measurements of samples corresponding to middle therefore preferred label values - meaning values around 0.5 - are plotted in grey and example sensor measurements of samples corresponding to high label values - meaning values higher than 0.9 - are plotted in orange. Sensor offsets for deviating metrology measurements are clearly visible.

between preferred shape of signals from the source domain meaning signals with metrology measurements close to 0.5 to corresponding and deviating signals from target domain. Therefore, DBACS is used to map selected source signals into the target domain. Hence, signals representing recipe input can be tuned to reach an optimal setting while signals from feedback sensors can be tracked and compared for better understanding of misbehavior. Different sensors measurements and their euclidean barycenter averages of groups according to low, middle and high metrology measurements are shown in Figure 8.22 and compared to mapped sensor signals (source to target) corresponding to the middle meaning preferred metrology group in the source domain. Some signals do not show shape differences according to their label value while others clearly show an overlap of middle mapped source and middle target grouped sensors versus shifts of sensors corresponding to low respective high target metrology measurements.

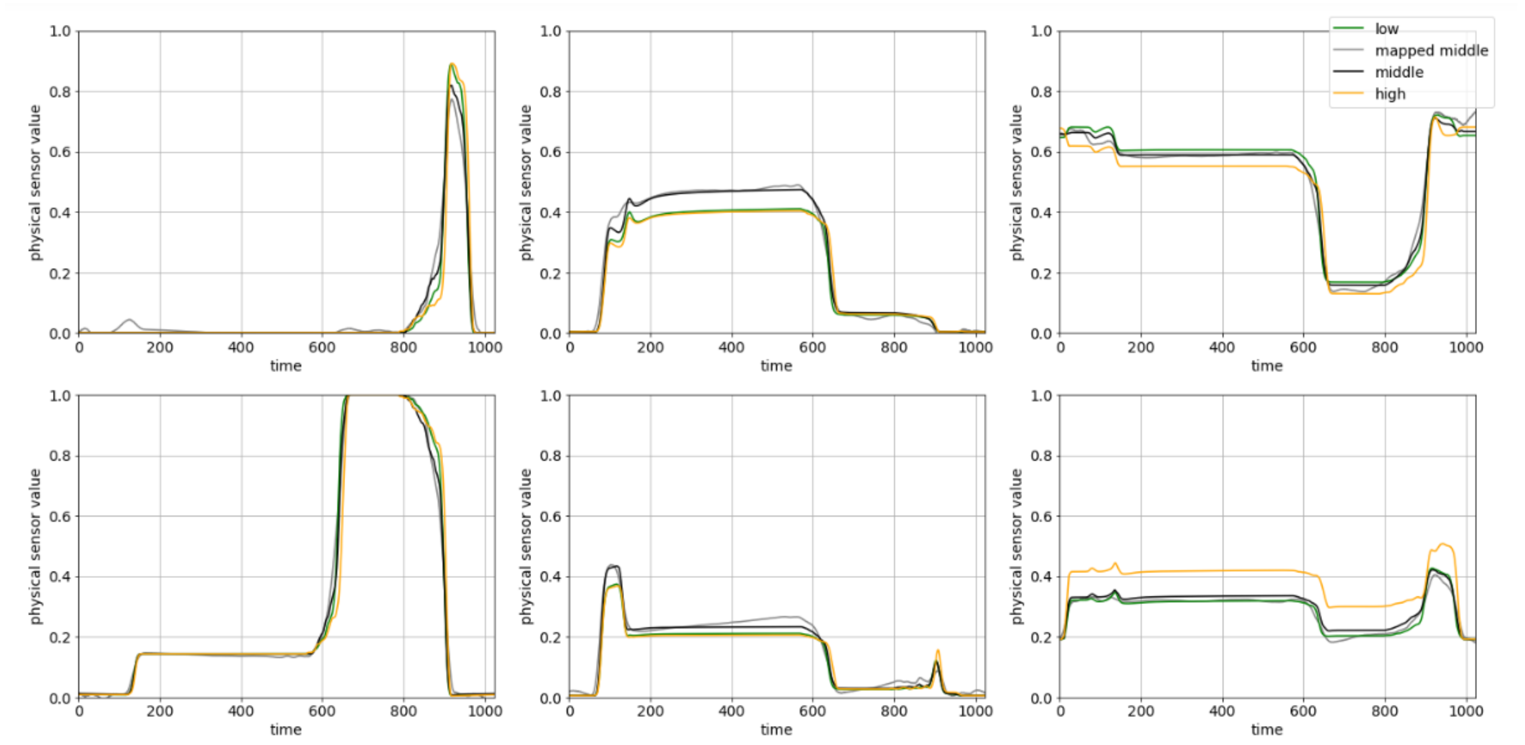


Figure 8.22: Visualization of equipment matching: mapped source sensors vs. target sensors grouped into low, middle high label values. Graphical representation of 6 sensor signals. The plots show euclidean barycenter averages of signals. Groups are defined by their metrology values and plotted in different colors. The x axis shows the timestamps of the sensor signals, y axis the sensor measurement values. Selected target sensor measurements of samples corresponding to low label values - meaning values smaller 0.1 - are plotted in green, example target sensor measurements of samples corresponding to middle therefore preferred label values - meaning values around 0.5 - are plotted in black and example sensor measurements of samples corresponding to high label values - meaning values higher than 0.9 - are plotted in orange. Mapped sensor signals corresponding to source samples with middle label values are colored black. Sensor offsets for deviating metrology measurements are clearly visible from middle target as well as mapped middle source signals.

8.6 Conclusion

In this chapter a virtual metrology use case for a plasma etching process is discussed under various perspectives: First a VM modeling transfer task for homogeneous data representations coming from two parallel running chambers is discussed. Transfer is necessary since data from different chambers show differences in their data distributions hence a dedicated source model is not able to compute sufficient predictions hence small enough performance errors when used on the target domain. First the VM prediction model itself is tested with different classical statistical methods. Confirming latest research results, the deep learning model using 1dCNN architecture on raw sensor measurements shows significant better prediction performance compared to classical approaches. Introducing more time series suitable DL methods like TCN and LSTM show their advantages above 1dCNN for prediction but also transfer. Nevertheless, the transfer capabilities of all models are limited, hence DBAM is applied a method that is introduced to allow interpretability and comparison of aligned versus original sensor signals. Overall, the results can be summarized as follows:

- DBAM shows its functionality for for stationary and time series under different architectures and data types;
- raw sensor measurements in form of time series data is preferable, giving best prediction results since they contain all information that is registered during the process;
- recent publications suggest the combination of 1dCNN and LSTM architectures, exploiting the best of both worlds. Applying those finding to the VM use case show on the one hand improvement for the prediction task and on the other hand confirms the superiority of those methods. Those combined architectures can also be used for DBAM.

The second part of this chapter covers a VM modeling transfer task for heterogeneous data representations coming from two non-identical equipment running the same process. Since a common prediction model is not possible, DBACS an extented version of DBAM is applied. Therefore, a common prediction model can be used and the sensor measurements of both equipment can be compared after transformation. The cyclic architectures allow equipment matching on top of adaptation. The appraoch is compared to methods inspired by multi-view lwearning but the DL based approach show again its superiority especially for complex VM modeling tasks. Overall, the results of the second part can be summarized as follows: Heterogeneous VM2:

- DBACS is applied to a heterogeneous domain adaptation task and shows teh possibility of a common prediction model for different data representations;
- multi-task learning in the context of semiconductor manufacturing and process control it exploited as benchmark;
- linear transformation show their potential when it comes to heterogeneous data representations, but also have limitations for kernel since kernel too computationally expensive. Nevertheless, the selected benchmark models based on their simplicity are promising alternatives especially for small amount of data;
- DBACS enables equipment comparison and matching so a combination of domain adaptation and matching is presented.

The presented work can be further enhanced by smart sampling approaches that could enable an even better performance of the most simple models but as well speed up the deep learning based transfer approaches.

9 Predictive Maintenance (PdM)

Predictive maintenance (PdM) as maintenance option targeting to foresee failure or estimate time to failure based on data and statistical methods, is a key part of industry4.0 and manufacturing automation. Successful PdM leads to longer equipment uptime, production speed improvement, better quality assurance and lower costs. Down events or equipment breakdowns can be triggered by various causative events depending on equipment, equipment parts and running processes besides other influential factors. While scheduled maintenance is planed based on average usage and average degradation of equipment, PdM is especially useful for avoidance of unscheduled breakdowns and enables best effort and preparation for upcoming maintenance tasks.

Predictive maintenance systems exploits raw sensor measurements or descriptive statistics of sensor data that is already collected during a production process by the equipment itself avoiding additional costs. In addition so called (RTC) data describing the equipment status in the sense of up, down, stand-by, productive beside others is used to define equipment life cycles as well as distinguish between scheduled and unscheduled maintenance events. PdM developed, evolved and improved tremendously over the last years as described in Iskandar et al. [98]. Main advantages in general but also compared to traditional maintenance approaches are:

- *uptime* shorter down times due to immediate availability of people as well as necessary spare parts.

- *speed* arrange compensation and updated dedication of running equipment to avoid bottlenecks
- *quality* no further harm towards tool and product compared to run-to-fail
- *costs* proper and full usage of all equipment parts in a controlled manner

The research presented in this chapter are obtained in collaboration with University of Padua, Information Engineering Department, Prof. Dr. Alessandro Beghi, Prof. Dr. Gian Antonio Susto, Luciano Lorenti and Marco Zanetti. The focus topic is predictive maintenance (PdM) for implant equipment including different failure types with focus on plasma flood gun related breakages.

9.1 Introduction

In manufacturing, maintenance describes the process of equipment life cycle including inspections of the equipment, the equipment repair and anything else related to know and control the condition of production equipment.

In this chapter the focus is on the equipment health and maintenance of five implant equipment. In order to enable a wide usage of a highly automated prediction model, deployed methods need to be transferable and applicable when confronted with all five equipment or in a bigger picture running modes, failure types and process history up to different equipment types. Hence, this existing high data complexity demands more effort and research in the field of scalability especially when it comes to continuously needed equipment transfer. This aspect of existing equipment behavior differences leading to differences in data distributions represents a challenge but also an opportunity to exploit the full potential of all collected equipment and failure data, hence equipment group wide therefore scalable installation of automated prediction of remaining useful life (RUL).

Here, those issues are addressed by exploiting domain adaptation methods for RUL predictions for five implant equipment grouped into two cluster showing similar behavior. In addition, diverse influences and data imperfections are discussed that typically occur for PdM related tasks. Differences in data - respective the data distribution - can occur from the beginning and are caused by not or not directly influenceable factors.

The upcoming sections are structured in the following way: First use case specific literature is covered from a historical overview up to classification methods for PdM, regression tasks for PdM and closing with adaptation related methods for PdM. Then the PdM use case for Remaining Useful Lifetime estimation is introduced. This is followed by a detailed discussion on data preprocessing steps: filtering of non-natural deaths, equipment grouping, recipe specific outlier removal, recipe effect removal, scaling, feature selection and time window definition. The following experiments are split in two; first the focus is on the RUL prediction model and testing of prediction benchmarks. Then the results for domain adaptation via DBAM as well as DBACS are presented. In the last section called *imperfect data* feature- as well as instance-based domain adaptation benchmarks are covered and results as well as overall difficulties and hurdles in PdM scalability are discussed.

9.2 Literature

Due to the long history of maintenance and all its aspects and advantages, a long list of of literature is available: A pre industry4.0 and historic overview of maintenance is presented by Mobley [147], a current survey with focus on PdM for industry4.0 is given by Krupitzer et al. [117]. A project-based summary of different PdM applications for smart factories is given by Cerquitelli et al. [27]. Overall random forest (RF), neural networks (ANN), support vector machine (SVM) and k-means clustering are the most commonly used methods for PdM as stated in Carvalho et al. [26].

In the following, a short overview is given of selected survey paper that cover the overall history of maintenance. Then, for a deep-dive we further distinguish between literature on classification and regression tasks and have a separate look at published approaches related to transfer learning with a focus on literature related to semiconductor manufacturing. Next, the experimental design is detailed out and DBAM, DBACS as well as selected feature- and instance based domain adaptation benchmark models are evaluated. The use case chapter closes with a detailed discussion on imperfect data, data complexity and their influence on PdM prediction in general.

Classification task for PdM

Susto et al. [194] compare multiple standard ML-methods like descriptive statistics, k-nearest neighbors (k-NN) and different SVM methods - linear, gaussian kernel, standard. They are used in a multiple classifier setting where different classifier models with different input and focus are evaluated at the same time for better results. Those classification types used in [194] for PdM either predict if the next or current run is faulty or to classify the following $m \in \mathbb{N}$ runs for early detection of possible failures hence avoidance of breakdowns. Strong results using LSTM for predicting RUL ranges or time intervals instead of classes are presented by Vishnu et al. [209] (winner of 2018 PHM Data Challenge) and by Jalali et al. [101].

Regression task for PdM

A very common regression target for PdM is to predict remaining useful life (RUL) of the corresponding equipment, see Figure 9.1. Heimes [85] are successfully comparing performance of different MLP based architectures for RUL prediction. In addition, different ways - upper limited RUL versus linear decreasing RUL - of modeling the target output are considered. Jalali et al. [101] present an estimation of time-to-failure (TTF) testing very similar ML models as seen for the PdM related classification problem described above. Ensemble learning and sample (in this case degradation-dependent) weighting are successfully implemented by Li et al. [133] for a semiconductor PdM use case. Li et al. [131] makes use of well studied deep convolution neural networks based on one dimensional convolutional filter, showing its superiority towards classical ML methods. LSTM are the leading architecture when it comes to PdM and RUL prediction for time series input data, see Wu et al. [218] where multiple deep recurrent neural network-based approaches are compared and Huang et al. [96] where early fault detection combined with LSTM shows the best performance. Main idea of both articles is to focus on the degradation phase for improved prediction; meaning to use early degradation classifier respective a degradation related feature together with a recurrent neural network architecture. Hsu et al. [95] confirms the superiority of time series based RNN methods by combining TCN for feature construction and LSTM plus attention to capture time dependencies.

Adaptation related methods for PdM

Adaptive methods, like ensemble learning and sample weighting, are well researched in general but also present in PdM related literature where they are used to cover the process complexity behind different product and technologies besides other influential factors. An overview of challenges towards PdM is given by Compare et al. [44]. While Li et al. [133] show the power of sample weighting and ensemble learning for PdM RUL prediction for fault detection, Lu and Lee [137] show kernel-based ensemble techniques applied to RUL prediction and Hsu and Chien [94] as well as Kang and Kang [107] the power of ensemble learning applied with CNN for wafer map classification with high data and pattern complexity, those ensemble and weighting methods are rarely applied for transfer/DA in semiconductor PdM setting. Their success nevertheless is shown for other applications: Wang et al. [213] applies TrAdaBoostR2 (by Pardoe and Stone [158]) to a fault detection use case, Xiao et al. [220] combines CNN with TrAdaBoost for classification of induction motors under different operating conditions and fault types and de Mathelin et al. [48] discuss multiple feature-based (encoded feature space/subspace) as well as instance-based (sample weighting) DA methods for a tire design use case. From the class of sample weighting, k-nearest neighbor [58] is an elegant choice because of its non-parametric nature, [136] extents it to a transfer learning setting. Zhang et al. [234] exploits NNW for a semiconductor fault detection use case with multiple process modes. de Oliveira da Costa et al. [50] presents a LSTM-DANN (besides other time series suitable architectures) for RUL prediction and tests it against non-adaptive methods, Transfer Component Analysis (TCA)(Pan et al. [156]), Correlation Alignment (CORAL) (Sun and Saenko [189]) and the supervised domain adaptation method proposed in Zhang et al. [233] (based on transfer learning and target fine-tuning applied on RNN) on the NASA Commercial Modular Aero-Propulsion System Simulation (C-MAPPS) Turbofan degradation data sets ([175]). Li et al. [132] proposes Adaptive Batch Normalization (AdaBN) to increase the generalization ability of a ANN by simply changing the batch normalization layers according to the target domain. AdaBN applied on PdM is presented by Li et al. [129]. Another research applying AdaBN and improves additionally network parameter finding and hyperparameter tuning using an bayesian optimization is presented by Li and He [128]. Liu et al. [134] presents the original DANN approach combined with different recurrent neural network architectures on the data from the 2018 PHM Data Challenge to align failure modes as well as equipment in a two step approach.

9.3 Remaining Useful Life (RUL) prediction for Predictive Maintenance (PdM)

We present an use case based on implant process data collected during front-end production. For a description of the implantation process see Part I, Chapter 2.2.The data is collected from 5 identical-in design implant equipment running in parallel. The equipment is clustered into two groups that share same running modes and data behavior.

The task at hand is to create a prediction model to forecast so called remaining useful life (RUL) of each equipment. Therefore, fault detection and classification (FDC) data derived from the raw sensor measurements using descriptive statistics is taken into account for best model selection. Labels are generated based on the available information in the resource time classification (RTC) data. It logs the status and status changes of an equipment like productive, stand-by, unscheduled down, scheduled down, repair. Detailed remarks on the repair can be used for detailed failure labeling and helps to distinguish between run-to-failure, unscheduled down and scheduled or preventive down. It can be used to classify different failure types and corresponding equipment parts like ion source or plasma flood gun. Here we focus on plasma flood gun related down events.

The time period between finished maintenance event respective first time the equipment is set (back to) productive and down event is called 'life'. Mainly two ways of modeling equipment degradation time - later used as labels for supervised training - are presented in literature, e.g by [233]:

- affine-linear with negative gradient;
- piecewise affine-linear function with zero gradient first followed by negative gradient.

For a visualization of (piecewise) affine-linear target function see Figure 9.1. The selec-

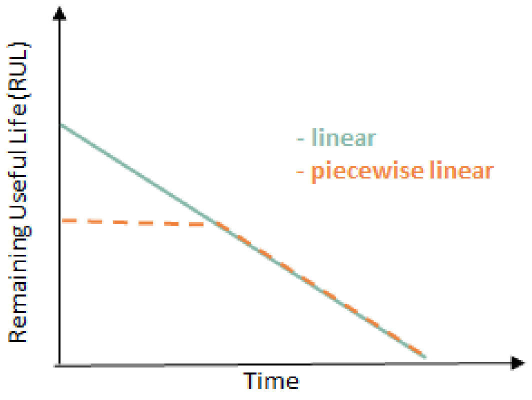


Figure 9.1: Linear and piecewise linear target/label function for RUL. Linear and piecewise linear target/label function as presented in [233]. For the presented use case the linear target function is selected (blue).

tion of a piecewise affine-linear function is based on the assumption that the degradation at the beginning is negligibly small respective hard to estimate. Hence modeling the decrease only makes sense after a certain 'anomaly point'. We consider a group of equipment with known differences in their running modes. Hence a specific 'anomaly point' would be hard to define. Therefore, we choose the linear modeling of the degradation as target function. There exists different failure modes that can lead to unscheduled down. Plasma flood gun is a very important part of implant equipment often responsible for breakage and therefore often used as target failure group in PdM literature for semiconductor manufacturing.

Data Preparation

We consider five different implant equipment from the same production side. In general, they are able to run the same processes and no equipment dedication is specified:

- *IMPP13-01*
- *IMPP13-03*
- *IMPP13-03*
- *IMPP13-04*
- *IMPP13-05*.

The considered time period is between beginning of the year 2020 until April 2022. Data is separately collected for each equipment and only whole lives within the selected time period are taken into account.

Each processed wafer is monitored and raw sensor measurements are taken during the process and automatically used to compute descriptive statistics known as Fault Detection and Classification data (FDC). For each equipment RTC data is stored and used to mark status changes like down events. The information retained from RTC is used to filter out scheduled down events in the training data to simulate optimal run-to-failure conditions.

A first selection of features is done based on expert knowledge. In addition incomplete lives from start and end of the whole time period as well as very long or very short lives are discarded based on the 3rd percentile. The considered lives for all five equipment are presented in Figure 9.2 and marked with different colors. Overall 49 lives are taken into account. The following preprocessing steps are applied to the features and the corresponding lives due to their benefits for the modeling process:

Flagging versus filtering of non-natural lives A discussion about flagging versus filtering of non-natural deaths is necessary since no run-to-failure strategy is applied on the involved equipment. If a life is ended early for example by scheduled maintenance, data

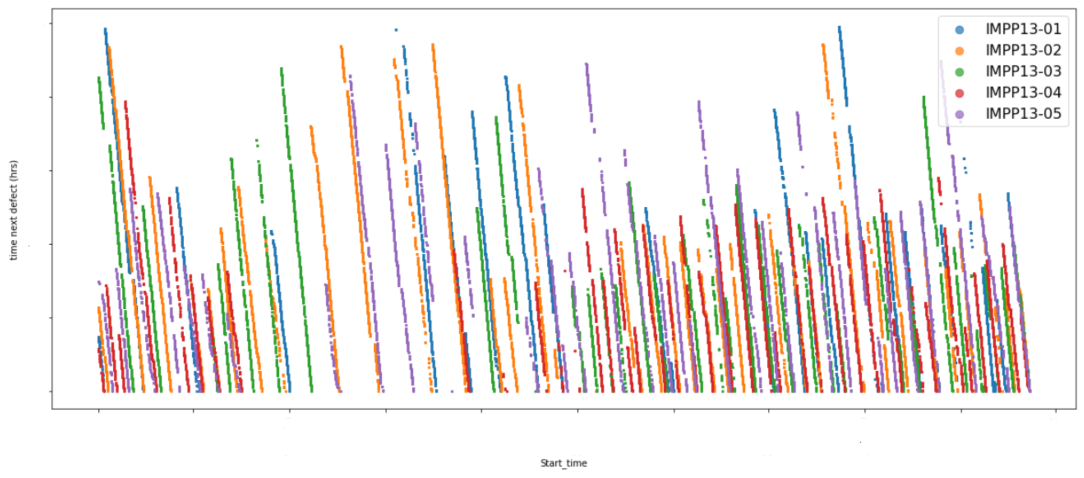


Figure 9.2: Visualization of equipment lives. Visualization of equipment lives in the form of linear target function. Different colors mark different equipment.

features do not - or only to a limited extent - show any or sufficient signs of degradation. Therefore the model is not able to correctly learn the relationship between input and health status of the equipment. This fact influences not only the quality of training but especially testing where the model evaluation gets unreliable if the used labels are incorrect. Three different approaches are taken into account:

- taking all lives without preselection;
- flagging non-natural deaths;
- filtering out non-natural deaths.

. Specific features show distinctive pattern e.g. large value drops or high ascend when coming closer to the end of a life. Such behavior is used to distinguish between natural and non natural deaths and to select or flag natural deaths out of all available lives. For the use case presented here two features strongly related to plasma flood gun exhibit a clear dominant behavior towards end of a life. One of them is selected to determine the true nature of a life and a corresponding value threshold is set. The feature including the threshold set to $0,1$ is visualized in Figure 9.3.

Equipment grouping The selected 5 equipment are split into two groups based on their data properties like life lengths and feature data distribution. It is confirmed by process experts that matching and standardization efforts are finished and remaining data differences

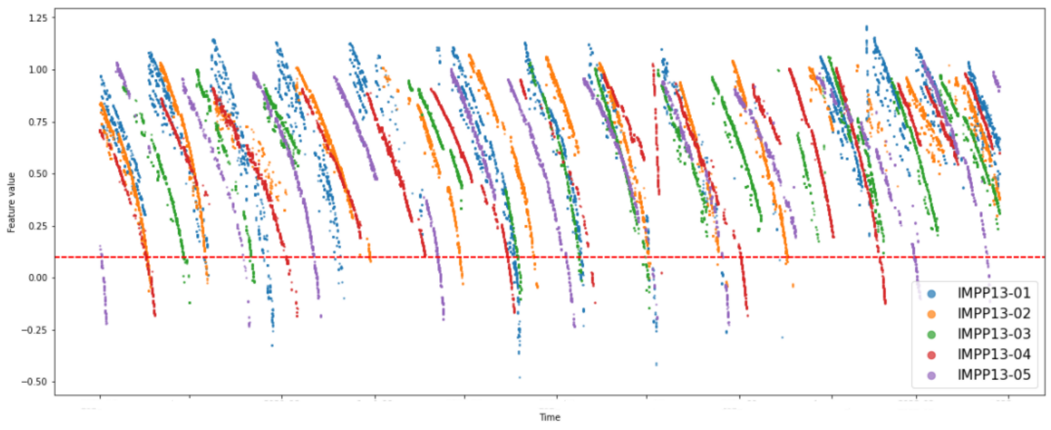


Figure 9.3: Visualization of feature thresholding for non-natural versus natural death filtering/flagging. Visualization of feature thresholding for filtering or flagging non natural versus natural deaths. Different colors mark different equipment.

need to be handled within the modeling approach. IMPP13-01, IMPP13-03 and IMPP13-04 build the first group. Group 1 is also selected as source for the domain adaptation approaches. IMPP13-02, IMPP13-05 together define group 2 that is later used as target domain for the domain adaptation modeling.

