Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

University of Padua

Department of Information Engineering

Master Thesis in ICT for Internet and Multimedia

A Domain Adaptation approach for sequence modeling through Deep Learning in semiconductor manufacturing: adversarial training setup with Temporal Convolutional Network and Long-Short Term Memory models

Supervisor
Prof. Gian Antonio Susto
Co-advisor
Natalie Gentner

Master Candidate
Filippo Dalla Zuanna

Academic Year 2021/2022
Graduation Date 11/07/2022

Abstract

The aim of this work is to develop Deep Learning strategies to extend previous research [1, 2] concerning a task related to Virtual Metrology field in a semiconductor manufacturing. This intends to compare the previously developed model and other models more suited to the time series data under examination. The strategy used in this problem is based on Domain Adaptation through an adversarial training setup; the approach is recent and promising, allowing for better management in the absence of large amounts of data and making previously trained models reusable in similar scenarios.
This research is encouraged for its effects in this field; further improving the performance of the models used means savings on production costs, since it would limit conformity tests which lead to product damage. This also leads to a possible saving of materials, an important aspect from an ecological point of view. Finally, it is increasingly necessary to consider models that use less data or are more suitable for similar processes, as in the case of Domain Adaptation techniques. The latter aspect is relevant because the data is expensive to obtain in the considered application.

Sommario

Lo scopo di questo lavoro è sviluppare delle strategie di *Deep Learning* per estendere la ricerca precedente [1, 2] riguardante una mansione relativa all'ambito di *Virtual Metrology* in una produzione di semiconduttori. Questo vuole essere un confronto tra il modello sviluppato in precedenza e alcuni modelli più adatti ai dati di serie temporali in esame. La strategia utilizzata in questo problema si basa sul *Domain Adaptation* attraverso un'impostazione di *Adversarial Training*; l'approccio è recente e promettente, permettendo una miglior gestione in assenza di grandi quantità di dati e rendendo i modelli precedentemente addestrati riutilizzabili in scenari simili.

Questa ricerca è incoraggiata per gli effetti che può portare in questo campo; migliorare ulteriormente le prestazioni dei modelli utilizzati significa risparmiare sui costi di produzione, poiché permetterebbe di limitare test di conformità che danneggiano i prodotti. Ciò comporta anche un possibile risparmio di materiali, aspetto importante dal punto di vista ecologico. Infine, è sempre più necessario considerare l'uso di modelli che utilizzano meno dati o sono più adatti a processi simili, come nel caso delle tecniche di *Domain Adaptation*. Quest'ultimo aspetto è rilevante perché i dati sono costosi da ottenere nell'applicazione considerata.

Contents

Listing of figures

Listing of tables

Listing of acronyms

ML	Machine Learning
DL	Deep Learning
VM	Virtual Metrology
DA	Domain Adaptation
TL	Transfer Learning
FDC	Fault Detection and Classification
PdM	Predictive Maintenance
R2R	Run-to-Run
ULSI	Ultra-Large Scale Integration
CV	Cross-validation
MSE	Mean Squared Error
ME	Maximal Residual Error
EV	Explained Variance Regression Score
R2	R-square Regression Score
NN	Neural Network
ANN	Artificial Neural Network
ELU	Exponential Linear Unit
ReLU	Rectified Linear Unit
CNN	Convolutional Neural Network
1DCNN	One Dimesional Convolutional Neural Network
TCN	Temporal Convolutional Network

RNN	Recurrent Neural Network
NLP	Natural Language Processing
LSTM	Long Short Term Memory Network
GAN	Generative Adversarial Network
WGAN	Wasserstein Generative Adversarial Network
DANN	Domain Adversarial Neural Network
DBAM	DANN-Based Alignment Model

1 Introduction

Nowadays, the industrial sector is facing a continuous increase in the complexity of the processes aimed at improving productivity and integrating the new methodologies brought about by the digital revolution. The semiconductor manufacturing sector is no exception because its processes are extremely complex so spending resources to improve them is not surprising. This includes integrating recent strategies that leverage the collection of large amounts of data to extract useful information, used for different types of optimizations. These methods include statistical techniques and Machine Learning, with Deep Learning also emerging strongly in recent years. The latter is favoured by the promising performances shown in various fields and the greater availability of data and computational resources than in the past.

In this work, the problem of Virtual Metrology is taken into consideration, dealing specifically with the Etching process in semiconductor manufacturing. This work aims to study Deep Learning solutions to improve previous research on this specific problem [1, 2], expanding the analysis with the use of architectures more suited to the data type related to the problem. This configuration also deals with Domain Adaptation through an adversarial training setup for solving this aspect. This is necessary as each of the machines involved in the process, although the same, generates data that follows different distributions. This disparity could be caused by imperfections in the machines, a slightly different environment while in function, non-identical settings or different usage times. This forces to manage this aspect since, otherwise, it would be necessary to develop a different model for each machine,

wasting resources and losing effectiveness.
As the first aspect of the work, the elements necessary to understand the industrial process and the overview of the main topics of this work are present in the second chapter. Then, in the third one, the contents focus on comparing the new models with the benchmark in the regression aspect of the Virtual Metrology task. This includes the description of the various architectures, the dataset details, the steps of the methodology for solving the prediction problem and the results obtained. In the final chapter, the work handles the Domain Adaptation aspect of the problem, including the architecture description for the adversarial training with its elements and the steps for solving the Domain Adaptation problem. Finally, the results of the various configurations are presented and commented on.
This work has been done in collaboration with *Infineon*; it provided both the data and the resources to train and evaluate the various models. Due to the NDA agreement, the signal plots don't show some data information.