Outliers removal via interquantile range (IQR) An outlier is a sample in the data that shows some kind of different behavior compared to other observations - in the sense of distance for example. Hence, it can be assumed that it is generated by a different mechanism. While this can be of high interest during process evaluation, the idea of training a model is to learn normal or targeted behavior. Since some models are sensitive towards outliers as presented in Friedman et al. [66], we apply interquantile range (IQR) outlier removal procedure: Let Q_1, Q_3 describe the 25th, 75th percentile. Then:

$$IQR = Q_3 - Q_1 \tag{9.1}$$

$$F_l = Q_1 - 1.5 * IQR \tag{9.2}$$

$$F_u = Q_3 + 1.5 * IQR \tag{9.3}$$

F_u, F_l are used as upper and lower limits and all samples above or below those thresholds are marked as outliers and removed during training.

Recipe effect removal The description of how a process needs to be carried out on an equipment is called recipe. A recipe consists mainly of 3 components: what ingredients to use, how much of them and the set of instructions how and in what order to use them. The 'recipe effect' is a phenomena in the data where data heterogeneity occurs due to a change in running recipes. This can result in differences in the sensor readings.

To avoid misinterpretation of those naturally occurring shifts, a removal of the recipe effect is done. The basic idea is to shift the data of each feature-recipe pair by its median value. To consider time dependency in the data as well as to avoid jumps in the data caused by recipe changes, a slightly adopted version is applied that does not consider the whole time period at once but shorter time spans instead:

Let $i = 1, \dots, N$ be the number of recipe shifts, let kn^i_{rl} and kn^i_{rf} be the last value before recipe change i respective the first value after recipe change i, and x^i_r the recipe specific shift to be optimized. Then the overall shift distance d is defined as:

$$\begin{aligned} d &= \sum_i |(kn^i_{rl} - x^i_r) - (kn^{i+1}_{rf} - x^{i+1}_r)| \\ &= \sum_i |(kn^i_{rl} - kn^{i+1}_{rf}) - x^i_r + x^{i+1}_r| \end{aligned} \tag{9.4}$$

The goal is to minimize the overall shift distance d. Hence for all features and all recipes i:

$$d = \sum_i |\Delta kn - x^i_r + x^{i+1}_r| \to 0 \tag{9.5}$$

Let A be a sparse matrix with $\{-1, 0, 1\}$ entries indicating the recipe change, x the vector of all recipe shifts and Δkn the jumps of the features due to recipe changes. Then the optimization problem can be reformulated into a linear system of equations hence a matrix equation:

$$\Delta kn = A \cdot x \tag{9.6}$$

A solution of a linear system is an assignment of values to the variables x such that each of the equations is satisfied respective optimized. The results of the recipe effect removal procedure is shown in Figure 9.4.

Data scaling The models considered for this use case expect scaled input data. Since we need to cover the whole process history of an equipment in order to timely predict its degradation, different recipes are included in the data. Hence, feature as well as recipe specific scaling is applied to avoid different levels in the data that are known to not be degradation or failure related. An adopted version of Min-Max scaling is applied (see [160]). It scales the

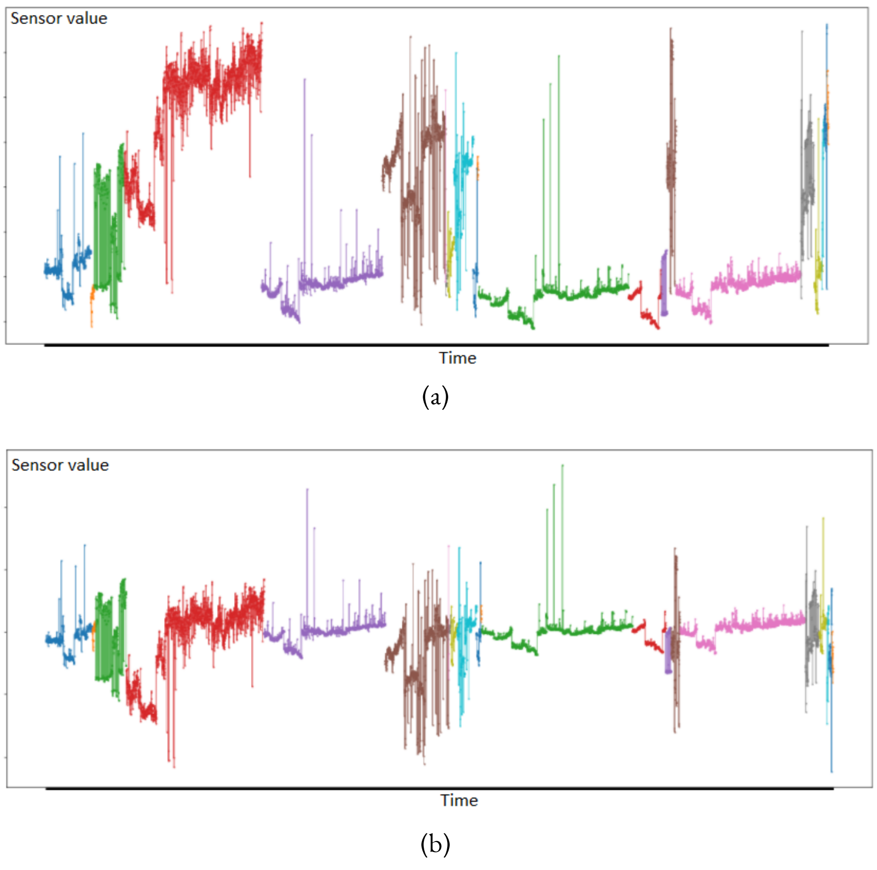


Figure 9.4: Visualization of recipe effect removal. Visualization of recipe effect removal for a selected feature. Panel (a) shows the feature for multiple lives without recipe removal and panel (b) with removed recipe effect. The alignment to one level is clearly visible. Different colors mark the different recipes.

data to the range $[0, 1]$. Outliers, that are excluded for computing the scaler, can lie outside of this range afterwards.

Let $S = \{x_i\}_{i=1}^{N}$ be a data sample from the input space X with k features and $x_i \in \mathbb{R}^k$ with $N = |S|$ the number of drawn samples in S. Let $x_{i_r} \subset x_i$ samples from feature i that belong to a fixed recipe r and $x_{i_r,max}, x_{i_r,min}$ be the maximum respective minimum recipe-feature specific sample values. Then:

$$x'_{i_r} = \frac{x_{i_r} - x_{i_r,min}}{x_{i_r,max} - x_{i_r,min}}, \forall \text{ i, r.} \tag{9.7}$$

Feature selection Both Chandrashekar and Sahin [28] and Khalid et al. [112] give a general respective ML specific comparison of different feature selection methods. Feature selection is divided into two techniques:

- *supervised* where labels are used to identify the importance or the influence of each feature on a regression or classification task;
- *unsupervised* where relationship between the features or distances are analysed without usage of labels.

Those two techniques can be further divided into:

- *Filter methods* where intrinsic properties of the features like correlation are used;
- *Wrapper methods* where a best performing (related to a task and method) feature subset is selected like for example Recursive Feature Elimination (RFE);
- *Embedded methods* that uses interactions of features like Random Forest Importance based on the Gini Impurity.

Autoencoder are a well known DL based method for feature selection and dimension reduction and very useful in high dimensional complex semiconductor environments, see for example [138].

Boruta Shap by Keany [110] is a combined method. It combines Boruta - a wrapper method selecting features by comparing the relevance of a feature with its randomly shuffled counterpart or "shadow" feature. Boruta is stable and accurate, but computationally expensive hence Shap - connecting game theory with local explanations [140] - is added to speed up the included feature ranking. For simplification and speed, BorutaShap is applied using a RF prediction model. The feature importance is shown in Figure 9.5, Figure 9.6 and Figure 9.7 respectively for accepted, tentative and rejected features. One-hot encoded recipes features are discarded due to previously removing the recipe effect, one-hot encoded equipment features are originally kept but later removed for domain adaptation. Tentative features are also kept due to preferences towards a conservative approach and usage of regularization techniques.

Time window Since we use time-series models besides models that are applied on stationary features, we reshape the preprocessed data by building time-windows in order to use them for 1DCNN and LSTM:

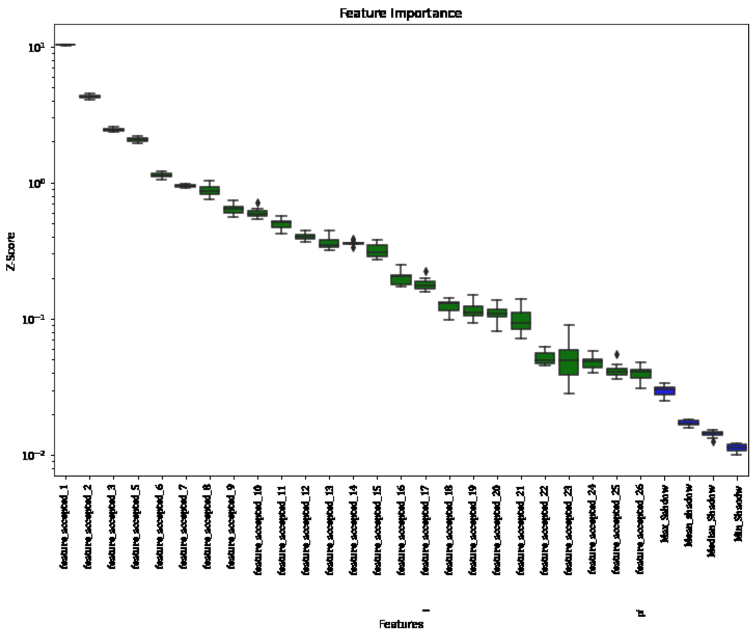


Figure 9.5: Boxplot of feature accepted by BorutaShap with relative importance

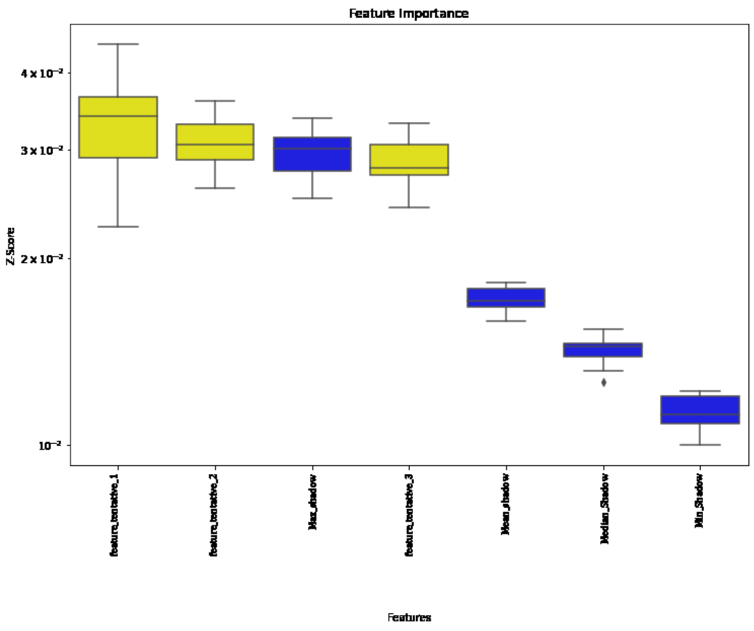


Figure 9.6: Boxplot of feature tentatively accepted by BorutaShap with relative importance

Instead considering a single value x_i per feature for a sample i to predict RUL y_i, it is extended to an interval of consecutive samples $x_{i-n}, \ldots, x_i$. The size n of this window is a

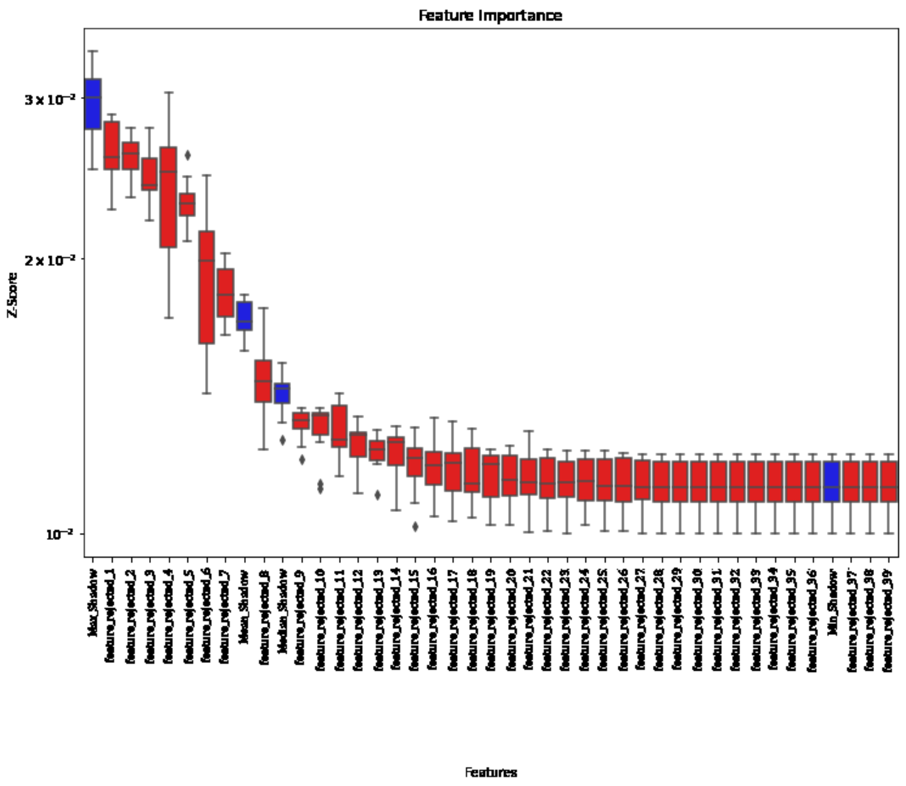


Figure 9.7: Boxplot of feature rejected by BorutaShap with relative importance

hyperparameter that needs to be defined. Overall the data is reshaped from (sample, features) to (sample, window, features).

Experimental Design

In the first part of the experiment the focus is on the RUL prediction model itself. Different benchmark models including different neural network architectures are analyzed based on their presence in literature. The best performing prediction model is selected and used in the domain adaptation part.

In the second part of the experiment different domain adaptation methods are compared to DBAM and DBACS presented in Section 4 and 5. The same features are used for all experiments, for time series input the existing features are reshaped but not further processed. Different hyperparameter settings are checked and K-fold cross validation based on lives is applied. For K-fold CV the data is split into K independent sets. A model is then trained on the union of $K-1$ sets and the remaining set is used for testing and evaluation. Mean m and variance σ of all K training rounds are then used a final model performance scores. The pseudo code for K-fold CV is presented in Algorithm 9.1.

Algorithm 9.1 Pseudo code for K-fold cross validation (CV)

```
Data: Data set $S$, number of partitions $K$, learning algorithm $A$, performance measure $L$
Result: Final scores $m$, $\sigma$
Split $S$ into $K$ partitions of same size $S_1, \dots, S_k$
 for $i = 1 \dots K$ do
   fit $A(S \backslash S_i)$
   $e_i = L(A(S_i))$
end
$m = \frac{1}{K} \sum_{i=1}^{K} e_i$
 $\sigma = \frac{1}{K} \sum_{i=1}^{K} (e_i - m)^2$
```

Benchmark Models

LR, Lasso and RF are selected as benchmark prediction models. Gradient Tree Boosting (GTB) respective its XGBoost implementation [34] is used instead of RF here for improved results. The benchmark models are described in Chapter 6. For details and explanations on ANN, 1DCNN and LSTM we refer to Chapter 3.1.

For the domain adaptation approach different benchmark models are considered that are present in PdM related publications. Based on their prediction performance, LR and ANN are selected as prediction models used for training benchmark models. If possible, benchmark methods are trained in an unsupervised DA manner meaning no target labels are used for training but only for evaluation. For feature-based DA TCA, CORAL and SA are selected. We apply different latent space dimensions for TCA: 5,10 and 20. For an theoretical and use case independent introduction and further mathematical details see Chapter 6. For instance-based DA we evaluate NNW and TrAdaBoostR2. Again more methodological details are presented in Chapter 6.

Metrics and Losses

To measure the success of the alignment between the two equipment groups as well as compare RUL prediction model performance we select two groups of losses. For more detail see

Chapter 7.

For distribution comparison and alignment evaluation, the following distribution-based losses are applied:

- For the purpose of visualization T-distributed Stochastic Neighbor Embedding (t-SNE, [206]) is applied;
- For the purpose of comparing inner domain distance - the distance between for example train and test set of one domain - and inter respective outer domain distance - the divergence between data selected from source and data selected from target domain - we use Frechet inception distance (FID).

For overall performance comparison, the following performance-based losses are applied:

- For overall performance comparison we use mean absolute error (MAE) and mean squared error (MSE).

Models and Hyperparameter

Preprocessing as described above is applied. Hyperparameter optimization using Random Search is done on all models including benchmarks. In order to avoid bias caused by particular splits, a cross validation (CV) method is applied. K-fold cross validation is used for comparing different domain adaptation approaches. In the PdM use case all data splits cannot happen within one life to avoid information leakage leading to an overestimated model performance on seemingly unseen data. Therefore only whole lives are assigned to the splits. Only lives related to natural death are selected for training. Test lives corresponding to non-natural deaths are either way removed to assure reliable high quality predictions.

The analysis is started with the selection of best performing RUL prediction model using equipment Group 1 defined as source domain. RUL prediction models and selected hyperparameter are summarized. Based on the literature review, seven different ML methods are selected as candidates for RUL prediction and evaluated. We summarize hyperparameter optimization, model architectures and implementation details:

- **Zero Rule algorithm** (or Baseline resp. Dummy regressor) where mean RUL value is always predicted. Let $n \in \mathbb{N}$ be the number of samples in the training data, $y^{pred} =$

$\{y_i^{pred}\}_{i=1}^n$ the predicted values and $y^{true} = \{y_i^{true}\}_{i=1}^n$ the observed values or labels. Then:

$$y^{pred} = \frac{1}{n}\sum_{i=1}^{n} y_i^{true};$$

- **Linear regression:** To train the Linear Regression we use the implementation from Scikit-Learn [160]. Default parameters are used;

- **Lasso Regression:** Scikit-learn implementation [160] of Lasso is used. The parameter λ is set to $\lambda = 1.2$;

- **Gradient Tree Boosting:** To train the gradient boosting mode, we use the regularized implementation called XGBoost [34]. In particular, we set $gamma = 5$, learning rate $eta = 0.005$, maximum depth of a tree $max\_depth = 10$ and number of estimators $n\_estimators = 250$;

- **Fully Connected Neural Network**: ANN model is written using the keras library [40]. It has four hidden layers with respectively 260, 130, 65 and 32 neurons in each layer. Each hidden layer use ReLU activation function and a ℓ_1 regularization equal to $\ell_1 = 0.001$. Neurons are initialized using He normal initialization [83], since it is more stable compared to Xavier Initialization when hidden layers use ReLu activation function. Each hidden layer is followed by a dropout layer, where the probability of dropping a neuron is set to $p = 0.2$. As optimizer, Adam [113] with learning rate $lr = 1e-5$, $\beta_1 = 0.9$ and $\beta_2 = 0.99$ is used. The model is trained using early stopping, with patience set to 20 epochs. The loss function used to train this network is Mean Squared Error (MSE); MAE is additionally added as second metric.

- **1 Dimensional Convolutional Neural Network**: Here we experimented with two different architectures, where one replaces the global pooling with an attention layer. 1DCNN models are implemented through the keras library [40]. The first architecture has three 1D convolutional layers with 32, 16, and 8 filter size, with kernel sizes of 5, 3, and 3 respectively and stride of 1. ReLu activation function is applied to those layers, and we use padding to preserve the input shape. The first two convolutional layers are followed by a 1D Average Pooling layer with a pooling size of 3, while a 1D Global Max Pooling layer follows the last layer. Two dense layers after the Global Max pooling layer, with 120 and 50 neurons are defined. ReLu activation function is used in those layers, together with a ℓ_1 regularization of 0.02. Each layer is followed by a dropout layer with probability $p = 0.1$. As optimizer, Adam [113] with learning rate $lr = 0.00003$, $\beta_1 = 0.9$ and $\beta_2 = 0.99$ is used. The model uses early stopping, with patience set to 30 epochs. The loss used to train the network is MSE.
  The second architecture is the similar to the first one, but no pooling layers follows

the convolutional layers. Instead, a self-attention layer is placed after the last convolutional layer. Three 1DCNN layers have a filter size of 32, 16, and 8. Two dense layers with 120 and 50 neurons respectively follows the self-attention layer. Window size is set to $n = 15$.

- **Long Short Term Memory Network**: To deploy the LSTM model, we use once more the keras library [40]. The architecture is composed of three LSTM layers with respectively 265, 128 and 64, followed by a dropout layer with $p = 0.1$. Two fully connected layers with 100 and 50 neurons follow. ReLu activation function, $\ell_1 = 0.01$ regularization are applied and dropout with $p = 0.1$ are applied. Window size is set to $n = 15$. Early stopping is used with patience set to 30. MSE loss is used once more as loss function, the learning rate is raised to 0.0001.

All ANN architectures have a single neuron in their output with linear activation function applied to it since we are in a regression setting where RUL is not scaled but true to remaining hours.

Based on the different RUL prediction models described above, different domain adaptation methods based on literature review are presented. Depending on the specific principle of the domain adaptation approach, RUL models are either directly involved in the data alignment (task specific alignment, supervised) or later involved in a second step and separately trained with already aligned features (no task specific alignment, unsupervised).

First, implementation details for DBAM and DBACS as representative of deep domain adaptive methods are presented:

- **Dann-based Alignment Model (DBAM):** For training ANN-based DBAM the previously on source trained ANN predictor is reused with fixed weights. The domain discriminator for the ANN has 4 fully connected dense layers with Leaky ReLU activation function and linear output function. Layers have size 32, 16, 8, and 4. The aligner in form of an autoencoder consist of 3 dense layers of size 64, 128, 256 plus LeakyReLU activation function in the encoder part as well as in reversed order in the decoder part. Dropout layers are used with $p = 0.2$ in the two middle layers and $p = 0.3$ else. The output activation function is linear. The aligner is pretrained to mirror the target data using Adam optimizer with MAE as loss function to ensure a good initialization.

- **Dann-based Alignment with Cyclic Supervision (DBACS):** For training ANN-based DBACS the previously on source trained ANN predictor is reused with fixed weights. Both domain discriminators are build in an identical way copied from the DBAM setting:each one has 4 hidden layers with 32, 16, 8, and 4 neurons respectively. LeakyReLU activation function with $alpha = 0.1$ is used, Adam optimizer is used with a learning rate $lr = 0.0001$, $\beta_1 = 0.5$ and $\beta_2 = 0.9$. The aligner set ups are again identical: The encoder has 4 hidden layers with 600, 500, 450 and 350 neurons, respectively. Each layer use LeakyReLU activation function followed by a dropout layer with $p = 0.2$ or $p = 0.3$. The central layer as bottleneck has 250 neurons and is followed by a batch normalization layer. The decoder part is built as a mirrored encoder. Both input and output layers have the size of the number of features that must be aligned. Adam optimizer is used with a learning rate $lr = 0.0001$ and default values for β_1, β_2. Both aligner are pretrained to mirror the target data for aligner F respective source data for aligner G using Adam optimizer with MAE as loss function for improved initialization. The aligner is pretrained with source-source respective target-target pairs.

The training phase happens in iterations, with a ratio of $10 : 1$ between domain discriminators and aligners during the training phase to ensure stability. The predictor related part is weighted by a factor $\alpha = 0.1$. Batch size is set to 128 for both aligners and discriminators. The gradient penalty weight is set to $gp = 10$. Pretraining is done for both aligners as suggested in [69] and as mentioned above. MSE is used as loss function and batch size equals 32. As before, the pretraining phase leads to a better initialization and overall helps convergence.

Second feature-based benchmark methods are defined based on PdM related mainly semiconductor literature that shows their potential for the RUL task at hand:

- **Transfer Component Analysis (TCA):** TCA is an unsupervised feature-based DA method. We select LR and ANN as baseline models. For ANN the network architecture is adopted from the previously on source trained ANN predictor and reused by reducing the input layer size accordingly. All other aspects are kept identical. The implementation is used from python module *transfertools* [?]. Number of latent components are selected as 5, 10 and 20. To speed up the computation a sample subset consisting of 8000 data points is randomly selected from the original training data. First the TCA is applied and then the selected model is retrained using the transformed features. The evaluation is done using transformed source and target test set.

- **Correlation Alignment (CORAL, DeepCORAL):** CORAL is an unsupervised feature-based DA method. The implementation is used from python module *transfertools* [?], LR and ANN are selected as baseline. First CORAL is applied and then the selected model is retrained using the transformed source domain.
For DeepCORAL the implementation is used from python module *adapt* [49], ANN

is selected as baseline using same architecture as described above meaning ANN for task network and the aligner for the encoder part. Supervised training is applied with ADAM as optimizer, MAE loss and batch size 32. The weightage of the CORAL loss is a hyperparameter and tuned via testing the following values $0.01, 0.1, 1, 10, 100$. Unfortunately none of the tested settings lead to a successful training independent of the selected fold. Hence the results are omitted.

- **Subspace Alignment (SA):** SA is an unsupervised feature-based DA method. Implementation is used from python module *adapt* [49], LR and ANN are selected as baseline. For ANN using same architecture as described above meaning ANN predictor for the task network. The number of PCA components is selected equal to the number of input features.

Last, instance-based benchmark methods are defined. Since they are (to the best of our knowledge) not yet applied for DA in a PdM RUL prediction setting, the choice is based on related literature showing their successful implementation to other but related tasks:

- **Nearest Neighbors Weighting (NNW):** NNW is an unsupervised instance-based DA method. The implementation is used from python module *adapt* [49], LR and ANN are selected as baseline. For ANN same task/estimator architecture as described above is used. The algorithm 'brute' (standing for brute-force search or exhaustive search for computing nearest neighbor, see for example Kumar et al. [118]) is used since it gives best results in hyperparameter search and is default for sparse data input. Number of neighbors is set to 5 (searched with 2,5,10,50) and radius to 1 after hyperparameter search (default values). The standard euclidean metric is used (metric='minkowski', p=2). No parallelization is applied.

- **Transfer AdaBoost for Regression (TrAdaBoostR2):** TrAdaBoostR2 is an supervised instance-based DA method. The implementation is used from python module *adapt* [49], LR and ANN are selected as baseline. For ANN same task/estimator architecture as described above is used. Number of estimators are checked via hyperparameter search (10,20,50,100) and set to 100. Default learning rate $lr = 0.01$ is used.

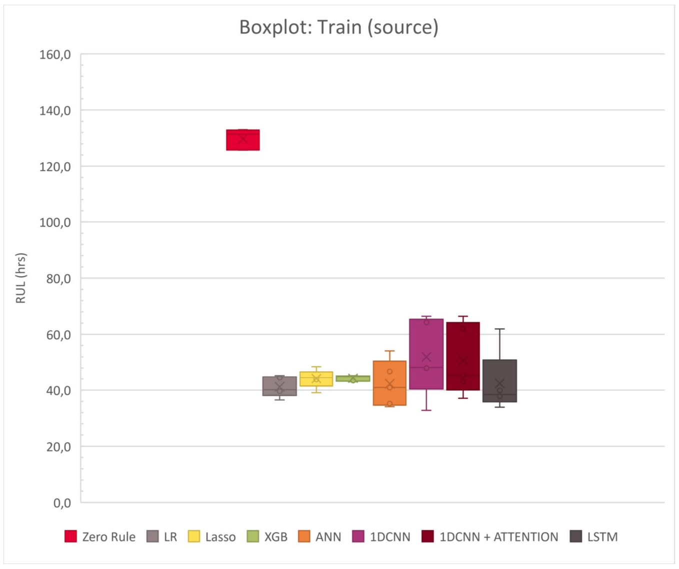
Results and Discussion Part 1: RUL prediction

Evaluation results for the RUL prediction models based on the data set including only lives corresponding to natural deaths are now presented. All non natural death lives are dropped and excluded from training as well as evaluation. Performance scores based on 5-fold CV are presented in Table 9.1. Scores from 5-fold CV for the target domain evaluated with the models only trained by source data are given in addition. Figure 9.8 shows boxplot visualization for all 5 folds for training (left) and testing (right) and all tested models (marked with different colors).

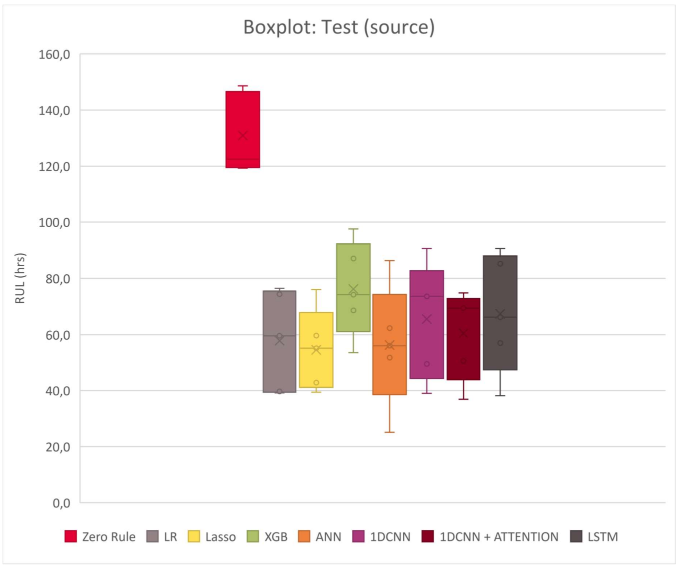
Evaluation of ML algorithms after filtering for natural deaths

	Source		Target	
	Train (MAE)	Test (MAE)	Train (MAE)	Test (MAE)
Zero Rule	129,8	130,9	134,5	134,1
Linear Regr.	**41,2**	57,9	106,7	119,1
Lasso Regr.	44,1	**54,6**	117,8	123,5
XGBoost	44,3	76,2	120,2	115,1
ANN	42,2	56,3	**84,0**	**89,8**
1DCNN	51,9	65,5	100,7	102,1
1DCNN + Attention	50,8	60,5	98,0	92,7
LSTM	42,4	67,4	92,4	91,9

Table 9.1: Evaluation of ML algorithms after removing non natural deaths. Errors are measured in hours (hrs). Models are trained on training data from source and tested afterwards. Data from target is evaluated using source dedicated models without retraining. All scores are mean values for 5-fold CV.

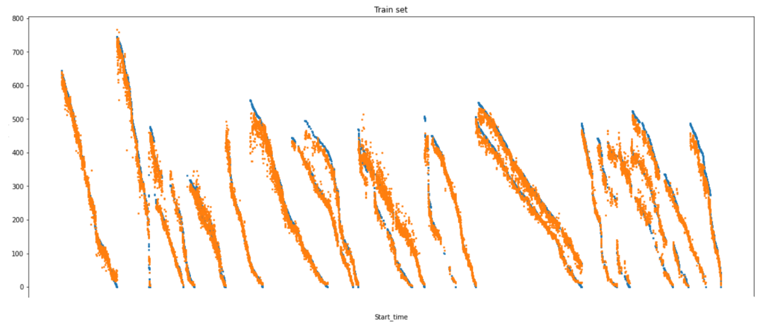


(a)

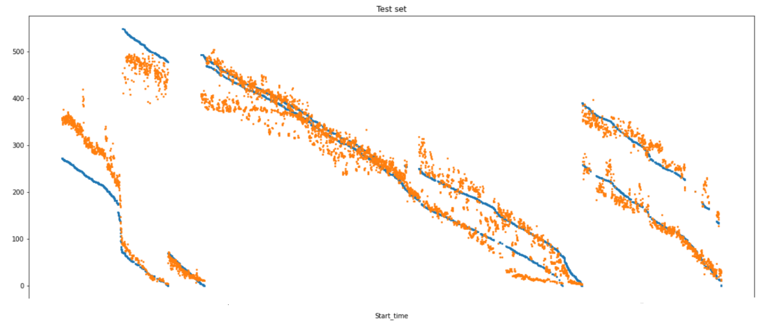


(b)

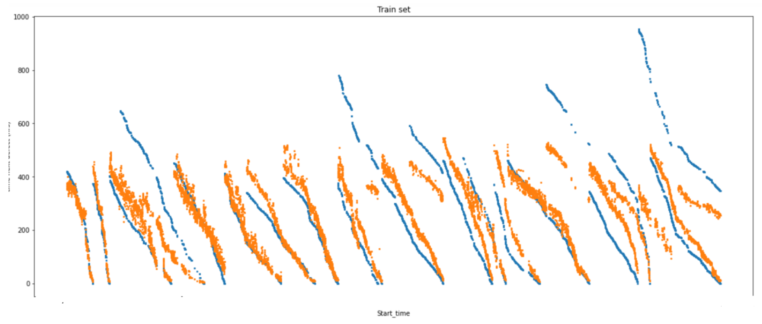
Figure 9.8: Boxplot of 5-fold cross validation prediction performance of dedicated source models. Boxplot representation of prediction performance for all used models and all 5 folds. Different colors represent different underlying models. (a) presents results for training data and (b) results for test data. Errors are measured in hours (hrs).



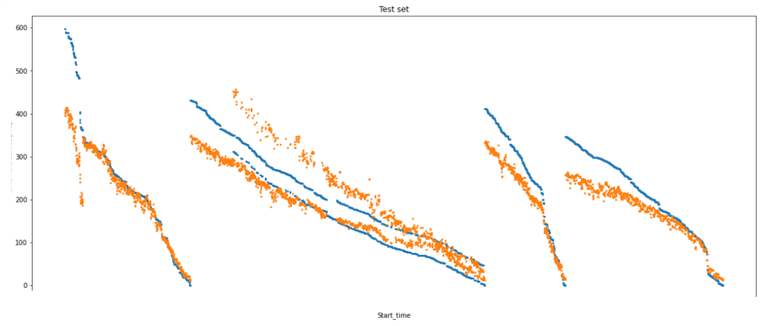
(a) True versus predicted values for source training data



(b) True versus predicted values for source testing data



(c) True versus predicted values for target training data



(d) True versus predicted values for target testing data

Figure 9.9: True vs. predicted scatter plot for dedicated ANN source model. Scatter plots representation of RUL prediction true versus predicted for ANN model and best performing fold. (a) presents results for source training data and (b) results for source test data. (c) presents results for target training data and (d) results for target test data when evaluated with the dedicated models only trained on source.The other models are presented in Figure 9.10. RUL is measured in hours (hrs).

All models are beating the Zero Rule score, while all of them show high test variability towards different data splits, see Figure 9.8. For more complex DL models, higher variations can be seen already within the training data compared to traditional ML methods.

The performance of the dedicated source models for the target data set is not sufficient for all models as presented in Table 9.1, nevertheless DL models are able to generalize better for target data than the traditional ML models, hence would be a recommended choice for increasing complexity in the data.

Figure 9.10 presents scatter plots for true versus predicted RUL for source training data (left column) and source test data (right column). For Linear Regression and Lasso an overall shift of the RUL prediction is visible, meaning the equipment health is overestimated. Both models are also allowing negative predictions that of course do not make much sense. XGBoost seems to have a higher error at the beginning of each life causing an higher overall error while towards the end the predicted values are very close to the true value. Its tendency to underestimate the RUL seems more fitting if a more conservative maintenance strategy is preferred.

ANN is the second best (for train and test) performing method but it shows most stable performance over all folds and also the best transfer performance when applied to the target data. Hence it is later the choice for the fixed RUL prediction model for domain adaptation using DBAM. ANN RUL scatter plots are presented in Figure 9.9.

The usage of the ANN model on target data shows a poor estimate especially for longer lives and a tendency to overestimate RUL respective equipment health. The models based on the reshaped time series have comparable performances, with LSTM and 1DCNN + Attention layer slightly better than standalone 1DCNN. All time series based models show rainfall shaped noise within their predictions.

Results and Discussion Part 2

Evaluation results for the domain adaptation models based on the data set including only lives corresponding to natural deaths are now presented. Here, DBAM and DBACS are applied and compared. Other domain adaptation benchmarks are evaluated and compared in section 9.4: Imperfect data. Equipment group 1 is selected as fixed source domain since it contains most of the lives and higher number of equipment. We refer to the same source train-test split used to train the corresponding predictor model in Result and Discussion Part 1. Data down sampling is performed of the source data to align total number of samples between source and target.

First, one takes a look at lower and upper bound. Lower bound are the results when using dedicated models only trained and used for pre-selected equipment. It is the best possible performance and all transfer learning approaches try to come as close as possible to the lower bound. This approach faces some limitations:

1. high effort in maintaining multiple models including generate evaluation, analyzing results, defining monitoring kpis, updating the models besides others;
2. only applicable when enough data and especially labels are available;
3. no direct comparison between equipment, RUL, hence available prediction models is possible.

The upper bound is the direct transfer of a trained model to unseen data coming from (eventually different) equipment data not used for training. If the unseen data does differ, this direct model transfer has limited success. Lower and upper bound are both presented in Table 9.2.

Upper and lower bounds of dedicated ML algorithms

	Target lower bound		Target Upper bound	
	Train (MAE)	Test (MAE)	Train (MAE)	Test (MAE)
Zero Rule	133,7	135,3	134,5	134,1
Linear Regr.	61,0	83,0	106,7	119,1
ANN	**53,8**	**64,1**	**84,0**	**89,8**

Table 9.2: Evaluation of dedicated ML algorithms after removing non natural deaths. Errors are measured in hours (hrs). For the upper bound (see also Tab. 9.1) models are trained only on source and tested on target data afterwards. For the lower bound dedicated models are trained on target training data and evaluated afterwards with test target data. All scores are mean values for 5-fold CV.

Depending on the adaptation approach applied a) one either has the model directly involved in the training where both source and target data gets mapped into a latent space, b) after the domain alignment for both domains has happened a model is trained on both aligned source and aligned target data, or c) the dedicated model is fixed and the alignment happens only for the target domain such that the existing model can be reused.

Next, results for DBAM and DBACS are presented. DBACS is best suitable since the unpaired setting is fitting for the PdM task at hand from theoretical perspective. The evaluation results are presented in table 9.3 for RUL prediction accuracy using MAE and 9.4 for FID comparison. An overview of all folds and CV deviations for DBAM and DBACS is given in Figure 9.18.

Evaluation of DBAM and DBACS for aligned target

	Target domain	
	Train (MAE)	Test (MAE)
Lower bound	53,8	64,1
Upper Bound	84,0	89,8
DBAM	27,8	78,7
DBACS	**16,9**	**78,6**

Table 9.3: Source and aligned target data training and test MAE average over 5-fold CV. RUL prediction models are trained only on source data and evaluated on test data. Target data is mapped to source domain using trained aligner F from DBAM respective DBACS and evaluated after the mapping using the RUL prediction model trained on source.

FID evaluation of inner and outer domain distance

	Source	Target
Inner Domain	0,4	0,4
	Train	Test
Outer Domain	2,0	2,3
DBAM	**0,2**	**0,9**
DBACS	0,3	0,9

Table 9.4: FID evaluation of inner and outer domain distance on train and test data before and after the alignment on average over all 5-fold training runs for source and target.

Scatter plots including RUL prediction true versus predicted for upper limit (no alignment) from DBAM and DBACS are depicted in Figure 9.11.

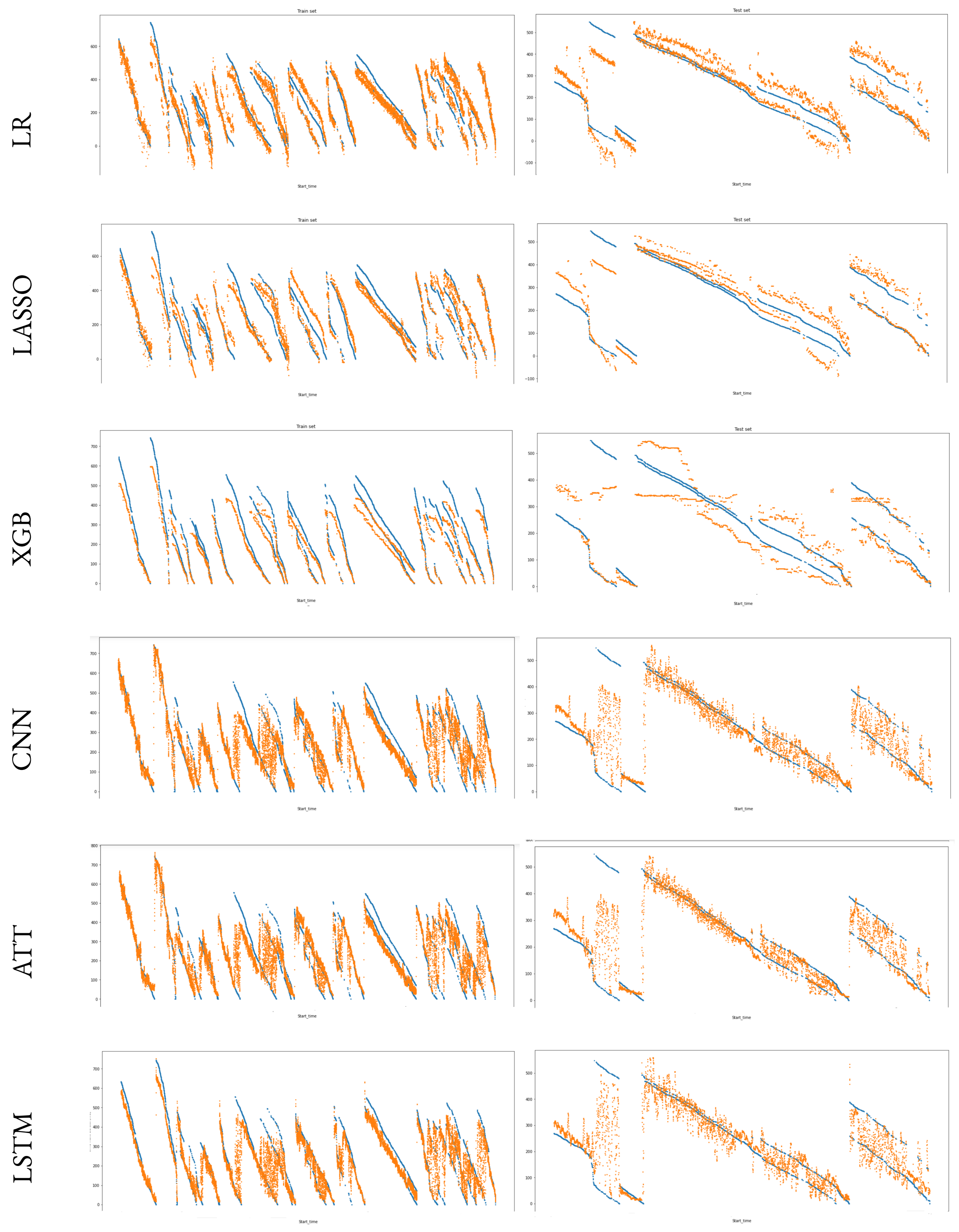


Figure 9.10: True vs. predicted scatter plot for dedicated benchmark source models. Scatter plots representation of RUL prediction true versus predicted for all used models and best performing fold. Left column presents results for training data and right column results for test data. The models are presented in the following order: Linear Regression, Lasso Regression, XGBoost, 1DCNN, 1DCNN + Attention, LSTM. ANN is presented in Figure 9.9. RUL is measured in hours (hrs).

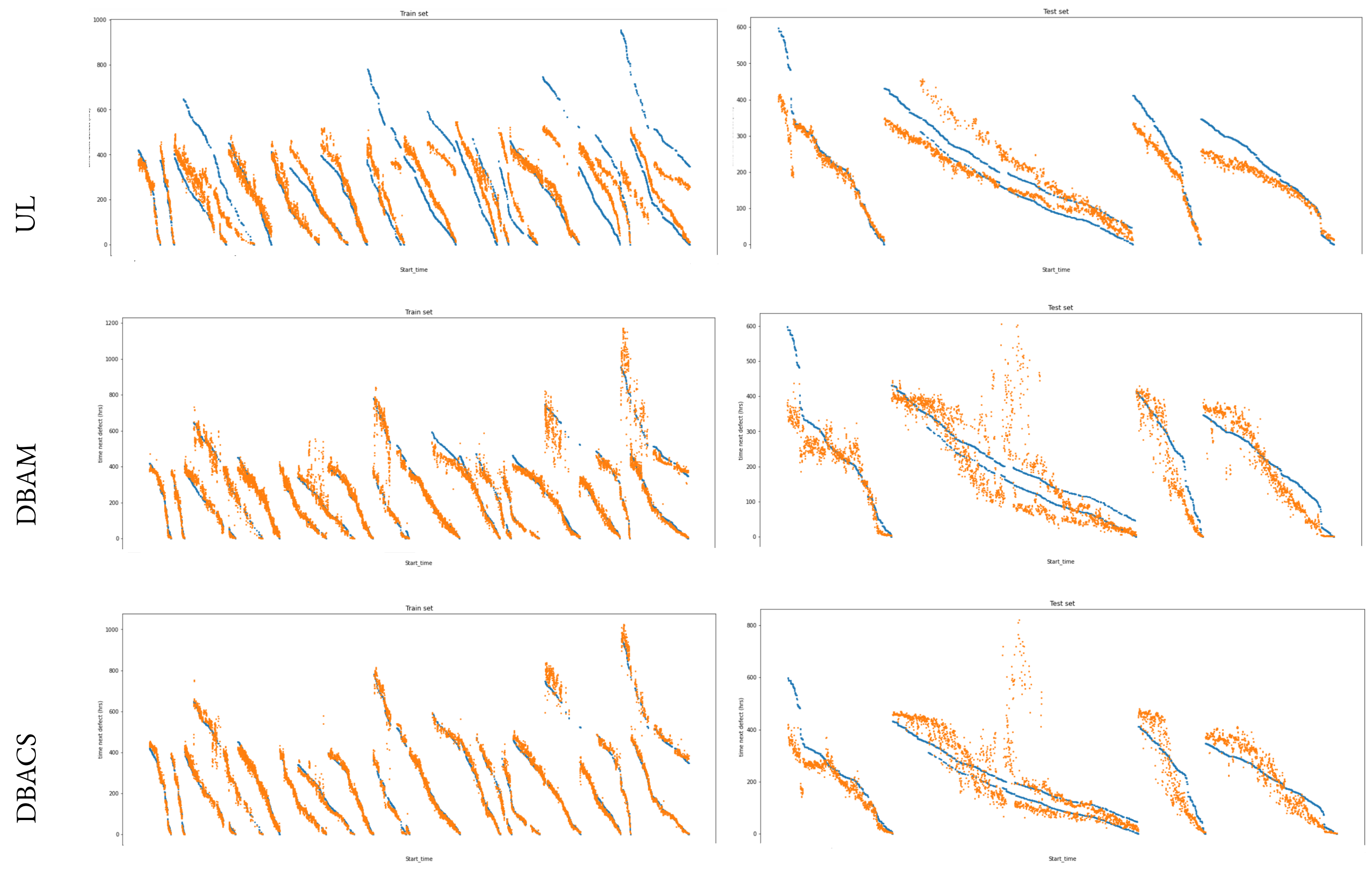


Figure 9.11: True versus predicted scatter plots for ANN DABM and DBACS before and after alignment. Scatter plots representation of RUL prediction true versus predicted for ANN before and after alignment with DBAM and DBACS for best performing fold. Left column presents results for training target data and right column results for training test data. The models are presented in the following order: ANN only trained on source, ANN aligned using DBAM, ANN aligned using DBACS. ANN data is also presented in Figure 9.9. RUL is measured in hours (hrs).

The prediction MAE is reduced, bringing the MAE down to usable levels. FID scores confirm the alignment. Distribution alignment is also supported by T-SNE plots shown in Figure 9.12. T-SNE gives a visual comparison of the two domains based on KL-divergence. All plots are done using the aligner F to map all target features to the source feature space, but in different stages of the alignment process: Top row shows features without usage of the aligner, bottom row shows after the DA with DBAM is done. An increased domain overlap is visible.

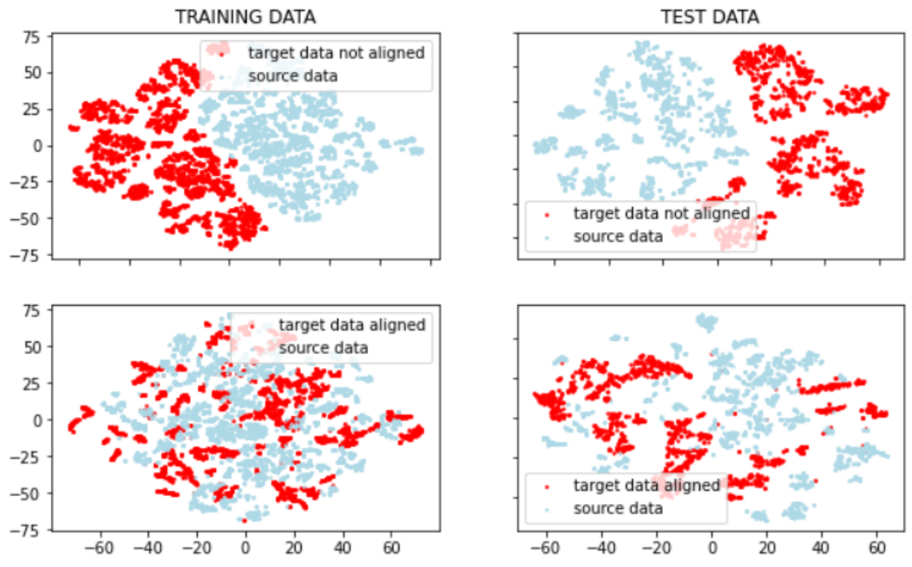


Figure 9.12: T-SNE visualization before and after alignment with DBACS. Graphical t-SNE representation of source and target domain in different stages of the alignment process: Top row shows features without usage of the aligner, bottom row shows after the DA with DBACS is done. The source is colored in blue and contains data from equipment group 1, the target is colored red and contains data from the equipment group 2. The axes are dimensionless. The effect of the adaptation of the input features after DBACS is applied during training: The adaptation brings the distributions of source and target domain closer and target overlaps source domain after the alignment.

The advantage of DBAM respective DBACS is used to compare input features before and after the alignment. A visualization of the aligner output is presented in Figure 9.13. The aligner set up enables interpretability as well as a comparison between source and aligned target features. The histogram of example features from test data before and after the alignment are shown for example features, where source data is plotted in red and target data in blue and the aligned target data in black. The alignment is visible through different kind of shifts: the shape and the scale of the histogram is adapted.

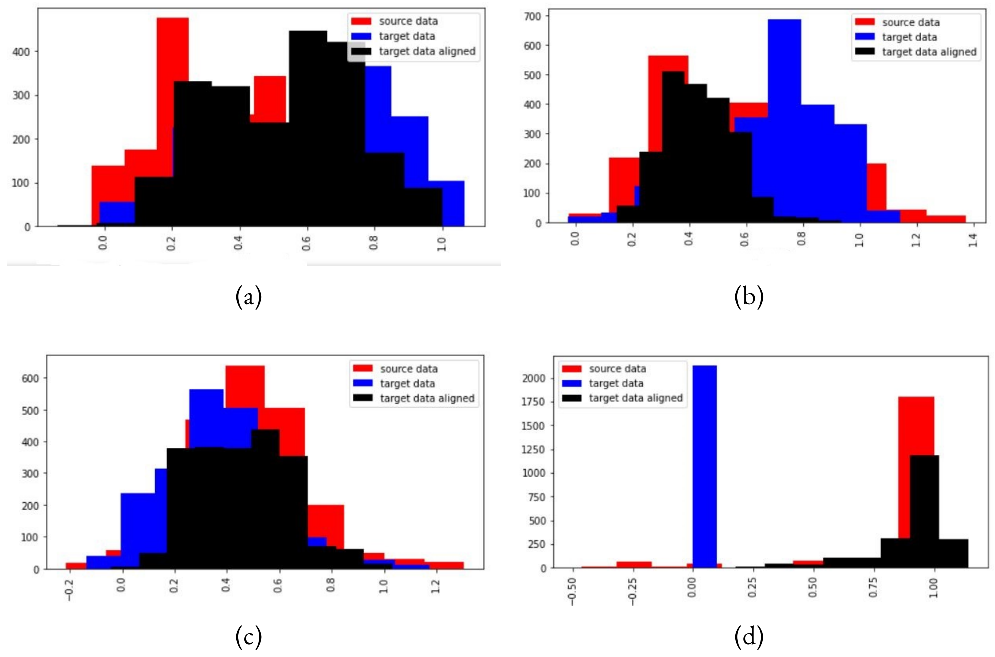


Figure 9.13: Aligner histogram for feature visualization before and after alignment with DBACS. Graphical representation of the histogram of features respective keynumber of both source and target and before and after the alignment with DBACS. A shift in shape and scale of the histogram is visible. The source is marked red, the target blue and the aligned target black. The graph shows test data from one one fold.

9.4 Imperfect data

In this section we present evaluation results from selected benchmark models and discuss their performance. A main focus for this discussion is the data itself since the interpretation of the performance error is not always straight forward and often process or equipment expert revision is recommended. The given explanations shall point out different modeling aspects crucial for a successful PdM prediction system, decision making and roll out.

Feature-based benchmark models

Feature-based methods are already successfully applied to PdM as discussed in the literature section above. Introducing fast and easy to interpret DA based on classical methods can speed up the transfer, support acceptance and enable fab wide model scalability. Table 9.5 shows the average 5-fold CV results for all selected benchmark methods with LR as baseline method.

Evaluation of feature-based domain adaptation on linear regression model

	Source domain		Target domain	
	Train (MAE)	Test (MAE)	Train (MAE)	Test (MAE)
Upper bound	41,2	57,9	106,7	119,1
TCA 5	72,4	79,7	117,0	113,9
TCA 10	57,0	73,2	154,5	162,9
TCA 20	48,6	75,4	228,6	241,2
CORAL	40,4	62,0	103,1	95,5
SA	41,2	57,9	101,0	104,8

Table 9.5: Feature-based domain adaptation for source and target data. Training and test MAE average over 5-fold CV is given. RUL prediction models are trained both on features generated by mapped source and target data and evaluated on new features from mapped test data. Task model is a linear regression.

Table 9.5 shows that CORAL slightly improves the target accuracy based on MAE over all lives while keeping the source accuracy comparably low. Figure 9.14 supports those improvements visually for CORAL but also shows some improved fits for TCA, especially TCA 5 with 5 latent features and TCA 10 with 10 latent features, for target lives. For avoidance of unpredicted failure and machine breakdowns, the samples at the end of a life are more important and are getting special attendance when evaluating a model accuracy. Therefore even if the numbers for SA look good, it still underestimates RUL over the whole period of a life and especially towards the end. TCA (5,10 and 20) are able to give good predictions towards the end of a life. Nevertheless, they have a big prediction offset at the beginning of almost all lives.

For the results of benchmark models applied together with ANN the already existing ANN based prediction model is reused. Table 9.6 shows the average 5-fold CV results for all selected benchmark methods with ANN.

Evaluation of feature-based domain adaptation on ANN model

	Source domain		Target domain	
	Train (MAE)	Test (MAE)	Train (MAE)	Test (MAE)
Upper bound	42,2	56,3	84,0	89,8
TCA 5	84,3	83,0	102,4	104,2
TCA 10	75,3	77,8	113,8	95,8
TCA 20	69,1	74,5	104,9	96,8
CORAL	97,0	108,0	133,7	136,3
SA	71,9	83,2	97,1	94,0

Table 9.6: Feature-based domain adaptation for source and target data. Training and test MAE average over 5-fold CV is given. RUL prediction models are trained both on features generated by mapped source and target data and evaluated on new features from mapped test data. Task model is ANN.

None of the methods are able to improve the performance compared to the upper bound. Figure 9.14 shows the corresponding scatter plots. It can be seen that the methods with dimension reduction suffer from information loss leading to noise introduction to the predictions and hence higher source test errors. SA performs well from visual perspective, fitting all target lives except one (that leads to the high MAE) very good. CORAL shows a tendency towards overestimating RUL that is especially critical when a conservative maintenance policy is applied. None of the benchmark models are able to achieve better or even comparable results to DBAM and DBACS (see again Table 9.3). An overview of all folds and CV deviations for feature-based DA models is given in Figure 9.18.

Instance-based benchmark models

Instance-based models like ensemble learning and sample weighting show promising results when applied to industrial settings as discussed in the literature section. Table 9.7 shows the average 5-fold CV results for two selected benchmark methods NNW and TrAdaBoostR2 with LR as baseline method.

Evaluation of instance-based domain adaptation on linear regression model

	Source domain		Target domain	
	Train (MAE)	Test (MAE)	Train (MAE)	Test (MAE)
Lower bound	41,2	57,9	106,7	119,1
NNW	67,1	76,7	132,4	137,5
TrAdaBoostR2	52,6	57,7	60,9	70,9

Table 9.7: Instance-based domain adaptation for source and target data. Training and test MAE average over 5-fold CV is given. RUL prediction models are trained both on weighted samples of source and target data. Task model is a linear regression.

Table 9.7 shows that TrAdaBoostR2 is significantly improving the target accuracy based on MAE over all lives while keeping the source accuracy stable. Figure 9.16 supports those improvements visually for TrAdaBoostR2. All lives of the target test set are adapted if necessary and improved accuracy are visible especially towards the end of a life that is the most critical phase. While NNW is decreasing accuracy for both source and test set, visually it can be noticed that despite adding noise in the source test prediction, some target test lives show an overall improved fit.

For the results of benchmark models applied together with ANN the already existing prediction model is reused. Table 9.8 shows the average 5-fold CV results for the two selected benchmark methods trained together with ANN.

Evaluation of instance-based domain adaptation on ANN model

	Source domain		Target domain	
	Train (MAE)	Test (MAE)	Train (MAE)	Test (MAE)
Lower bound	41,2	57,9	106,7	119,1
NNW	41,3	56,9	87,5	92,7
TrAdaBoostR2	47,9	59,0	74,8	79,7

Table 9.8: Instance-based domain adaptation for source and target data. Training and test MAE average over 5-fold CV is given. RUL prediction models are trained both on weighted samples of source and target data. Task model is ANN.

Both NNW as well as TrAdaBoost shows an improved target train and test accuracy. For the visualization in Figure 9.17 fold2 is chosen that does not show much differences when compared to the upper bound. Nevertheless, for TrAdaBoost other folds show improvements and a significantly decreased variation between the folds. Since for NNW, the results vary between folds stronger than before and show higher variation, the stability of NNW is in question for ANN. An overview of all folds and CV deviations for instance-based DA models is given in Figure 9.18.

Conclusion

PdM is a very complex task with a large set of influential factors hence modeling decisions that need to be made including the ones mentioned in [209]:

There exists two type of sensors:

(i) input sensor;

(ii) feedback sensor.

While feedback sensors are the one that are not controlled by recipe and therefore most interesting to hint toward changed conditions and degradation, the input whose values are kept at a certain level, can also be of high interest when it comes to the different recipe settings and influences. Hence keeping both feedback but also input can provide for example info about the degradation speed of specific feature settings. But it also needs specific attention in their preprocessing meaning removal of the recipe effect as shown earlier to avoid misinterpretations of jumps, peaks or drops due to a recipe change. A powerful feature selection can also enable usage of simpler, lower dimensional methods, as presented for example by Lu et al. [138].

There exists more than one function to define labels. This leads to different modeling assumptions depending if degradation start is put right at the beginning of a live or a separate step is necessary for its detection; meaning defining a constant RUL at the beginning of each live for a certain time span (that also needs to be defined). There are three ways to accommodate to this scenario:

1. using a piecewise affine linear target function as described in Figure 9.1;
2. upstream a classifier before the RUL prediction model that detects start time of degradation as shown for example in [218] (as alternative you could create a input feature that takes care of this [96]);
3. weighting your samples accordingly so that the samples closer to the end of a live are predicted with best possible accuracy while some error at the beginning can be tolerated.

The latter fits to the idea to apply instance-based DA methods to focus on a best possible transfer towards the end of the target lives.

More complexity is added by different failure mode and the task to distinguish them based on the data at hand. Run-to-fail is not a common practice hence lives are corrupted. Therefore, adding more lives means adding additional equipment to the data that could lead again to new phenomena.

Last but not least the decision if a RUL PdM model is acceptable and scalable or not is mainly based on use case focus and specifications: For uptime and utilization focused use case the ratio between process duration, process time and RUL is a key factor. For a resource planning use case RUL is an additional input factor that needs to be considered besides operator availability, machine utilization, order fulfillment, dedications, spare part inventory and more. Under certain circumstances a overestimation of RUL can be tolerated. For a avoidance of destruction use case a very conservative approach is preferable. Run to fail and further destruction of product avoidance has highest priority and underestimating RUL is acceptable for a prediction model.

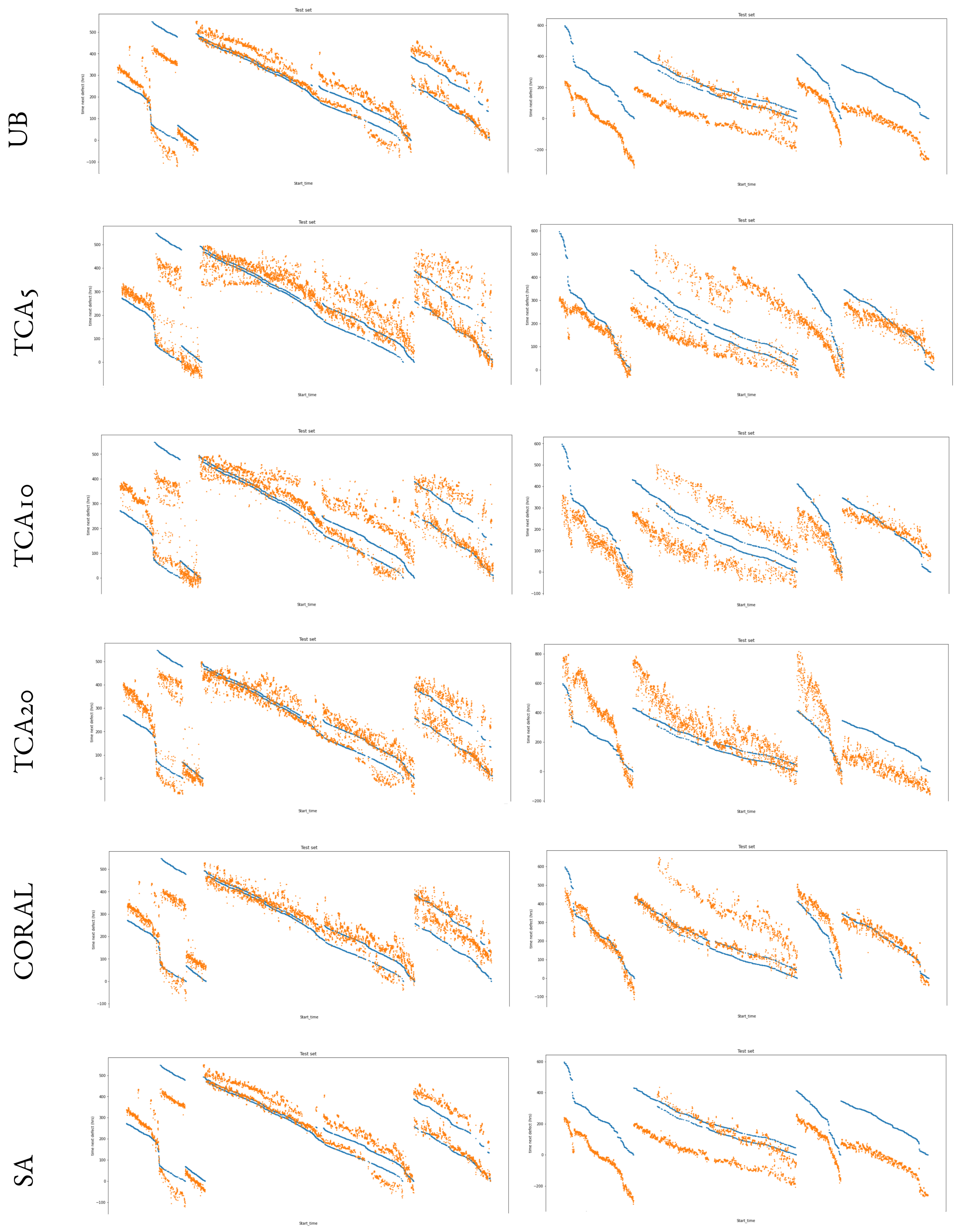


Figure 9.14: True vs. predicted scatter plots for feature-based domain adaptation models with LR before and after alignment. Scatter plots representation of RUL prediction true versus predicted for LR before and after alignment with feature-based benchmark models for best performing fold. Left column presents results for source test data and right column results for target test data. The models are presented in the following order: Upper bound LR without alignment, LR aligned using TCA with latent space dimension 5, 10, and 20, LR aligned using CORAL and LR aligned using SA. RUL is measured in hours (hrs).

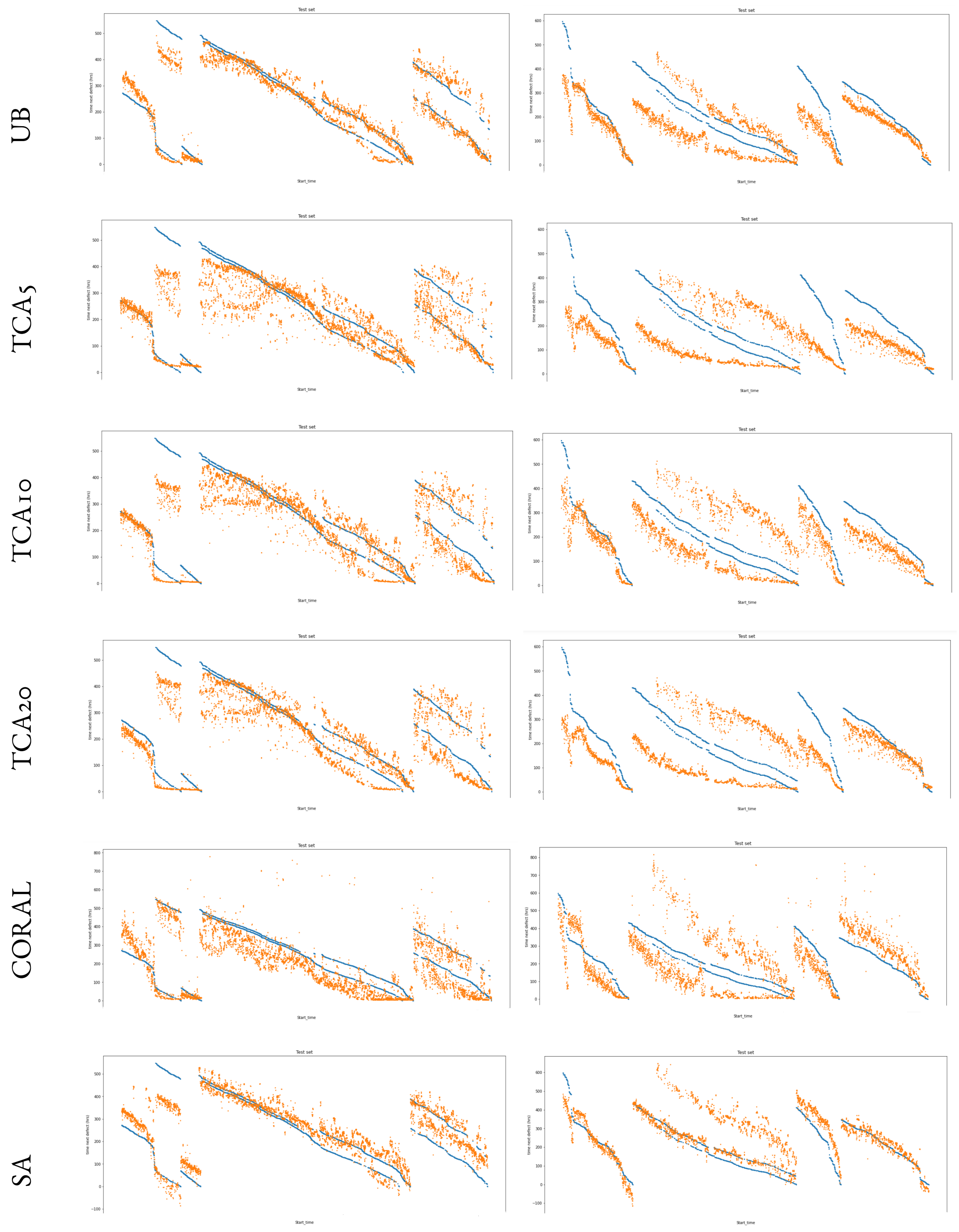


Figure 9.15: True vs. predicted scatter plots for feature-based domain adaptation models with ANN before and after alignment. Scatter plots representation of RUL prediction true versus predicted for ANN before and after alignment with feature-based benchmark models for best performing fold. Left column presents results for source test data and right column results for target test data. The models are presented in the following order: Upper bound ANN without alignment, ANN aligned using TCA with latent space dimension 5, 10, and 20, ANN aligned using DeepCORAL and ANN aligned using SA. RUL is measured in hours (hrs).

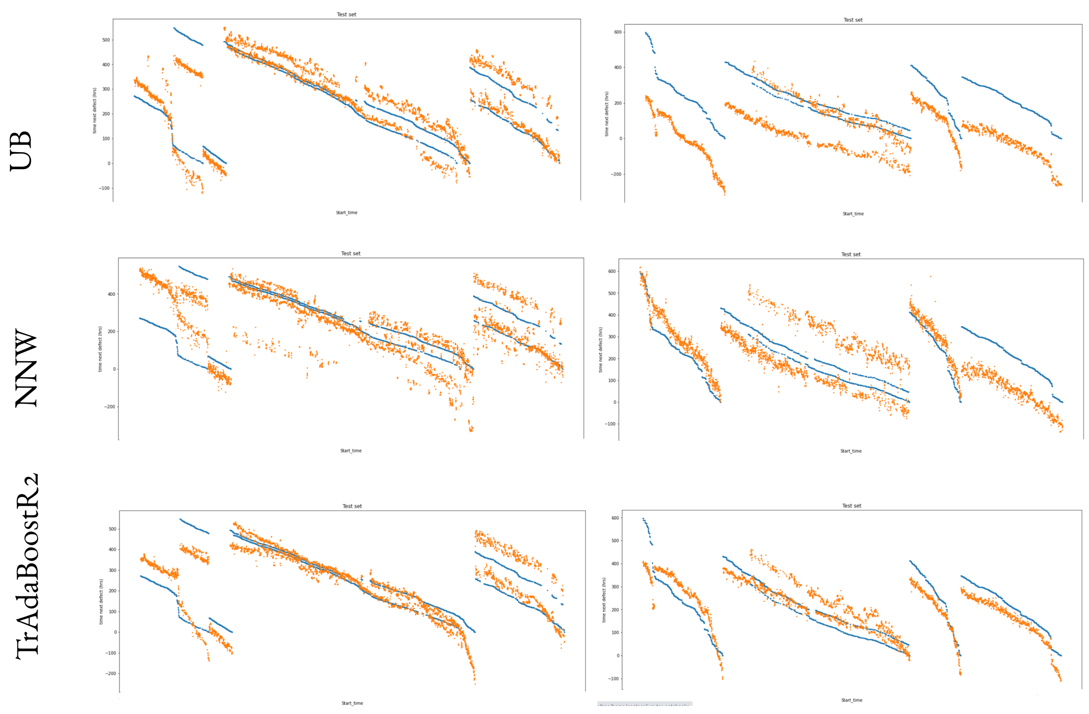


Figure 9.16: True vs. predicted scatter plots for instance-based domain adaptation models with LR before and after alignment. Scatter plots representation of RUL prediction true versus predicted for LR before and after alignment with instance-based benchmark models for best performing fold. Left column presents results for source test data and right column results for target test data. The models are presented in the following order: Upper bound LR without alignment, LR aligned using NNW and LR aligned using TrAdaBoostR2. RUL is measured in hours (hrs).

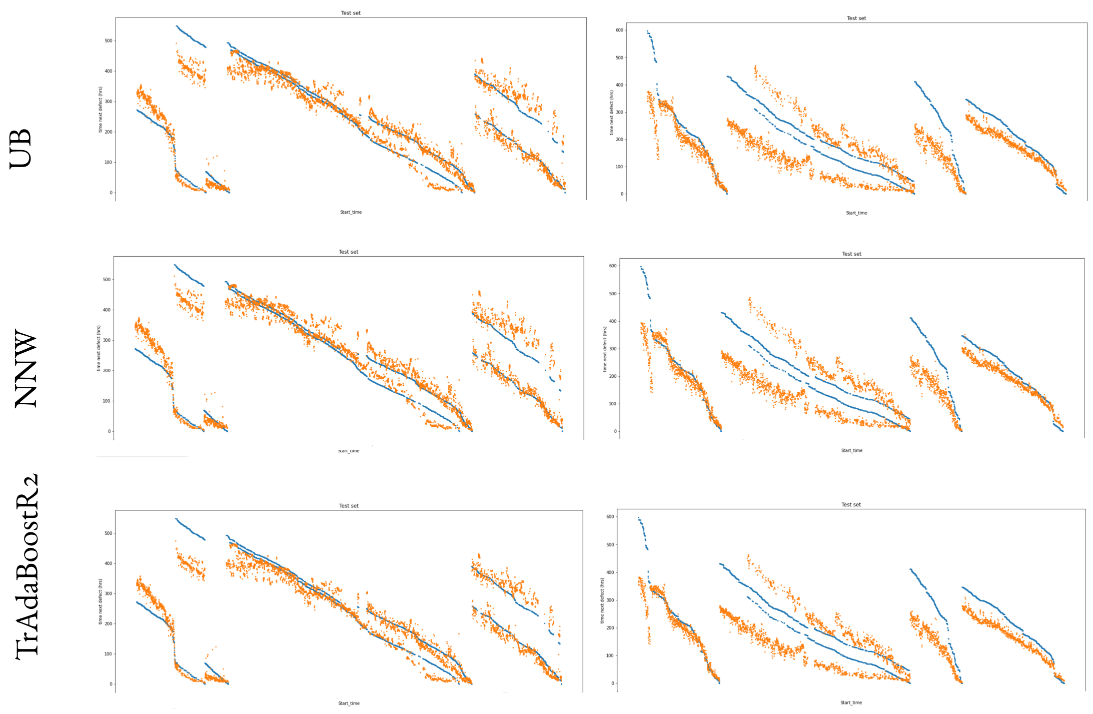
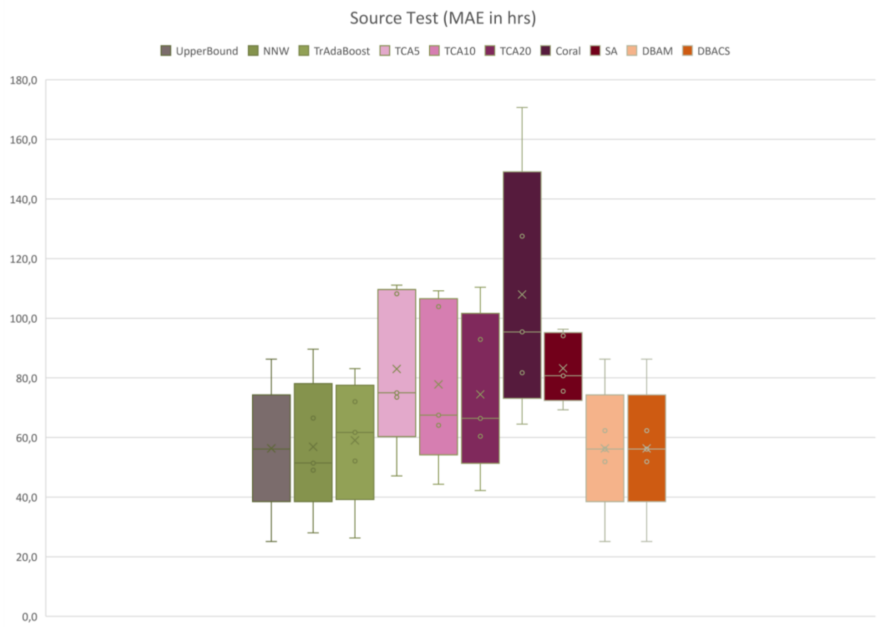
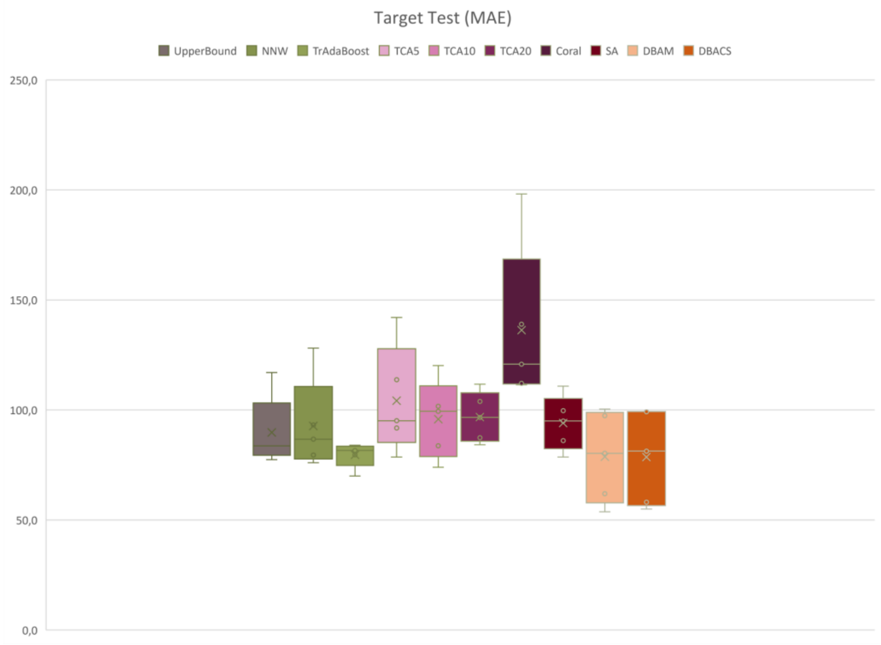


Figure 9.17: True vs. predicted scatter plots for instance-based domain adaptation models with ANN before and after alignment. Scatter plots representation of RUL prediction true versus predicted for ANN before and after alignment with instance-based benchmark models for best performing fold. Left column presents results for source test data and right column results for target test data. The models are presented in the following order: Upper bound ANN without alignment, ANN aligned using NNW and ANN aligned using TrAdaBoostR2. RUL is measured in hours (hrs).



(a)



(b)

Figure 9.18: 5 fold cross validation model error for prediction including all DA models after alignment. 5 fold cross validation results to present inner model variance and overall stability of all presented domain adaptation models. Panel (a) shows test results of the source test domain and panel (b) the test results of the target test domain after domain adaptation is applied.

10
Defect Classification (DC)

Detecting and classification of different defects on the wafer is one of the most important mechanism for assuring highest quality and early as possible identification of faulty or malfunctioning products and hence increased overall yield. Starting from rule-based systems as described in Chou et al. [41] or template respective golden wafer based detection as in Shankar and Zhong [181]. With the introduction of convolutional neural networks (for an overview see for example Albawi et al. [2]) including powerful architectures able to classify complex data sets including versatile classes, literature shows tremendous success in fully automated defect detection and classification based on transfer learning (Yosinski et al. [226]) also in industrial settings (Zhu et al. [240]).

The results presented in this chapter are obtained in collaboration with University of Bologna, Department of Computer Science and Engineering, Prof. Dr. Samuele Salti and Adrian Poniatowski. A manuscript submitted for publication is not yet published.

10.1 Introduction

While human inspection/classification for process control in semiconductor manufacturing was heavily employed in the past, automatic modules are nowadays particularly appealing for both performance and costs related reasons. The availability of pretrained models and the application of transfer learning methods has tremendously changed and improved the field of computer vision over the last decade. By applying such tools, industry 4.0 applications like defect classification, holds great success even in the presence of limited amount of available data. Besides machine learning-based solutions to automate such procedure ([4, 80]), semiconductor manufacturing heavily leveraged the advancement of deep learning (DL) in the field of computer vision. Thanks to Convolutional Neural Networks (CNN) and to many algorithmic advancements, automatic defect classification has become cost-effective and widely adopted to bring classification performances to the next level.

Nevertheless, in order to enable fab wide usage of those highly automated classification models, deployed methods need to be transferable and applicable when confronted with new optical properties and different image structures aroused from technological diversity as well as reoccurring production steps in different stages of the manufacturing process, see Figure 2.6 and depicted defect images. Hence, this existing high data complexity demands more effort and research in the field of scalability when it comes to limited or missing labels and diverse and complex image content. This aspect of existing differences in data distributions, plus given the abundance of unlabeled data in modern manufacturing environment, represents a challenge but also an opportunity to exploit the full potential of all collected data hence fab wide therefore scalable installation of automated defect classification.

Here, we address those issues and exploit transfer learning methods unified with un- and semi-supervised learning. The upcoming sections are structured in the following way: First use case specific literature is covered. Then, the use case defect classification for SEM images with diverse background pattern is introduced. Data preparation followed by the experimental design is explained. The focus shifts towards benchmark models with pseudo-labeling and AdaMatch. Then, DBACS set up as well as metrics and losses are defined. After presenting model details and hyperparameter choices, the results are presented and discussed. The use case is closed with a discussion on influence of data augmentation as well as distribution alignment within AdaMatch inspired by [153].

10.2 Literature

Literature about image classification in semiconductor manufacturing mainly covers two areas:

- *image defect classification*, where images, like SEM images, are taken at relevant spots in the wafer and classified into a-priori known defects. For a visualization of that process see SEM image in Figure 2.6;

- *wafer level/wafer-map defect pattern classification*, describes a map of the wafer where pre-identified anomalies/potential defects have been already identified and the task is to classify such maps w.r.t. known defect patterns (see defect map in Figure 2.6).

Some research focusing on wafer-level *wafer map* defect classification and defect pattern classification were published in the last recent years, eventually also driven by the wafer map dataset WM-811k made publicly available by Wu et al. [217]. In Nakazawa and Kulkarni [151] binary codes for wafer maps were generated by employing CNN and used for wafer-level defect pattern classification. Kyeong and Kim [119] builds an individual CNN-based classifier for each defect class: if two defect patterns coexist, two classification models are expected to detect them, while the other models won't notice them. In Yu et al. [229] an architecture with a 8-layer CNN model to inspect wafer map defect and a 13-layer model to classify defect patterns is proposed, while in Jin et al. [104] the authors a DBSCAN-like method that allows to perform outlier detection and defect cluster pattern extraction at the same time. In Saqlain et al. [174] the authors proposed a voting ensemble classifiers with multi-types features (as described by the authors, density-, geometry-, and radon-based features) to identify wafer map defect patterns. Hsu and Chien [94] confirms the power of ensemble learning using CNN for complex defect wafer map classification. In Santos et al. [173] authors showed how data-driven deep generative model can outperform classical approaches if enough data is available.

In Yu et al. [228] a semi-supervised deep transfer learning algorithm is proposed: the presented approach uses CNN to extract transferable features of wafer maps and then introduces a multilayer domain adaptation and pseudo-label learning block based on the generative adversarial network (GAN); with this procedure the authors are able to reduce the distribution discrepancy and the among-class distance of the transferable features. Lu et al. [139] presents another GAN inspired approach using pix2pix (presented by Isola et al. [99]) to deal with class imbalance in defect inspection in industrial settings. Another semi-supervised approach has been presented in Kong and Ni [116] where a ladder network and a variational autoencoder are adopted to classify wafer bin maps: in the proposed approach, the authors also exploit active learning and pseudo labeling. In Shim et al. [184] another active learning framework is proposed by the authors to allow the defect pattern classification system to improve over time thanks to the availability of new samples.

With regards to the first listed category image defect classification, literature has been typically focused on SEM images; the introduction of deep learning based approaches have helped to drastically reduced the amount of time spent by human operators in manually tagging images Imoto et al. [97]. It also shows that CNN-base transfer learning methods can classify microscopic defect images with high accuracy hence shows the efficacy of CNN applied to chip-level defect image data. In Schlosser et al. [177] authors proposed a stacked hybrid Convolutional Neural Networks (CNN) that exploits modern approach of attention mechanisms. Also in this case, CNN and deep learning approaches are the typical choice Cheon et al. [38], O'Leary et al. [155] for performing the classification task. Cheon et al. [38] additionally demonstrates that a single convolutional neural network model can extract effective features for defect classification without using additional feature extraction algorithm. Two publications exploiting pseudo-labeling in the are of defect classification are Liu et al. [135] and Li et al. [130]. However only few works apply deep architectures in this context; while, for example, models like InceptionV3 by Szegedy et al. [196] are tested by Lee and Lee [124] for similar tasks.

10.3 Defect Classification (DC) for SEM Images with diverse Background Pattern

Data Preparation

The data described in this section and used in the experiments is composed by SEM images. An introduction into metrology and especially defect control is given in Part I, Chapter 2, Section 2.3,

The data includes different predefined *product technologies* that are characterized by different backgrounds on which the defects lie. Only defects are selected that can occur on multiple product technologies due to overlapping production steps. Therefore a subset of the classes is chosen and occurring defects are merged into two final classes:

- points;
- particles.

A particle defect is characterized by some kind of foreign particle, like for example dust, that is present on current surface layer of the wafer. Some particles are presented in the first row in Figure 10.1. Point defects are impurity of the layer material and can disturb physical properties like for example electrical conductivity of the wafer. Some points are presented in the second row in Figure 10.1.

Three different product technologies presenting 3 different image backgrounds are selected based on availability of training data. 3 domains are defined based on the corresponding product technologies: images in *domain 0* present a plain, unstructured background; images in *domain 3* have horizontal lines in the background; images in *domain 8* exhibit square-like shapes of different size and location in the background. Example images of defect classes including diverse background pattern are presented in Figure 10.1.

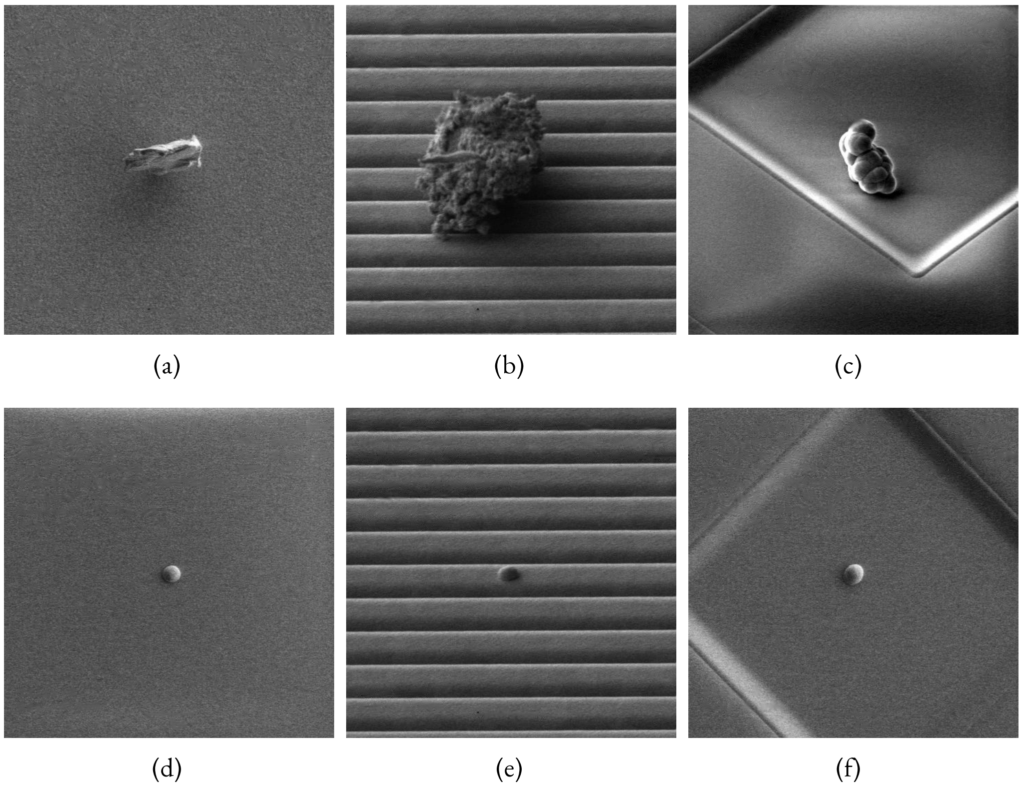


Figure 10.1: Visualization of defect images. Image examples of particle a), c), e) (left column) and point b), d), f) (right column) defects for each of the domains. First row shows domain 0 with plain, unstructured background, second row shows domain 3 with horizontal lines in the background and third row shows domain 8 with squared structures in the background.

The number of images per class and domain are shown in Figure 10.2. Class distribution is balanced for domain 0 and 3 but imbalanced for domain 8.

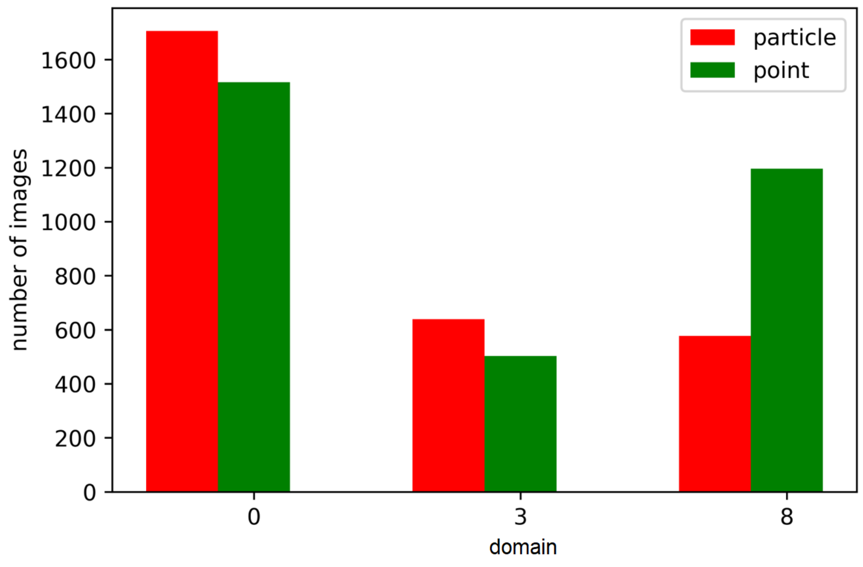


Figure 10.2: The number of images per class and domain. Class distribution is balanced for domain 0 and 3 but imbalanced for domain 8.

We are going to simulate unsupervised domain adaptation (UDA) as well as semi-supervised domain adaptation (SSDA). To simulate an UDA scenario a source domain is selected and assumed to be fully supervised hence all available labels are considered during training. One of the other domains is selected as target domain and all available labels are ignored and not used for training. For example, for UDA from domain 0 to 3, the labels of the data from domain 0 are considered, while those from domain 3 are ignored. In the SSDA scenario a predefined percentage of data from the defined target domain, for example 5% of images for each class, is selected as labeled. Those target data labels are also used during training. For evaluation of model accuracy, available labels in the corresponding test data sets are used.

Experimental Design

For all the experiments only labeled data is used. This means that in the UDA and SSDA scenarios a part of the labels are ignored, for example the labels of the whole train set of the target domain in the case of UDA. The data sets of the three domains were split into 5 folds and all the experiments evaluated with a 5-fold cross validation. Grayscale images of shape 128x128 were used. All the models are implemented in Python by using Tensorflow (Abadi et al. [1]), with the exception of the AdaMatch model which was implemented in

JAX (Bradbury et al. [22]), by using its authors' original implementation.

Benchmark Models

For training the baseline models a pretrained (on ImageNet) MobileNetV2 [172] architecture is chosen. The same model is used for all models and trainings. Pseudo-Labeling including online as well as offline training and AdaMatch are selected since they are state-of-the-art in unsupervised domain adaptation right now. All benchmark models are described in Chapter 6. DBACS (see 5) is selected for unpaired image alignment and used in semi-susupervised as well as unsupervised fashion.

Metrics and Losses

For pseudo-labeling and AdaMatch the classifier loss is defined as the categorical cross entropy loss with softmax output.

For DBACS, the different parts of the final loss function used for trainingare defined as follows:

- The classifier loss is the categorical cross entropy loss. For semi-supervised learning, it supports task specific alignment of target data;
- for the adversarial loss applied on both discriminator outputs we use KL divergence. This is the original loss function used in GAN [75] and in DANN [68] for classification. Feature matching (FM) loss inspired from [72] added to the adversarial loss. It is introduced in order to encourage aligned and original samples to produce similar activation at each layer of the discriminator instead of just output layer. We follow the notations given in Chapter 5. Then FM is defined for discriminator A and B as as

$$\mathcal{L}_{fm_A}(F, D_A) = \frac{1}{l-1} \sum_{l=1}^{n-1} \|a_l(X_S) - a_l(F(X_T))\|_2^2 \tag{10.1}$$

$$\mathcal{L}_{fm_B}(G, D_B) = \frac{1}{l-1} \sum_{l=1}^{n-1} \|a_l(X_T) - a_l(G(X_S))\|_2^2 \tag{10.2}$$

  where $a_l \in D_A$ represents the raw activation of the l^{th} layer of the discriminator D_A, and n is the total number of discriminator's layers including output.

- For improved alignment, cycle consistency loss and identity loss are defined as L_1 norm and added to both aligner trainings. To preserve the most important information by better preserving features visible to humans rather than noisy, high frequency information, SSIM loss is added in addition to the cyclic loss. KL and SSIM are further described in Section 7.

Models and Hyperparameter

In order to avoid bias caused by particular splits, a cross validation (CV) method is applied. 5-fold cross validation is used for comparing different domain adaptation approaches.

For all models a classifier model needs to be selected. Therefore, a pretrained (on ImageNet) MobileNetV2 [172] architecture is chosen as classifier model for training all models. The MobileNetV2 is chosen because of its compactness and its computational efficiency, thanks to the usage of inverted residual blocks with depthwise separable convolutions. The model was created by loading the feature extractor part of MobileNetV2 without the classification head. The following changes are applied:

- A binary classifier head suitable for the task at hand is attached, made by one dense layer;
- The network is pretrained on colored images hence build with a 3-channel input and a specified shape (e.g. 128,128,3). Since the defect images are grayscale and it is more convenient to work with one channel only, a Conv2D layer with 3 filters was added on top as first and new input layer; its input has shape (128,128,1) and output (128,128,3).

For pseudo-labeling the following settings are installed:

- The offline self-training models are trained with the same hyperparameters as the classifier baselines with a confidence threshold $\tau = 0.9$ for a total of $N = 10$ iterations.
- The online self-training models are also trained with the same hyperparameters as the classifier baseline models with a confidence threshold $\tau = 0.9$, unlabeled to labeled sample ratio is set to $r = 3$.

For AdaMatch models are trained with the same MobileNetV2 architecture as the classifier baseline models. The following hyperparameter are set:

- a confidence threshold $\tau = 0.9$ and unlabeled to labeled sample ratio is set to $r = 3$,

- for training a weight decay of 0.001, a learning rate of 0.0002, a batch size of 64 is chosen.

This means that at each of the training steps the mini-batch would be composed of 64 source images and $64 * r$ target images. Given the limited amount of data, the models are trained for 1M images, instead of the 8M suggested and used in [19]. Hence with a total batch size of $64 * 3 = 256$, the training lasts for 16 epochs, with 256 steps per epoch.

In order to compensate for the reduced training time, the weight of the unsupervised loss is also adapted, to prevent too much weight to the unsupervised loss too early in the training. The total number of steps used in the calculation of the weight of the unsupervised loss is defined as if the model is trained for 8M images. This results in a slowly growing value of the weight as shown in Figure 10.3.

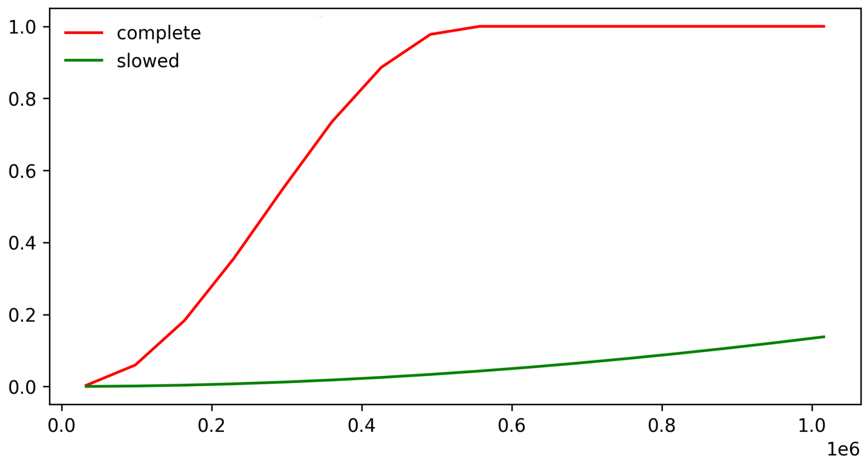


Figure 10.3: AdaMatch schedule of the weight of the unsupervised loss. Progress of the original schedule of the weight of the unsupervised loss ('complete μ schedule') as presented in [19] and the adapted one ('slowed μ schedule') in AdaMatch.

The final models are trained with no distribution alignment, as the label distribution of the target domain data is unknown for unsupervised learning.

In AdaMatch, each source and target image is augmented both in a weak and in a strong way:

- The weak augmentation is defined by a horizontal flip, or mirror. This means that each weakly augmented image is randomly horizontally mirrored with a certain probability.

- The strong augmentation is given by CTAugment [17] applied on top of the same horizontal flip. CTAugment is based on AutoAugment [47], it randomly selects transformations for each sample and learns the magnitudes of each individual transformation on-the-fly. In particular, given a collection of transformations (augmentations), each sample of a mini-batch is augmented with a pipeline consisting of two transformations which are randomly and uniformly sampled; Cutout [53], by a square as big as 1/4 of the image size by 1/4 of the image size, with an area of 1/16 of the total area, is applied on top of the two augmentations, as can be seen in Figure 10.4. The magnitude of each transformation is first chosen randomly and then updated according to how close model's predictions are to the true labels. The set of augmentations is given by: autocontrast, brightness, color, contrast, cutout, equalize, invert, identity, posterize, rescale, rotate, sharpness, shear_x, shear_y, smooth, solarize, translate_x, translate_y.

The DBACS classifier is trained with the same MobileNetV2 architecture as the classifier baseline models. Afterwards the classifier weights are frozen. The other parts are chosen as follows:

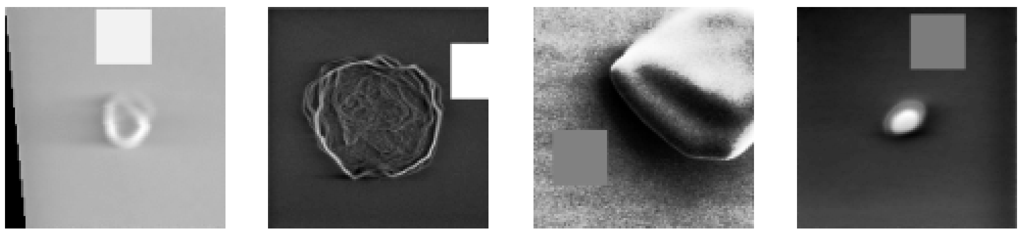
- Both aligner models have a U-Net [167] like architecture taken from [99] and adapted to work with image size 128x128. It has one less downsampling/upsampling layer and does not apply cropping after the second upsampling layer, in order to maintain the same output size as the corresponding downsampling layer, so that the skip connection could be applied.

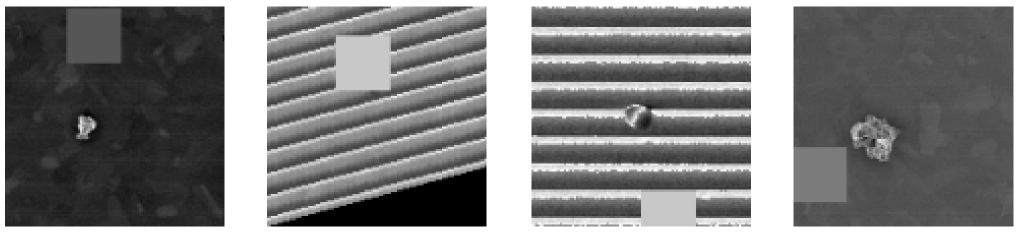
- The architecture for both discriminator models is taken from [72]. It improves CycleGAN's shape deformation by introducing *dilated convolutions*, often used in semantic segmentation: this allows the discriminator model to have a bigger receptive field with respect to the input image, at the same cost of a convolution with a much smaller filter.

The adversarial training of discriminators and aligners is carried out with a training ratio of $r = 2$ in favour of the discriminators. The batch size is $64 * 2 = 128$, the learning rate $lr = 0.00005$, Adam the optimizer and the number of epochs is 300. An epoch has as many steps as $N/batch\_size$ where N is the number of images in the training target domain data set. No early stopping on a validation set is applied since the loss converges. The weights assigned to the different loss terms are set inspired by the descriptions in [72]: $\lambda_{ce} = 1.0$, $\lambda_{adv} = 0.5$, $\lambda_{cycle} = 0.3$, $\lambda_{id} = 0.2$, $\lambda_{fm} = 0.0$. FM loss is dismissed due to no visible impact.

Results and Discussion

The baseline models accuracy are summarized in Table 10.1. The out-of-domain accuracy of the baseline model trained on domain 0 represent the *lower limit* for UDA with domain 0


(a)


(b)

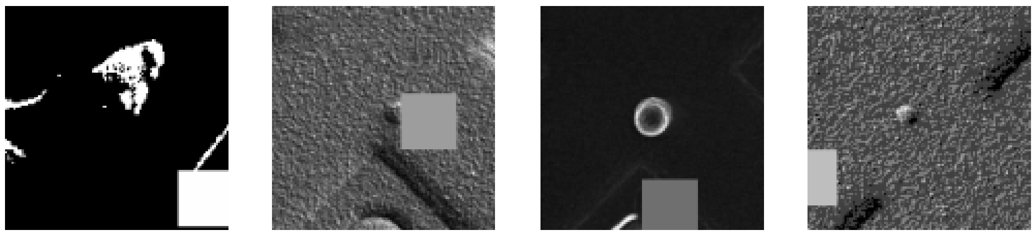

(c)

Figure 10.4: Visualization of augmented defect images. Examples of AdaMatch strongly augmented images with CTAugment for (a) domain 0, (b) domain 3 and (c) domain 8.

as source domain. This means that the DA models should improve the accuracy on domain 3 and 8 in order to be considered successful.

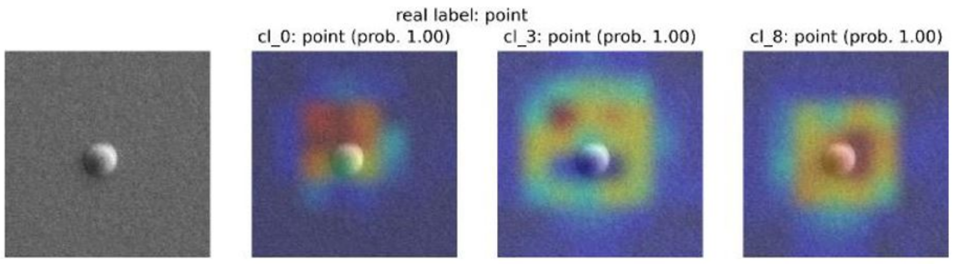
Baseline models accuracy without alignment

Training domain	Domain 0	Domain 3	Domain 8
0	94.26 ± 0.68	77.82 ± 4.91	61.84 ± 3.15
3	68.58 ± 3.47	92.56 ± 0.75	53.50 ± 4.39
8	84.88 ± 3.24	83.46 ± 4.87	90.40 ± 2.01
0 + 5% 3	94.90 ± 0.61	86.56 ± 2.98	64.46 ± 1.98
0 + 5% 8	94.38 ± 1.04	80.54 ± 5.31	83.00 ± 3.06
0 + 3	94.48 ± 1.56	91.04 ± 1.06	67.66 ± 3.56
0 + 8	94.58 ± 0.95	83.26 ± 2.17	91.30 ± 2.71

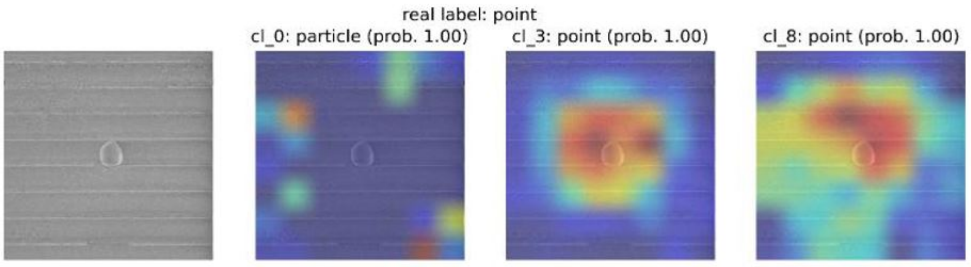
Table 10.1: Baseline models accuracy without alignment. The first column denotes the domain of the train data, the remaining columns the mean 5-fold cross validation accuracy on the corresponding test set and its standard deviation.

Two explainable approaches are presented to better understand the influence of the background towards classification of the baseline models. First, Grad-Cam (Gradient-weighted Class Activation Mapping) by Selvaraju et al. [179] uses the gradient information of the last convolutional layer to highlight most important neurons and hence areas of the image. Figure 10.5 shows Grad-Cam visualization for point defects for models trained on domains 0, domain 3 and domain 8 (left to right, first image presents the original image) while rows are point defect image examples from domain 0 (a), domain 3 (b) and domain 8(c). Second, SHAP (SHapley Additive exPlanations) by Lundberg and Lee [140] measure the influence of data points to the classification outcome by measures of game theory. Figure 10.6 shows SHAP used for visualization for point defects for models trained on domains 0, domain 3 and domain 8 (left to right, first image presents the original image) while rows are point defect image examples from domain 0 (a), domain 3 (b) and domain 8(c).

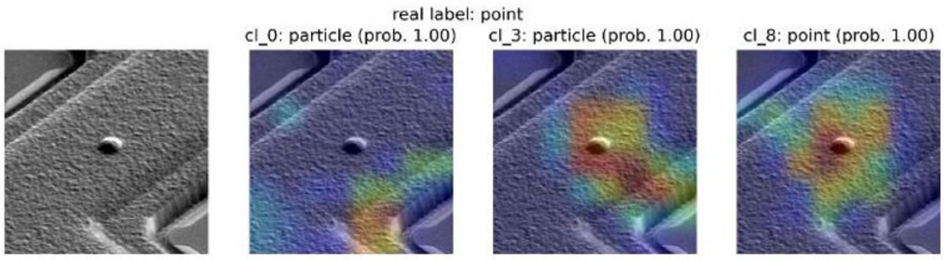
The results for unsupervised domain adaptation for different selection of source domain are presented in Table 10.2 (domain 0), Table 10.3 (domain 3) and Table 10.4 (domain 8).



(a)



(b)

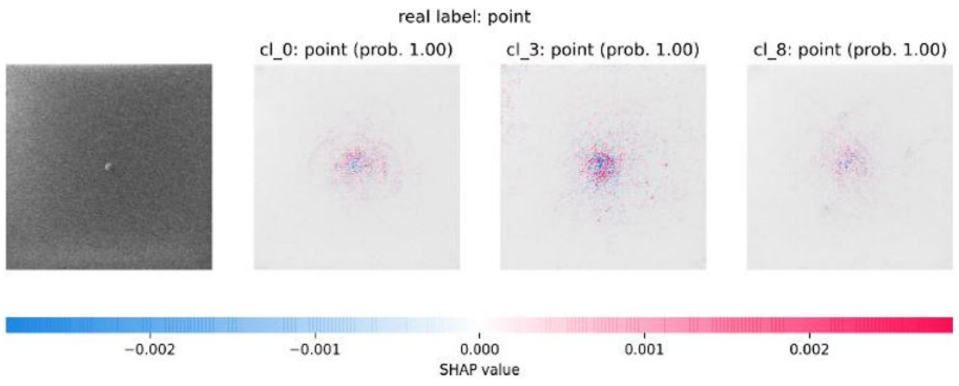


(c)

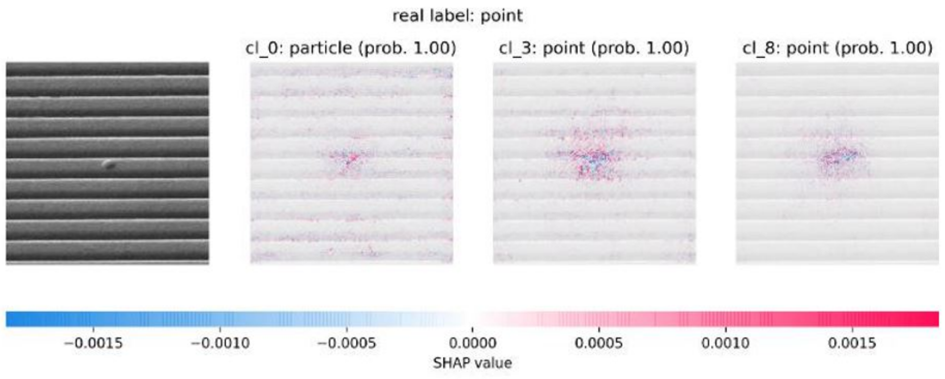
Figure 10.5: Grad-Cam visualization of dedicated classifier. Grad-CAM visual explanations on domains 0, 3 and 8. The first image represents the original image, while the remaining images are the plots of the Grad-CAM visual explanations with respect to the models trained on domain 0, 3 and 8, and the respective predictions. (a) shows an image example from domain 0, (b) from domain 3 and (c) from domain 8.

Evaluation of UDA for target domain - source domain 0

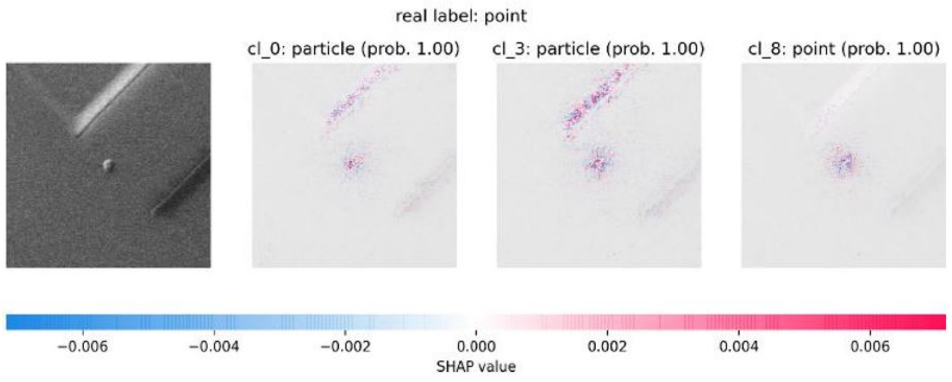
	0 to 3	0 to 8
lower limit	77.82 ± 4.91	61.84 ± 3.15
DBAM	**81.90 ± 2.46**	66.90 ± 5.19
Offline ST	79.02 ± 5.27	61.18 ± 6.34
Online ST	74.18 ± 8.34	61.10 ± 3.22
AdaMatch	**87.46 ± 2.00**	**82.34 ± 1.78**
Oracle	91.04 ± 1.06	91.30 ± 2.71



(a)



(b)



(c)

Figure 10.6: SHAP visualization of dedicated classifier. SHAP visual explanations on domains 0, 3 and 8. The first image represents the original image, while the remaining images are the plots of the SHAP visual explanations with respect to the models trained on domain 0, 3 and 8, and the respective predictions. (a) shows an image example from domain 0, (b) from domain 3 and (c) from domain 8.

Table 10.2: UDA models accuracy - source domain 0. The results are expressed as mean and standard deviation of the 5-fold cross validation accuracy.

Evaluation of UDA for target domain - source domain 3

	3 to 0	3 to 8
lower limit	68.58 ± 3.47	53.50 ± 4.39
DBAM	**71.68** ± **2.88**	**56.98** ± **3.74**
AdaMatch	51.43 ± 4.75	50.18 ± 8.32
Oracle	94.48 ± 1.56	90.16 ± 1.88

Table 10.3: UDA models accuracy - source domain 3. The results are expressed as mean and standard deviation of the 5-fold cross validation accuracy.

Evaluation of UDA for target domain - source domain 8

	8 to 0	8 to 3
lower limit	84.88 ± 3.24	83.46 ± 4.87
DBAM	72.38 ± 3.86	68.16 ± 1.99
AdaMatch	**86.31** ± **2.63**	**82.93** ± **7.09**
Oracle	94.58 ± 0.95	93.12 ± 0.81

Table 10.4: UDA models accuracy - source domain 8. The results are expressed as mean and standard deviation of the 5-fold cross validation accuracy.

In the SSDA scenario, all the models are trained in the same way as in the UDA scenario, with the difference that 5% of the target domain data is treated as labeled.

Evaluation of SSDA for target domain

	0 to 3	0 to 8
lower limit	86.56 ± 2.98	83.00 ± 3.06
DBAM	84.58 ± 5.98	69.36 ± 4.53
Offline ST	84.40 ± 3.12	80.42 ± 3.32
Online ST	86.32 ± 1.94	81.58 ± 1.83
AdaMatch	**87.10** ± **2.80**	**84.26** ± **1.82**
Oracle	91.04 ± 1.06	91.30 ± 2.71

Table 10.5: SSDA (5% of labeled data in the target domain) models accuracy. The results are expressed as mean and standard deviation of the 5-fold cross validation accuracy.

The experiments show that more data are pseudo-labeled after each iteration. This means that the model is more confident about the predictions on unlabeled target domain data as more unseen target domain data is available for the training of the model. Figure 10.7 shows the amount of remaining unlabeled data after each iteration during offline pseudo-labeling/offline ST.

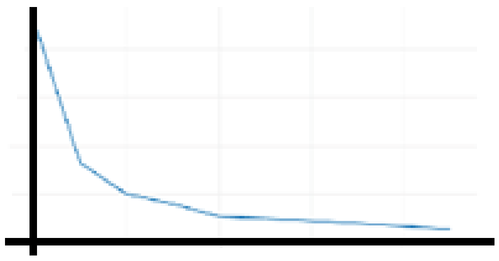
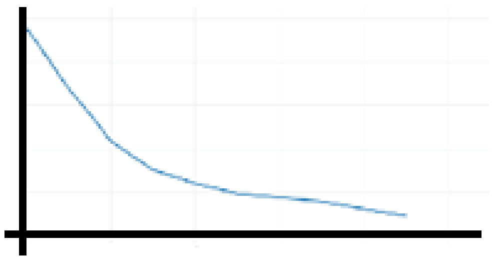



Figure 10.7: Visualization of remaining unlabeled data during pseudo-labeling. Offline ST - number of unlabeled data vs iterations. The x axis represents the number of iterations of an Offline ST model while the y axis represents the number of remaining unlabeled data after each iteration. Left side shows the labeling for domain 3 and the right one for domain 8. Source domain is domain 0.

Figure 10.8 shows examples of defect images from domain 3 respective domain 8 aligned to Domain 0 with DBACS in the UDA setting. Left column shows the original images, right column the aligned images with changed background.

An important step in the pipeline of the AdaMatch model training is the *distribution alignment*. The fundamental assumption behind it is that the target domain label distribution is the same as the source domain label distribution. This is true in case of domain adaptation from domain 0 to domain 3, as the classes in both domains are almost balanced, but it doesn't hold in the case of domain adaptation from domain 0 to domain 8, since the label distribution in domain 8 is clearly imbalanced (see Figure 10.2). In fact, the label distributions of domain 0, 3 and 8 are, respectively, (0.529,0.471), (0.559,0.441) and (0.325,0.675).

Figure 10.9 and Table 10.6 show consistency within pseudo- label accuracy independent of the applied distribution alignment for the 0 to 3 domain adaptation. For the 0 to 8 domain adaptation, instead, the accuracy of the generated pseudo-labels drops significantly when the alignment to the source "online" label distribution is applied, while the most accurate pseudo-labels are obtained when aligning the pseudo labels to the true label distribution. A slightly worse pseudo-label accuracy is obtained when no distribution alignment is applied at all.

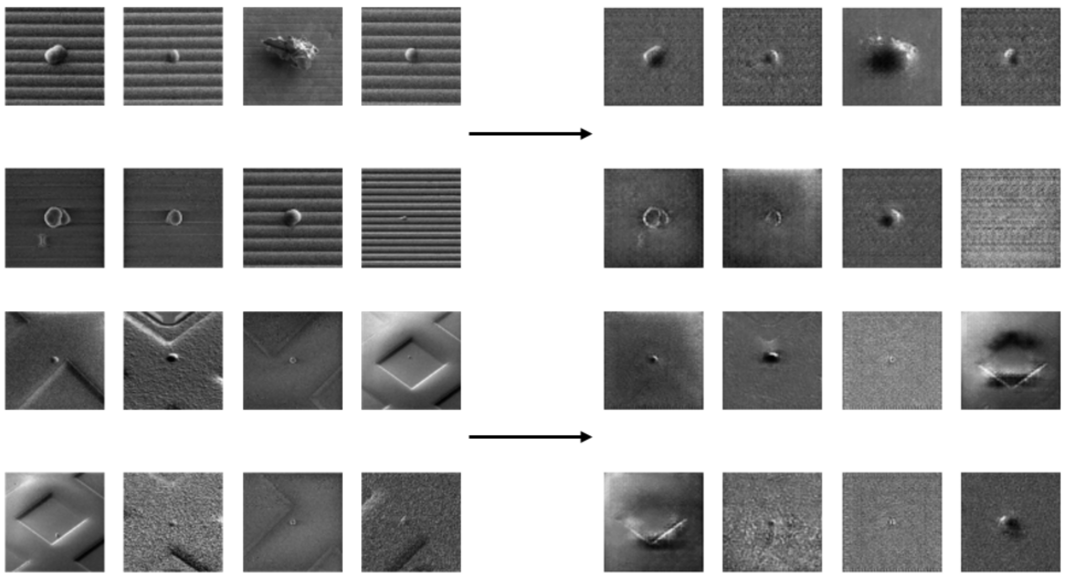


Figure 10.8: Visualization of defect images before and after alignment with DBACS. Examples of defect images aligned to Domain 0 with DBACS in the UDA setting. Left column shows the original images, right column the aligned images with changed background. First two rows show domain 3 and last two rows domain 8.

Comparison of different distribution alignment methods

	0 to 3	0 to 8
Source alignment	85.21 ± 2.60	76.09 ± 2.44
True target alignment	86.24 ± 2.64	84.20 ± 1.02
No alignment	87.46 ± 2.00	82.34 ± 1.78

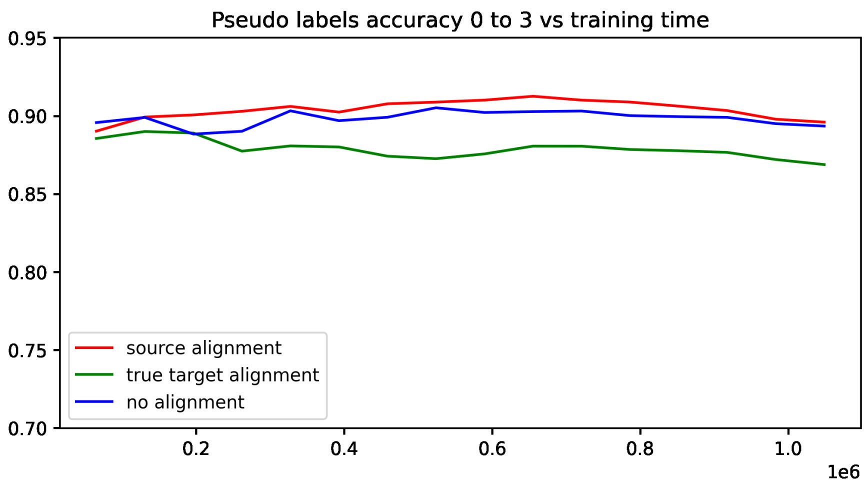
Table 10.6: Comparison of different distribution alignment methods. The results are expressed as mean and standard deviation of the 5-fold cross validation accuracy and represent models in case of alignment w.r.t. the model's output on source domain data on a single batch (source alignment), the true ('unknown in case of UDA') target distribution ('true target alignment') and in case of no distribution alignment ('no alignment').

10.4 Conclusion

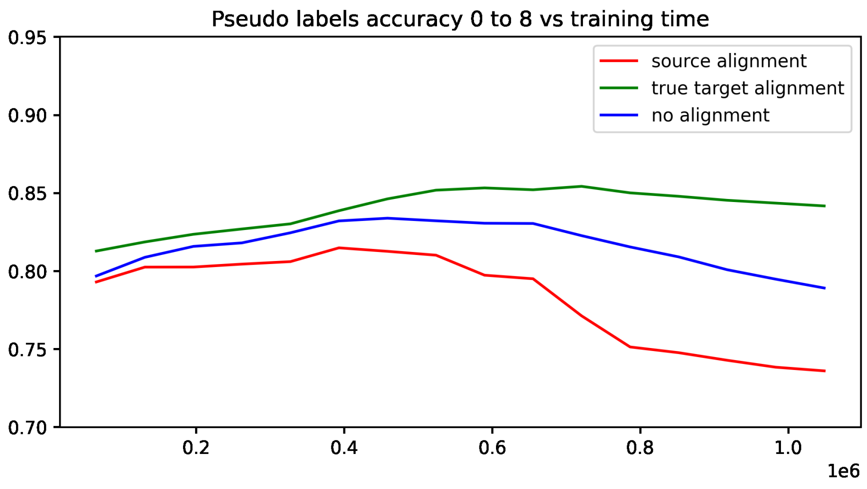
In this chapter a defect classification use case for SEM images with defects and diverse background is presented. Semi- as well as unsupervised training scenarios are covered and DBACS as generative and adversarial approach is tested against pseudo-labeling and state-of-the-art semi-supervised domain adaptation approach AdaMatch. All models show potential and good performance in certain set ups while presenting lower accuracy in others due to various influential factors like image complexity, distribution alignment, number of training samples, various sensitive hyperparameter set ups and specific characteristics of each domain.

Nevertheless, the presented case study shows that improved results can be attainable especially compared to the baseline. Since it only covers binary classification task and pairwise domain adaptation, an extension of this research is promising and straight forward. Therefore, future development areas of this project are:

- multi-source domain adaptation [237];
- multi-target domain adaptation [71];
- extent binary to multi-class classification.



(a)



(b)

Figure 10.9: Accuracy of the pseudo-labels generated by the models. Accuracy of the pseudo-labels generated by the models in case of alignment w.r.t. the model's output on source domain data on a single batch (source alignment), the true ('unknown in case of UDA') target distribution ('true target alignment') and in case of no distribution alignment ('no alignment'). Only one of the 5-fold cross validation curve is plotted.

Part IV

Conclusion

11 Conclusion

In this thesis the necessity, the possibilities but also the difficulties of scaling and generalization of machine learning models in the context of process control in semiconductor manufacturing is discussed. Thanks to growing research, a broad variety of methods and models are available to deal with automation and transfer learning and high efforts are spent to make them usable for industry 4.0 applications.

Inspired by the need of manufacturing for a deep understanding, explainability of models and comparability of features in the context of scalability, the thesis presents DANN-based Alignment Model (DBAM) and its extended version DANN-based Alignment with Cyclic Supervision (DBACS) that unites an interpretable approach of domain adaptation with (equipment) matching.

The application focus in this work is on process control, more precisely virtual metrology, predictive maintenance and defect classification. Relevant examples for process, failure patterns and data sets are selected to showcase all functionalities of the introduced methods but also discuss encountered challenges and shortcomings or limitations when dealing with the highest complexity of semiconductor production. Nevertheless, semiconductor manufacturing related descriptions presented in this thesis cannot be considered complete and the given review only covers most important content directly related to the presented applications. All topics concerning data engineering like data collection, storage and pipelining are also left out.

The selection of benchmark models is inspired by established statistical methods and current state-of-the-art approaches presented in literature. An offline model training is executed hence training time is not critical and therefore not explicitly discussed. Other research directions including optimal transport [165], diffusion models [54] and domain generalization [55] presents alternative approaches and views but are outside of the scope of this thesis.

Deep learning continues to prove itself as first choice for data intensive, demanding and complex manufacturing environments. Nevertheless, inevitable topics that come with fab-wide roll outs and comprehensive use of DL are machine-human interaction as well as trustworthy, ethical and sustainable AI.

References

[1] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al. (2016). Tensorflow: A system for large-scale machine learning. In *12th Symposium on Operating Systems Design and Implementation (16)*, pages 265–283.

[2] Albawi, S., Mohammed, T. A., and Al-Zawi, S. (2017). Understanding of a convolutional neural network. In *2017 International Conference on Engineering and Technology (ICET)*, pages 1–6.

[3] Alipour, N. and Tahmoresnezhad, J. (2022). Heterogeneous domain adaptation with statistical distribution alignment and progressive pseudo label selection. 52(7):8038–8055.

[4] Arena, S., Bodrov, Y., Carletti, M., Gentner, N., Maggipinto, M., Yang, Y., Beghi, A., Kyek, A., and Susto, G. A. (2021). Exploiting 2d coordinates as bayesian priors for deep learning defect classification of sem images. *IEEE Transactions on Semiconductor Manufacturing*, 34(3):436–439.

[5] Arjovsky, M. and Bottou, L. (2017). Towards principled methods for training generative adversarial networks.

[6] Arjovsky, M., Chintala, S., and Bottou, L. (2017). Wasserstein gan. *arXiv*, abs/1701.07875.

[7] Azamfar, M., Li, X., and Lee, J. (2020). Deep learning-based domain adaptation method for fault diagnosis in semiconductor manufacturing. *IEEE Transactions on Semiconductor Manufacturing*, 33(3):445–453.

[8] Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv*, abs/1409.0473.

[9] Bai, S., Kolter, J. Z., and Koltun, V. (2018). An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv*, abs/1803.01271.

[10] Bashath, S., Perera, N., Tripathi, S., Manjang, K., Dehmer, M., and Streib, F. E. (2022). A data-centric review of deep transfer learning with applications to text data. *Information Sciences*, 585:498–528.

[11] Baxter, J. (1995). Learning internal representations. In *Proceedings of the eighth annual conference on Computational learning theory*, pages 311–320.

[12] Bello, I., Zoph, B., Le, Q., Vaswani, A., and Shlens, J. (2019). Attention augmented convolutional networks. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 3285–3294.

[13] Ben-David, S., Blitzer, J., Crammer, K., Kulesza, A., Pereira, F., and Vaughan, J. W. (2010). A theory of learning from different domains. *Machine learning*, 79(1):151–175.

[14] Ben-David, S., Blitzer, J., Crammer, K., and Pereira, F. (2007). Analysis of representations for domain adaptation. In Schölkopf, B., Platt, J., and Hoffman, T., editors, *Advances in Neural Information Processing Systems*, volume 19. MIT Press.

[15] Ben-David, S. and Schuller, R. (2003). Exploiting task relatedness for multiple task learning. In Schölkopf, B. and Warmuth, M. K., editors, *Learning Theory and Kernel Machines*, pages 567–580, Berlin, Heidelberg. Springer Berlin Heidelberg.

[16] Bergstra, J. and Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(2):281–305.

[17] Berthelot, D., Carlini, N., Cubuk, E. D., Kurakin, A., Sohn, K., Zhang, H., and Raffel, C. (2019). Remixmatch: Semi-supervised learning with distribution alignment and augmentation anchoring. abs/1911.09785.

[18] Berthelot, D., Roelofs, R., Sohn, K., Carlini, N., and Kurakin, A. (2021a). Adamatch: A unified approach to semi-supervised learning and domain adaptation. abs/2106.04732.

[19] Berthelot, D., Roelofs, R., Sohn, K., Carlini, N., and Kurakin, A. (2021b). Adamatch: A unified approach to semi-supervised learning and domain adaptation. abs/2106.04732.

[20] Bishop, C. M. (2006). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg.

[21] Bleakie, A. and Djurdjanovic, D. (2013). Feature extraction, condition monitoring, and fault modeling in semiconductor manufacturing systems. *Computers in Industry*, 64(3):203–213.

[22] Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., and Zhang, Q. (2018). JAX: composable transformations of Python+NumPy programs.

[23] Breiman, L. (2004). Random forests. *Machine Learning*, 45(1):5–32.

[24] Cao, Y., Ding, Y., Jia, M., and Tian, R. (2021). A novel temporal convolutional network with residual self-attention mechanism for remaining useful life prediction of rolling bearings. *Reliability Engineering System Safety*, 215.

[25] Carletti, M., Maggipinto, M., Beghi, A., Antonio Susto, G., Gentner, N., Yang, Y., and Kyek, A. (2020). Interpretable anomaly detection for knowledge discovery in semiconductor manufacturing. In *2020 Winter Simulation Conference (WSC)*, pages 1875–1885.

[26] Carvalho, T. P., Soares, F. A., Vita, R., Francisco, R. d. P., Basto, J. P., and Alcalá, S. G. (2019). A systematic literature review of machine learning methods applied to predictive maintenance. *Computers & Industrial Engineering*, 137.

[27] Cerquitelli, T., Nikolakis, N., O'Mahony, N., Macii, E., Ippolito, M., and Makris, S., editors (2021). *Predictive Maintenance in Smart Factories*. Information Fusion and Data Science. Springer, Singapore.

[28] Chandrashekar, G. and Sahin, F. (2014). A survey on feature selection methods. *Computers & Electrical Engineering*, 40(1):16–28.

[29] Chapelle, O., Schlkopf, B., and Zien, A. (2010). *Semi-Supervised Learning*. The MIT Press, 1st edition.

[30] Chapman, J. and Wang, H.-T. (2021). Cca-zoo: A collection of regularized, deep learning based, kernel, and probabilistic cca methods in a scikit-learn style framework. *Journal of Open Source Software*, 6(68):3823.

[31] Chen, C.-H., Zhao, W.-D., Pang, T., and Lin, Y.-Z. (2020). Virtual metrology of semiconductor pvd process based on combination of tree-based ensemble model. *ISA Transactions*, 103:192–202.

[32] Chen, H., Guan, M., and Li, H. (2021). Arcyclegan: Improved cyclegan for style transferring of fruit images. *IEEE Access*, 9:46776–46787.

[33] Chen, P., Wu, S., Junshien Lin, Ko, F., Lo, H., Wang, J., Yu, C. H., and Liang, M. S. (2005). Virtual metrology: a solution for wafer to wafer advanced process control. In *ISSM 2005, IEEE International Symposium on Semiconductor Manufacturing, 2005.*, pages 155–157.

[34] Chen, T. and Guestrin, C. (2016). Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 785–794, New York, NY, USA. Association for Computing Machinery.

[35] Chen, X., Sun, Y., Athiwaratkun, B., Cardie, C., and Weinberger, K. (2018). Adversarial deep averaging networks for cross-lingual sentiment classification. *Transactions of the Association for Computational Linguistics*, 6:557–570.

[36] Chen, Z., Zhang, K., Ding, S. X., Shardt, Y. A., and Hu, Z. (2016). Improved canonical correlation analysis-based fault detection methods for industrial processes. *Journal of Process Control*, 41:26–34.

[37] Cheng, F.-T., Chen, Y.-T., Su, Y.-C., and Zeng, D.-L. (2008). Evaluating reliance level of a virtual metrology system. *IEEE Transactions on Semiconductor Manufacturing*, 21(1):92–103.

[38] Cheon, S., Lee, H., Kim, C. O., and Lee, S. H. (2019). Convolutional neural network for wafer surface defect classification and the detection of unknown defect class. *IEEE Transactions on Semiconductor Manufacturing*, 32(2):163–170.

[39] Cho, K., Van Merriënboer, B., Bahdanau, D., and Bengio, Y. (2014). On the properties of neural machine translation: Encoder-decoder approaches. *arXiv*, abs/1409.1259.

[40] Chollet, F. et al. (2015). Keras. https://keras.io.

[41] Chou, P. B., Rao, A. R., Sturzenbecker, M. C., Wu, F. Y., and Brecher, V. H. (1997). Automatic defect classification for semiconductor manufacturing. *Machine Vision and Applications*, 9(4):201–214.

[42] Chouichi, A., Blue, J., Yugma, C., and Pasqualini, F. (2020). Chamber-to-chamber discrepancy detection in semiconductor manufacturing. *IEEE Transactions on Semiconductor Manufacturing*, 33(1):86–95.

[43] Chung, J., Gulcehre, C., Cho, K., and Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv*, abs/1412.3555.

[44] Compare, M., Baraldi, P., and Zio, E. (2020). Challenges to iot-enabled predictive maintenance for industry 4.0. *IEEE Internet of Things Journal*, 7(5):4585–4597.

[45] Courty, N., Flamary, R., Tuia, D., and Rakotomamonjy, A. (2016). Optimal transport for domain adaptation. *arXiv*, abs/1507.00504.

[46] Crammer, K., Kearns, M., and Wortman, J. (1995). Learning from data of variable quality. In *In NIPS 18*.

[47] Cubuk, E. D., Zoph, B., Mane, D., Vasudevan, V., and Le, Q. V. (2018). Autoaugment: Learning augmentation policies from data. abs/1805.09501.

[48] de Mathelin, A., Deheeger, F., Mougeot, M., and Vayatis, N. (2021a). Handling distribution shift in tire design. In *NeurIPS 2021 Workshop on Distribution Shifts: Connecting Methods and Applications*.

[49] de Mathelin, A., Deheeger, F., Richard, G., Mougeot, M., and Vayatis, N. (2021b). Adapt: Awesome domain adaptation python toolbox. *arXiv*, abs/2107.03049.

[50] de Oliveira da Costa, P. R., Akçay, A., Zhang, Y., and Kaymak, U. (2020). Remaining useful lifetime prediction via deep domain adaptation. *Reliability Engineering System Safety*, 195.

[51] Delalleau, O. and Bengio, Y. (2011). Shallow vs. deep sum-product networks. In Shawe-Taylor, J., Zemel, R., Bartlett, P., Pereira, F., and Weinberger, K., editors, *Advances in Neural Information Processing Systems*, volume 24. Curran Associates, Inc.

[52] Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv*, abs/1810.04805.

[53] DeVries, T. and Taylor, G. W. (2017). Improved regularization of convolutional neural networks with cutout. abs/1708.04552.

[54] Dhariwal, P. and Nichol, A. (2021). Diffusion models beat gans on image synthesis. In Ranzato, M., Beygelzimer, A., Dauphin, Y., Liang, P., and Vaughan, J. W., editors, *Advances in Neural Information Processing Systems*, volume 34, pages 8780–8794. Curran Associates, Inc.

[55] Dou, Q., Coelho de Castro, D., Kamnitsas, K., and Glocker, B. (2019). Domain generalization via model-agnostic learning of semantic features. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

[56] Dowson, D. and Landau, B. (1982). The fréchet distance between multivariate normal distributions. *Journal of Multivariate Analysis*, 12(3):450–455.

[57] Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(61):2121–2159.

[58] Dudani, S. A. (1976). The distance-weighted k-nearest-neighbor rule. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-6(4):325–327.

[59] Fan, S.-K. S., Hsu, C.-Y., Tsai, D.-M., He, F., and Cheng, C.-C. (2020). Data-driven approach for fault detection and diagnostic in semiconductor manufacturing. *IEEE Transactions on Automation Science and Engineering*, 17(4):1925–1936.

[60] Fang, Z., Lu, J., Liu, F., and Zhang, G. (2022). Semi-supervised heterogeneous domain adaptation: Theory and algorithms. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

[61] Farahani, H. S., Fatehi, A., Nadali, A., and Shoorehdeli, M. A. (2020). A novel method for designing transferable soft sensors and its application. *arXiv*, abs/2008.02186.

[62] Farshchian, A., Gallego, J. A., Cohen, J. P., Bengio, Y., Miller, L. E., and Solla, S. A. (2019). Adversarial domain adaptation for stable brain-machine interfaces. *arXiv*, abs/1810.00045.

[63] Feng, J., Jia, X., Zhu, F., Moyne, J., Iskandar, J., and Lee, J. (2019). An online virtual metrology model with sample selection for the tracking of dynamic manufacturing processes with slow drift. *IEEE Transactions on Semiconductor Manufacturing*, 32(4):574–582.

[64] Fernando, B., Habrard, A., Sebban, M., and Tuytelaars, T. (2013). Unsupervised visual domain adaptation using subspace alignment. In *2013 IEEE International Conference on Computer Vision*, pages 2960–2967.

[65] Freund, Y. and Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139.

[66] Friedman, J., Hastie, T., and Tibshirani, R. (2001). *The elements of statistical learning*, volume 1. Springer series in statistics New York.

[67] Friedman, J. H. (2001). Greedy function approximation: a gradient boosting machine. *Annals of statistics*, 29:1189–1232.

[68] Ganin, Y., Ustinova, E., Ajakan, H., Germain, P., Larochelle, H., Laviolette, F., Marchand, M., and Lempitsky, V. (2016). Domain-adversarial training of neural networks. *Journal of machine learning research*, 17(1):2096–2030.

[69] Gentner, N., Carletti, M., Kyek, A., Susto, G. A., and Yang, Y. (2021). Dbam: Making virtual metrology/soft sensing with time series data scalable through deep learning. *Control Engineering Practice*, 116:104914.

[70] Gentner, N., Kyek, A., Yang, Y., Carletti, M., and Susto, G. A. (2020). Enhancing scalability of virtual metrology: A deep learning-based approach for domain adaptation. In *2020 Winter Simulation Conference (WSC)*, pages 1898–1909. IEEE.

[71] Gholami, B., Sahu, P., Rudovic, O., Bousmalis, K., and Pavlovic, V. (2020). Unsupervised multi-target domain adaptation: An information theoretic approach. *IEEE Transactions on Image Processing*, 29:3993–4002.

[72] Gokaslan, A., Ramanujan, V., Ritchie, D., Kim, K. I., and Tompkin, J. (2018a). Improving shape deformation in unsupervised image-to-image translation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 649–665.

[73] Gokaslan, A., Ramanujan, V., Ritchie, D., Kim, K. I., and Tompkin, J. (2018b). Improving shape deformation in unsupervised image-to-image translation. In Ferrari, V., Hebert, M., Sminchisescu, C., and Weiss, Y., editors, *Computer Vision – ECCV 2018*, pages 662–678, Cham. Springer International Publishing.

[74] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

[75] Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, page 2672–2680, Cambridge, MA, USA. MIT Press.

[76] Grill, J.-B., Strub, F., Altché, F., Tallec, C., Richemond, P. H., Buchatskaya, E., Doersch, C., Pires, B. A., Guo, Z. D., Azar, M. G., Piot, B., Kavukcuoglu, K., Munos, R., and Valko, M. (2020). Bootstrap your own latent: A new approach to self-supervised learning. *arXiv*, abs/2006.07733.

[77] Gu, J., Wang, Z., Kuen, J., Ma, L., Shahroudy, A., Shuai, B., Liu, T., Wang, X., Wang, G., Cai, J., and Chen, T. (2018). Recent advances in convolutional neural networks. *Pattern Recognition*, 77:354–377.

[78] Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., and Courville, A. (2017). Improved training of wasserstein gans. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 5769–5779, Red Hook, NY, USA. Curran Associates Inc.

[79] Guo, F., Xie, R., and Huang, B. (2020). A deep learning just-in-time modeling approach for soft sensor based on variational autoencoder. *Chemometrics and Intelligent Laboratory Systems*, 197.

[80] Haddad, B. M., Yang, S., Karam, L. J., Ye, J., Patel, N. S., and Braun, M. W. (2016). Multifeature, sparse-based approach for defects detection and classification

in semiconductor units. *IEEE Transactions on Automation Science and Engineering*, 15(1):145–159.

[81] Hardoon, D. R., Szedmak, S., and Shawe-Taylor, J. (2004). Canonical correlation analysis: An overview with application to learning methods. *Neural Computation*, 16(12):2639–2664.

[82] Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., and Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, 585(7825):357–362.

[83] He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034.

[84] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778.

[85] Heimes, F. O. (2008). Recurrent neural networks for remaining useful life estimation. In *2008 International Conference on Prognostics and Health Management*, pages 1–6.

[86] Heng, H., Liao, T., Didari, S., and Rajagopal, H. (2021). *Chamber matching with neural networks in semiconductor equipment tools*. Applied Materials, Inc., US Patent 11,133,204.

[87] Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., and Hochreiter, S. (2017). Gans trained by a two time-scale update rule converge to a local nash equilibrium. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

[88] Hilleringmann, U. (1996). *Silizium-Halbleitertechnologie*. Springer.

[89] Hinton, G. (2012). Neural networks for machine learning lecture 6. Coursera.

[90] Hochreiter, S., Heusel, M., and Obermayer, K. (2007). Fast model-based protein homology detection without alignment. *Bioinformatics*, 23(14):1728–1736.

[91] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8):1735–1780.

[92] Hoffman, J., Tzeng, E., Park, T., Zhu, J.-Y., Isola, P., Saenko, K., Efros, A., and Darrell, T. (2018). CyCADA: Cycle-consistent adversarial domain adaptation. In Dy, J. and Krause, A., editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1989–1998. PMLR.

[93] Hofmann, T., Schölkopf, B., and Smola, A. (2008). Kernel methods in machine learning. *Annals of Statistics*, 36(3):1171–1220.

[94] Hsu, C.-Y. and Chien, J.-C. (2022). Ensemble convolutional neural networks with weighted majority for wafer bin map pattern classification. *Journal of Intelligent Manufacturing*, 33(3):831–844.

[95] Hsu, C.-Y., Lu, Y.-W., and Yan, J.-H. (2022). Temporal convolution-based long-short term memory network with attention mechanism for remaining useful life prediction. *IEEE Transactions on Semiconductor Manufacturing*, 35(2):220–228.

[96] Huang, W., Khorasgani, H., Gupta, C., Farahat, A., and Zheng, S. (2018). Remaining useful life estimation for systems with abrupt failures. In *Annual conference of the PHM society. September*, pages 24–27.

[97] Imoto, K., Nakai, T., Ike, T., Haruki, K., and Sato, Y. (2018). A cnn-based transfer learning method for defect classification in semiconductor manufacturing. In *2018 international symposium on semiconductor manufacturing (ISSM)*, pages 1–3. IEEE.

[98] Iskandar, J., Moyne, J., Subrahmanyam, K., Hawkins, P., and Armacost, M. (2015). Predictive maintenance in semiconductor manufacturing. In *2015 26th Annual SEMI Advanced Semiconductor Manufacturing Conference (ASMC)*, pages 384–389.

[99] Isola, P., Zhu, J.-Y., Zhou, T., and Efros, A. A. (2017). Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1125–1134.

[100] Ito, H., England, J., Plumb, F., and Fotheringham, I. (1995). Plasma flood system for the reduction of charging of wafers during ion implantation. US Patent 5,399,871.

[101] Jalali, A., Heistracher, C., Schindler, A., Haslhofer, B., Nemeth, T., Glawar, R., Sihn, W., and De Boer, P. (2019). Predicting time-to-failure of plasma etching equipment using machine learning. In *2019 IEEE International Conference on Prognostics and Health Management (ICPHM)*, pages 1–8.

[102] Jebara, T. (2004). *Generative Versus Discriminative Learning*, pages 17–60. Springer US, Boston, MA.

[103] Jiang, X., Lao, Q., Matwin, S., and Havaei, M. (2020). Implicit class-conditioned domain alignment for unsupervised domain adaptation. In III, H. D. and Singh, A., editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 4816–4827. PMLR.

[104] Jin, C. H., Na, H. J., Piao, M., Pok, G., and Ryu, K. H. (2019). A novel dbscan-based defect pattern detection and classification framework for wafer bin map. *IEEE Transactions on Semiconductor Manufacturing*, 32(3):286–292.

[105] Jolliffe, I. (2010). *Principal Component Analysis*. Springer Series in Statistics. Springer New York.

[106] Jung, D., Ng, K. Y., Frisk, E., and Krysander, M. (2018). Combining model-based diagnosis and data-driven anomaly classifiers for fault isolation. *Control Engineering Practice*, 80:146–156.

[107] Kang, H. and Kang, S. (2021). A stacking ensemble classifier with handcrafted and convolutional features for wafer map pattern classification. *Computers in Industry*, 129:103450.

[108] Kang, S. (2017). On effectiveness of transfer learning approach for neural network-based virtual metrology modeling. *IEEE Transactions on Semiconductor Manufacturing*, 31(1):149–155.

[109] Kang, S., Kim, D., and Cho, S. (2016). Efficient feature selection-based on random forward search for virtual metrology modeling. *IEEE Transactions on Semiconductor Manufacturing*, 29(4):391–398.

[110] Keany, E. (2020). BorutaShap : A wrapper feature selection method which combines the Boruta feature selection algorithm with Shapley values.

[111] Kearns, M. J. and Vazirani, U. V. (1994). *An Introduction to Computational Learning Theory*. MIT Press, Cambridge, MA, USA.

[112] Khalid, S., Khalil, T., and Nasreen, S. (2014). A survey of feature selection and feature extraction techniques in machine learning. In *2014 Science and Information Conference*, pages 372–378.

[113] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv*.

[114] Kiranyaz, S., Avci, O., Abdeljaber, O., Ince, T., Gabbouj, M., and Inman, D. J. (2021). 1d convolutional neural networks and applications: A survey. *Mechanical Systems and Signal Processing*, 151:107398.

[115] Koch, G., Zemel, R., Salakhutdinov, R., et al. (2015). Siamese neural networks for one-shot image recognition.

[116] Kong, Y. and Ni, D. (2020). A semi-supervised and incremental modeling framework for wafer map classification. *IEEE Transactions on Semiconductor Manufacturing*, 33(1):62–71.

[117] Krupitzer, C., Wagenhals, T., Züfle, M., Lesch, V., Schäfer, D., Mozaffarin, A., Edinger, J., Becker, C., and Kounev, S. (2020). A survey on predictive maintenance for industry 4.0. *arXiv*, abs/2002.08224.

[118] Kumar, N., Zhang, L., and Nayar, S. (2008). What is a good nearest neighbors algorithm for finding similar patches in images? In Forsyth, D., Torr, P., and Zisserman, A., editors, *Computer Vision – ECCV 2008*, pages 364–378, Berlin, Heidelberg. Springer Berlin Heidelberg.

[119] Kyeong, K. and Kim, H. (2018). Classification of mixed-type defect patterns in wafer bin maps using convolutional neural networks. *IEEE Transactions on Semiconductor Manufacturing*, 31(3):395–402.

[120] Lea, C., Flynn, M. D., Vidal, R., Reiter, A., and Hager, G. D. (2016). Temporal convolutional networks for action segmentation and detection. *arXiv*, abs/1611.05267.

[121] LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (1989). Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation*, 1(4):541–551.

[122] Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86:2278 – 2324.

[123] Lee, D.-H. et al. (2013). Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks. In *Workshop on challenges in representation learning, ICML*, volume 3, page 896.

[124] Lee, J.-H. and Lee, J.-H. (2019). A reliable defect detection method for patterned wafer image using convolutional neural networks with the transfer learning. In *IOP Conference Series: Materials Science and Engineering*, volume 647. IOP Publishing.

[125] Lee, K. B. and Kim, C. O. (2020). Recurrent feature-incorporated convolutional neural network for virtual metrology of the chemical mechanical planarization process. *Journal of Intelligent Manufacturing*, 31(1):73–86.

[126] Li, B., Wang, Y., Zhang, S., Li, D., Darrell, T., Keutzer, K., and Zhao, H. (2020a). Learning invariant representations and risks for semi-supervised domain adaptation. *arXiv*, abs/2010.04647.

[127] Li, D., Li, L., Li, X., Ke, Z., and Hu, Q. (2020b). 411:351–363.

[128] Li, J. and He, D. (2020). A bayesian optimization adabn-dcnn method with self-optimized structure and hyperparameters for domain adaptation remaining useful life prediction. *IEEE Access*, 8:41482–41501.

[129] Li, J., Li, X., and He, D. (2019a). Domain adaptation remaining useful life prediction method based on adabn-dcnn. In *2019 Prognostics and System Health Management Conference (PHM-Qingdao)*, pages 1–6.

[130] Li, K. S.-M., Jiang, X.-H., Chen, L. L.-Y., Wang, S.-J., Huang, A. Y.-A., Chen, J. E., Liang, H.-C., and Hsu, C.-L. (2022). Wafer defect pattern labeling and recognition using semi-supervised learning. *IEEE Transactions on Semiconductor Manufacturing*, 35(2):291–299.

[131] Li, X., Ding, Q., and Sun, J.-Q. (2018a). Remaining useful life estimation in prognostics using deep convolution neural networks. *Reliability Engineering System Safety*, 172:1–11.

[132] Li, Y., Wang, N., Shi, J., Hou, X., and Liu, J. (2018b). Adaptive batch normalization for practical domain adaptation. *Pattern Recognition*, 80:109–117.

[133] Li, Z., Wu, D., Hu, C., and Terpenny, J. (2019b). An ensemble learning-based prognostic approach with degradation-dependent weights for remaining useful life prediction. *Reliability Engineering System Safety*, 184:110–122.

[134] Liu, C., Zhang, L., Li, J., Zheng, J., and Wu, C. (2021a). Two-stage transfer learning for fault prognosis of ion mill etching process. *IEEE Transactions on Semiconductor Manufacturing*, 34(2):185–193.

[135] Liu, J., Guo, F., Zhang, Y., Hou, B., and Zhou, H. (2021b). Defect classification on limited labeled samples with multiscale feature fusion and semi-supervised learning. *Applied Intelligence*, 52(7):8243–8258.

[136] Loog, M. (2012). Nearest neighbor-based importance weighting. In *2012 IEEE International Workshop on Machine Learning for Signal Processing*, pages 1–6. IEEE.

[137] Lu, H.-W. and Lee, C.-Y. (2022). Kernel-based dynamic ensemble technique for remaining useful life prediction. *IEEE Robotics and Automation Letters*, 7(2):1142–1149.

[138] Lu, Y.-W., Hsu, C.-Y., and Huang, K.-C. (2020). An autoencoder gated recurrent unit for remaining useful life prediction. *Processes*, 8(9).

[139] Lu, Y.-W., Liu, K.-L., and Hsu, C.-Y. (2019). Conditional generative adversarial network for defect classification with class imbalance. In *2019 IEEE International Conference on Smart Manufacturing, Industrial Logistics Engineering (SMILE)*, pages 146–149.

[140] Lundberg, S. M. and Lee, S.-I. (2017). A unified approach to interpreting model predictions. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 4768–4777, Red Hook, NY, USA. Curran Associates Inc.

[141] Lynn, S., Ringwood, J., Ragnoli, E., McLoone, S., and MacGearailty, N. (2009). Virtual metrology for plasma etch using tool variables. In *2009 IEEE/SEMI Advanced Semiconductor Manufacturing Conference*, pages 143–148. IEEE.

[142] Lynn, S. A., Ringwood, J., and MacGearailt, N. (2012). Global and local virtual metrology models for a plasma etch process. *IEEE Transactions on Semiconductor Manufacturing*, 25(1):94–103.

[143] MacKay, D. J. C. (2002). *Information Theory, Inference Learning Algorithms*. Cambridge University Press, USA.

[144] Maggipinto, M., Beghi, A., McLoone, S., and Susto, G. A. (2019). Deepvm: A deep learning-based approach with automatic feature extraction for 2d input data virtual metrology. *Journal of Process Control*, 84:24–34.

[145] Maggipinto, M., Masiero, C., Beghi, A., and Susto, G. A. (2018). A convolutional autoencoder approach for feature extraction in virtual metrology. *Procedia Manufacturing*, 17:126–133.

[146] May, G. and Spanos, C. (2006). *Fundamentals of Semiconductor Manufacturing and Process Control*. IEEE Press. John Wiley & Sons.

[147] Mobley, R. K. (2002). *An Introduction to Predictive Maintenance (Second Edition)*. Plant Engineering. Butterworth-Heinemann, Burlington, second edition edition.

[148] Mönch, L., Fowler, J., and Mason, S. (2012). *Production Planning and Control for Semiconductor Wafer Fabrication Facilities: Modeling, Analysis, and Systems*. Operations Research/Computer Science Interfaces Series. Springer New York.

[149] Moyne, J. and Iskandar, J. (2017). Big data analytics for smart manufacturing: Case studies in semiconductor manufacturing. *Processes*, 5(3):39.

[150] Mutegeki, R. and Han, D. S. (2020). A cnn-lstm approach to human activity recognition. In *2020 International Conference on Artificial Intelligence in Information and Communication (ICAIIC)*, pages 362–366.

[151] Nakazawa, T. and Kulkarni, D. V. (2018). Wafer map defect pattern classification and image retrieval using convolutional neural network. *IEEE Transactions on Semiconductor Manufacturing*, 31(2):309–314.

[152] Oja, E., Ogawa, H., and Wangviwattana, J. (1992). Principal component analysis by homogeneous neural networks: The weighted subspace criterion. *IEICE Transactions on Information and Systems*, 75(3):366–375.

[153] Oliver, A., Odena, A., Raffel, C., Cubuk, E. D., and Goodfellow, I. J. (2018). Realistic evaluation of deep semi-supervised learning algorithms. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, page 3239–3250, Red Hook, NY, USA. Curran Associates Inc.

[154] Oza, P., Sindagi, V. A., VS, V., and Patel, V. M. (2021). Unsupervised domain adaptation of object detectors: A survey. *arXiv*, abs/2105.13502.

[155] O'Leary, J., Sawlani, K., and Mesbah, A. (2020). Deep learning for classification of the chemical composition of particle defects on semiconductor wafers. *IEEE Transactions on Semiconductor Manufacturing*, 33(1):72–85.

[156] Pan, S. J., Tsang, I. W., Kwok, J. T., and Yang, Q. (2011). Domain adaptation via transfer component analysis. *IEEE Transactions on Neural Networks*, 22(2):199–210.

[157] Pan, S. J. and Yang, Q. (2010). A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359.

[158] Pardoe, D. and Stone, P. (2010). Boosting for regression transfer. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, page 863–870, Madison, WI, USA. Omnipress.

[Park and Kim] Park, C. and Kim, S. B. Virtual metrology modeling of time-dependent spectroscopic signals by a fused lasso algorithm. *Journal of Process Control*, 42:51–58.

[160] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

[161] Perry, R., Mischler, G., Guo, R., Lee, T., Chang, A., Koul, A., Franz, C., Richard, H., Carmichael, I., Ablin, P., Gramfort, A., and Vogelstein, J. T. (2021). mvlearn: Multiview machine learning in python. *Journal of Machine Learning Research*, 22(109):1–7.

[162] Purushotham, S., Carvalho, W., Nilanon, T., and Liu, Y. (2017). Variational recurrent adversarial deep domain adaptation. In *ICLR*.

[163] Radford, A., Metz, L., and Chintala, S. (2016). Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv*, abs/1511.06434.

[164] Ramsundar, B. and Zadeh, R. (2018). *TensorFlow for Deep Learning: From Linear Regression to Reinforcement Learning*. O'Reilly Media.

[165] Redko, I., Courty, N., Flamary, R., and Tuia, D. (2019). Optimal transport for multi-source domain adaptation under target shift. In Chaudhuri, K. and Sugiyama, M., editors, *Proceedings of the Twenty-Second International Conference on Artificial Intelligence and Statistics*, volume 89 of *Proceedings of Machine Learning Research*, pages 849–858. PMLR.

[Reimers and Gurevych] Reimers, N. and Gurevych, I. Optimal hyperparameters for deep lstm-networks for sequence labeling tasks. *arXiv*, abs/1707.06799.

[167] Ronneberger, O., Fischer, P., and Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer.

[168] Ruder, S. (2016). An overview of gradient descent optimization algorithms. *arXiv*, abs/1609.04747.

[169] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323(6088):533–536.

[170] Saeys, Y., Inza, I., and Larrañaga, P. (2007). A review of feature selection techniques in bioinformatics. *Bioinformatics*, 23(19):2507–2517.

[171] Sagheer, A. and Kotb, M. (2019). Unsupervised pre-training of a deep lstm-based stacked autoencoder for multivariate time series forecasting problems. *Scientific reports*, 9(1):1–16.

[172] Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C. (2018). Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520.

[173] Santos, T., Schrunner, S., Geiger, B. C., Pfeiler, O., Zernig, A., Kaestner, A., and Kern, R. (2019). Feature extraction from analog wafermaps: A comparison of classical image processing and a deep generative model. *IEEE transactions on semiconductor manufacturing*, 32(2):190–198.

[174] Saqlain, M., Jargalsaikhan, B., and Lee, J. Y. (2019). A voting ensemble classifier for wafer map defect patterns identification in semiconductor manufacturing. *IEEE Transactions on Semiconductor Manufacturing*, 32(2):171–182.

[175] Saxena, A., Goebel, K., Simon, D., and Eklund, N. (2008). Damage propagation modeling for aircraft engine run-to-failure simulation. In *2008 International Conference on Prognostics and Health Management*, pages 1–9.

[176] Scheibelhofer, P., Gleispach, D., Hayderer, G., and Stadlober, E. (2016). A methodology for predictive maintenance in semiconductor manufacturing. *Austrian Journal of Statistics*, 41:161–173.

[177] Schlosser, T., Beuth, F., Friedrich, M., and Kowerko, D. (2019). A novel visual fault detection and classification system for semiconductor manufacturing using stacked hybrid convolutional neural networks. In *2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1511–1514. IEEE.

[178] Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117.

[179] Selvaraju, R. R., Cogswell, M., Das, A., Vedantam, R., Parikh, D., and Batra, D. (2017). Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE international conference on computer vision*, pages 618–626.

[180] Shang, C., Yang, F., Huang, D., and Lyu, W. (2014). Data-driven soft sensor development based on deep learning technique. *Journal of Process Control*, 24(3):223–233.

[181] Shankar, N. and Zhong, Z. (2005). Defect detection on semiconductor wafer surfaces. *Microelectronic Engineering*, 77(3):337–346.

[182] Shen, B., Yao, L., and Ge, Z. (2020). Nonlinear probabilistic latent variable regression models for soft sensor application: From shallow to deep structure. *Control Engineering Practice*, 94:104198.

[183] Shim, J. and Kang, S. (2022). Domain-adaptive active learning for cost-effective virtual metrology modeling. *Computers in Industry*, 135.

[184] Shim, J., Kang, S., and Cho, S. (2020). Active learning of convolutional neural network for cost-effective wafer map pattern classification. *IEEE Transactions on Semiconductor Manufacturing*, 33(2):258–266.

[185] Sriperumbudur, B. K., Fukumizu, K., Gretton, A., Schölkopf, B., and Lanckriet, G. R. G. (2009). On integral probability metrics, ϕ-divergences and binary classification.

[186] Su, A.-J., Jeng, J.-C., Huang, H.-P., Yu, C.-C., Hung, S.-Y., and Chao, C.-K. (2007). Control relevant issues in semiconductor manufacturing: Overview with some new results. *Control Engineering Practice*, 15(10):1268–1279.

[187] Sugiyama, M., Nakajima, S., Kashima, H., Buenau, P., and Kawanabe, M. (2007). Direct importance estimation with model selection and its application to covariate shift adaptation. In Platt, J., Koller, D., Singer, Y., and Roweis, S., editors, *Advances in Neural Information Processing Systems*, volume 20. Curran Associates, Inc.

[188] Sun, B., Feng, J., and Saenko, K. (2016). Return of frustratingly easy domain adaptation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30.

[189] Sun, B. and Saenko, K. (2016). Deep coral: Correlation alignment for deep domain adaptation. In *European conference on computer vision*, pages 443–450. Springer.

[190] Sun, S. (2013). *Neural computing and applications*, 23(7):2031–2038.

[191] Susto, G. A. and Beghi, A. (2012). Least angle regression for semiconductor manufacturing modeling. In *2012 IEEE International Conference on Control Applications*, pages 658–663. IEEE.

[192] Susto, G. A., Schirru, A., Pampuri, S., Beghi, A., and De Nicolao, G. (2018). A hidden-gamma model-based filtering and prediction approach for monotonic health factors in manufacturing. *Control Engineering Practice*, 74:84–94.

[193] Susto, G. A., Schirru, A., Pampuri, S., De Nicolao, G., and Beghi, A. (2012). An information-theory and virtual metrology-based approach to run-to-run semiconductor manufacturing control. In *2012 IEEE International Conference on Automation Science and Engineering (CASE)*, pages 358–363.

[194] Susto, G. A., Schirru, A., Pampuri, S., McLoone, S., and Beghi, A. (2015). Machine learning for predictive maintenance: A multiple classifier approach. *IEEE Transactions on Industrial Informatics*, 11(3):812–820.

[195] Sze, S. and Lee, M. (2012). *Semiconductor Devices: Physics and Technology*. Semiconductor Devices, Physics and Technology. John Wiley & Sons.

[196] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. (2016). Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826.

[197] Tachet des Combes, R., Zhao, H., Wang, Y.-X., and Gordon, G. J. (2020). Domain adaptation with conditional distribution matching and generalized label shift. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. F., and Lin, H., editors, *Advances in Neural Information Processing Systems*, volume 33, pages 19276–19289. Curran Associates, Inc.

[198] Taigman, Y., Polyak, A., and Wolf, L. (2017). Unsupervised cross-domain image generation. *arXiv*, abs/1611.02200.

[199] Tan, M. and Le, Q. V. (2020). Efficientnet: Rethinking model scaling for convolutional neural networks. abs/1905.11946.

[200] Tedesco, S., Susto, G. A., Gentner, N., Kyek, A., and Yang, Y. (2021). A scalable deep learning-based approach for anomaly detection in semiconductor manufacturing. In *2021 Winter Simulation Conference (WSC)*, pages 1–12.

[201] Theis, L., van den Oord, A., and Bethge, M. (2016). A note on the evaluation of generative models. In *International Conference on Learning Representations (ICLR 2016)*, pages 1–10, San Juan, Puerto Rico. International Conference on Learning Representations (ICLR 2016).

[202] Thill, M., Konen, W., Wang, H., and Bäck, T. (2021). Temporal convolutional autoencoder for unsupervised anomaly detection in time series. *Applied Soft Computing*, 112.

[203] Tsai, Y.-H. H., Yeh, Y.-R., and Wang, Y.-C. F. (2016). Heterogeneous domain adaptation with label and structure consistency. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2842–2846.

[204] Tsutsui, T. and Matsuzawa, T. (2019). Virtual metrology model robustness against chamber condition variation using deep learning. *IEEE Transactions on Semiconductor Manufacturing*, 32(4):428–433.

[205] van den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A. W., and Kavukcuoglu, K. (2016). Wavenet: A generative model for raw audio. *arXiv*, abs/1609.03499.

[206] van der Maaten, L. and Hinton, G. (2008). Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(86):2579–2605.

[207] Vapnik, V. (2000). *The Nature of Statistical Learning Theory*. Statistics for Engineering and Information Science. Springer.

[208] Vinyals, O., Blundell, C., Lillicrap, T., Kavukcuoglu, K., and Wierstra, D. (2016). Matching networks for one shot learning. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, page 3637–3645, Red Hook, NY, USA. Curran Associates Inc.

[209] Vishnu, T., Gupta, P., Malhotra, P., Vig, L., and Shroff, G. (2018). Recurrent neural networks for online remaining useful life estimation in ion mill etching system. In *Proceedings of the Annual Conference of the PHM Society, Philadelphia, PA, USA*, volume 22.

[210] Wan, J., Pampuri, S., O'Hara, P. G., Johnston, A. B., and McLoone, S. (2014). On regression methods for virtual metrology in semiconductor manufacturing. In *25th IET Irish Signals Systems Conference 2014 and 2014 China-Ireland International Conference on Information and Communications Technologies (ISSC 2014/CIICT 2014)*, pages 380–385.

[211] Wang, D., Gong, C., and Liu, Q. (2019). Improving neural language modeling via adversarial training. In Chaudhuri, K. and Salakhutdinov, R., editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 6555–6565. PMLR.

[212] Wang, M. and Deng, W. (2018). Deep visual domain adaptation: A survey. *Neurocomputing*, 312:135–153.

[213] Wang, W., Wang, C., Wang, Z., Yuan, M., Luo, X., Kurths, J., and Gao, Y. (2022). Abnormal detection technology of industrial control system based on transfer learning. *Applied Mathematics and Computation*, 412.

[214] Warde-Farley, D. and Bengio, Y. (2017). Improving generative adversarial networks with denoising feature matching. In *ICLR*.

[215] Wilson, A. C., Roelofs, R., Stern, M., Srebro, N., and Recht, B. (2017). The marginal value of adaptive gradient methods in machine learning. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

[216] Wilson, G. and Cook, D. J. (2020). A survey of unsupervised deep domain adaptation. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 11(5).

[217] Wu, M.-J., Jang, J.-S. R., and Chen, J.-L. (2014). Wafer map failure pattern recognition and similarity ranking for large-scale data sets. *IEEE Transactions on Semiconductor Manufacturing*, 28(1):1–12.

[218] Wu, S., Jiang, Y., Luo, H., and Yin, S. (2021). Remaining useful life prediction for ion etching machine cooling system using deep recurrent neural network-based approaches. *Control Engineering Practice*, 109:104748.

[219] Wu, X., Chen, J., Xie, L., Chan, L. L. T., and Chen, C.-I. (2020). Development of convolutional neural network based gaussian process regression to construct a novel probabilistic virtual metrology in multi-stage semiconductor processes. *Control Engineering Practice*, 96.

[220] Xiao, D., Huang, Y., Qin, C., Liu, Z., Li, Y., and Liu, C. (2019). Transfer learning with convolutional neural networks for small sample size problem in machinery fault diagnosis. *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science*, 233(14):5131–5143.

[221] Xie, X., Chen, J., Li, Y., Shen, L., Ma, K., and Zheng, Y. (2020). Self-supervised cycle-gan for object-preserving image-to-image domain adaptation. In Vedaldi, A., Bischof, H., Brox, T., and Frahm, J.-M., editors, *Computer Vision – ECCV 2020*, pages 498–513, Cham. Springer International Publishing.

[222] Xu, C., Tao, D., and Xu, C. (2013). A survey on multi-view learning.

[223] Yan, X., Hu, S., Mao, Y., Ye, Y., and Yu, H. (2021). Deep multi-view learning methods: A review. *Neurocomputing*, 448:106–129.

[224] Yang, S., Song, G., Jin, Y., and Du, L. (2020). Domain adaptive classification on heterogeneous information networks. In Bessiere, C., editor, *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, pages 1410–1416. International Joint Conferences on Artificial Intelligence Organization.

[225] Yang, Y. and Xu, Z. (2020). Rethinking the value of labels for improving class-imbalanced learning. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. F., and Lin, H., editors, *Advances in Neural Information Processing Systems*, volume 33, pages 19290–19301. Curran Associates, Inc.

[226] Yosinski, J., Clune, J., Bengio, Y., and Lipson, H. (2014). How transferable are features in deep neural networks? abs/1411.1792.

[227] Yu, F. and Koltun, V. (2015). Multi-scale context aggregation by dilated convolutions. *arXiv*, abs/1511.07122.

[228] Yu, J., Shen, Z., and Zheng, X. (2020). Joint feature and label adversarial network for wafer map defect recognition. *IEEE Transactions on Automation Science and Engineering*, 18(3):1341–1353.

[229] Yu, N., Xu, Q., and Wang, H. (2019). Wafer defect pattern recognition and analysis based on convolutional neural network. *IEEE Transactions on Semiconductor Manufacturing*, 32(4):566–573.

[230] Yu, Q., Li, L., Zhao, H., Liu, Y., and Lin, K.-Y. (2021). Evaluation system and correlation analysis for determining the performance of a semiconductor manufacturing system. *Complex System Modeling and Simulation*, 1(3):218–231.

[231] Yuan, X., Huang, B., Wang, Y., Yang, C., and Gui, W. (2018). Deep learning-based feature representation and its application for soft sensor modeling with variable-wise weighted sae. *IEEE Transactions on Industrial Informatics*, 14(7):3235–3243.

[232] Yuan, X., Li, L., and Wang, Y. (2020). Nonlinear dynamic soft sensor modeling with supervised long short-term memory network. *IEEE Transactions on Industrial Informatics*, 16(5):3168–3176.

[233] Zhang, A., Wang, H., Li, S., Cui, Y., Liu, Z., Yang, G., and Hu, J. (2018a). Transfer learning with deep recurrent neural networks for remaining useful life estimation. *Applied Sciences*, 8(12).

[234] Zhang, C., Gao, X., Li, Y., and Feng, L. (2019). Fault detection strategy based on weighted distance of k nearest neighbors for semiconductor manufacturing processes. *IEEE Transactions on Semiconductor Manufacturing*, 32(1):75–81.

[235] Zhang, J., Li, Y., Tian, J., and Li, T. (2018b). Lstm-cnn hybrid model for text classification. In *2018 IEEE 3rd Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*, pages 1675–1680.

[236] Zhang, K., Schölkopf, B., Muandet, K., and Wang, Z. (2013). Domain adaptation under target and conditional shift. In *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ICML'13, page 819–827. JMLR.org.

[237] Zhao, S., Li, B., Yue, X., Gu, Y., Xu, P., Hu, R., Chai, H., and Keutzer, K. (2019). Multi-source domain adaptation for semantic segmentation. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

[238] Zhao, S., Yue, X., Zhang, S., Li, B., Zhao, H., Wu, B., Krishna, R., Gonzalez, J. E., Sangiovanni-Vincentelli, A. L., Seshia, S. A., and Keutzer, K. (2022). A review of single-source deep unsupervised visual domain adaptation. *IEEE Transactions on Neural Networks and Learning Systems*, 33(2):473–493.

[239] Zhu, J., Park, T., Isola, P., and Efros, A. A. (2017). Unpaired image-to-image translation using cycle-consistent adversarial networks. *arXiv*, abs/1703.10593.

[240] Zhu, W., Braun, B., Chiang, L. H., and Romagnoli, J. A. (2021). Investigation of transfer learning for image classification and impact on training sample size. *Chemometrics and Intelligent Laboratory Systems*, 211:104269.

[241] Zhuang, F., Qi, Z., Duan, K., Xi, D., Zhu, Y., Zhu, H., Xiong, H., and He, Q. (2021). A comprehensive survey on transfer learning. *Proceedings of the Institute of Radio Engineers*, 109(1):43–76.

[242] Zou, Y., Yu, Z., Kumar, B. V. K. V., and Wang, J. (2018). Unsupervised domain adaptation for semantic segmentation via class-balanced self-training. In *ECCV*.

Acknowledgments

I want to thank Infineon Technologies AG for funding this research and for giving me the opportunity to pursue my long-awaited goal of writing a doctoral thesis. I express my gratitude to Hans Ehm for giving me the freedom to follow my own research direction and for his trust in my decisions. Your open-mindedness and dedication towards new research topics and young researchers and co-workers is unique. My special thanks goes to Olaf Herzog who not only initiated the topic of the thesis but was always available for deep dive discussion and willing to teach me more about semiconductor manufacturing. You are a great, pragmatic and smart boss. Thank you for your trust and support throughout this journey.

My deepest gratitude goes to my company supervisor Andreas Kyek. Thanks for all the great discussions and for sharing your ideas and knowledge with me. Your creativity and intelligence is inspiring and outstanding. Without your guidance and friendship this journey would not have been as successful and wonderful as it was. Thank you for going above and beyond!

A big thank you also goes out to my other colleagues at Infineon, my PhD years would not have been so pleasant without all of you! I want to thank Yao Yang for her advice and all the fruitful discussions throughout the last four years. I also want to thank Christa Bergmaier for being the heart of the team and always making me laugh and feel at home. I wish to extend my gratitude to Thomas Ponsignon, Tim Lauer, William Parker, Marco Ratusny and Lorenzo Servadei for being on this PhD journey together with me. Tim, I am looking forward to continue working with you and conquer new research and application fields. I also want to thank Johannes Sturm and Joon Khim Low for the interesting tasks we worked on together and all the insights and learnings I had. Our cooperation will definitely continue and grow.

Equally important, I want to thank University of Padua and my two advisors Alessandro Beghi and Gian Antonio Susto for their support and guidance. A very special thanks goes to Gian Antonio who has shown me his dedication, unbelievable work ethic and endless knowledge. Further thanks goes to Luciano Lorenti and Mattia Carletti and all the other people I met during the last 3 years. I consider myself blessed to get the chance to work with such skilled, creative, knowledgeable but also warm, nice, trustful and special people.

Finally I want to thank some people outside the academic world for their support: Foremost, I want to thank my parents Wilfried and Anita and my brother Daniel for always being there for me, giving advice when I needed it and above all always believing in me. Their love and support helped me through all ups and downs and made me the person I am today.

I also want to thank all my friends, for being part of this journey and even long before. Michael, thanks for inspiring and challenging me and getting me out of my comfort zone.

Last, I want to thank Mirfand. Your love, trust and believe give me the strength and the consistency to pursue my dreams and to be the best version of myself. Thank you for always being at my side, you are my future!