2 Background

This chapter talks about the necessary information needed to understand the field of application and the general topics concerning the task. So, the etching process and virtual metrology are described to understand the motivations behind the need for this task resolution. Accordingly, a general overview of the data is presented. Then, the general topic of Deep Learning regarding the models used for solving is introduced. Finally, Domain Adaptation is described, with its concepts to make the method used more generally.

2.1 Semiconductor Manufacturing

The field of semiconductor manufacturing is one of the most technologically advanced industrial sectors [9]; every process involved is extremely complex and costs a lot, together with the infrastructures that allow these processes to be carried out. This sector is one of the most important for mankind since it is responsible for the production of the basic elements that make up lots of objects present in our everyday life. Personal computers, smartphones and cars are some of an infinite number of examples. Given such premises, it is not surprising that semiconductor manufacturing companies are spending effort and resources to improve quality and optimize the processes, heading towards smaller, faster and higher quality devices along with cost reduction. These optimizations largely involve the use of tools for process control and analysis, such as Machine Learning used for Virtual Metrology (VM) systems, Fault Detection (FDC) systems, Predictive Maintenance (PdM) systems and Run-to-Run (R2R) control [10]. All of these technologies have spread in the past few years in semiconductor manufacturing facilities to improve productivity and decrease costs. The entire semiconductor manufacturing process takes usually six to eight weeks from the first stage up to final product shipping; it is performed in highly specialized fabrication plants, showing how is long and complex. The latter is composed of many different steps, starting from the wafer formation; this consist of the creation of a thin (525 - 775 μm) slice of semiconductor material with 125 - 300mm diameter that serves as the substrate for microelectronic devices. Wafers are formed from extremely pure crystalline material, usually silicon, thanks to the Czochralski process. Then, the front end processing relates to the formation of transistor chips on the silicon wafer and is performed in controlled environments known as clean rooms, which have the level of dust, vapours and particles artificially kept at a low level. This is done with air filtering and restricted access policies needed for performing the subprocesses of this category. These are:

- **Wafer cleaning**: cleaning procedures are needed to prepare the wafers for subsequent processes, as Ultra-Large Scale Integration (ULSI) technology is characterized by strict requirements concerning surface smoothness and particle contamination;
- **Deposition**: film deposition is widely used in Integrated Circuits (IC) fabrication and includes dielectric films, composed of silicon dioxide and silicon nitride which serve as isolation, mask and passivation layer, and polysilicon films, which can be used as a conducting layer, semiconductor, or resistor by proper doping with different impurities;
- **Lithography**: several techniques may be used to create ULSI circuit patterns on wafers, like the most common photomask exposition which employs the use of ultraviolet radiation that is transmitted through the clear part of the mask while the opaque part blocks the rest of the radiation. The resist film, being sensitive to the radiation, is then coated on the wafer surface, developing the resist image induced by the mask previously aligned on the wafer;
- **Etching**: described more in detail later.

All the front end processes are repeated several times to produce multiple interconnected layers on the wafer surface. After them, the products enter the testing phase looking for functional defects, subdivided into parametric and electrical tests. The firsts are performed on ad-hoc structures, called TAG, prepared on the device to monitor the efficiency of process steps and the goodness of the design. These tests consist of electric measurements of physical quantities, such as impedance, capacitance and resistance. The seconds verify that the

behaviour of each device is consistent and within specifications, thanks to electrical testing with sequential measurements; if some value is out of specification range the circuit is flagged as faulty, marking the non-passing quantities and storing that information. Finally, the microchips encounter the process of packaging, which provides electrical connection, protection from mechanical and environmental stress and a thermal path for the heat generated. This is crucial for the performance and the reliability of the chip witch impact consequently the final system.
The description of the phases involved in the production process makes us aware of its complexity and it's easy to understand how high the interest in optimizing the elements of the process can be, in our case with a focus on front end ones.

2.1.1 Etching Process

In semiconductor manufacturing, Etching is a process involved in the fabrication of semiconductor devices which consists of any method that selectively removes material from a thin film on a substrate. This removal creates the circuit arrangement which is defined by a mask resistant to etching formed in a previous process. The elimination of the not protected material can occur by either wet or dry methods. Wet methods consist most of immersing the wafers in a chemical solution or spraying the wafers with them. Instead, dry methods are synonymous with plasma-assisted etching, which denotes several techniques that use plasma in the form of low-pressure discharges, like plasma etching, reactive-ion etching, sputter etching, and high-density plasma etching. These include chemical and physical methods, where in physical ones positive ions bombard the surface at high speed.
In this complex process, the monitoring of the variables involved is crucial for the success of the process. For Plasma etching, such variables are for example gas flows, power and pressure that determine etching properties such as etch rate, uniformity, selectivity and anisotropy. This leads to the need of monitoring and controls these quantities through traditional control methods or statistical ones, such as mathematical models or Virtual Metrology methods [11].

2.2 Virtual Metrology

Virtual Metrology(also called Soft Sensing) refers to the set of methods that estimate quantities that are important for quality control or process control, quantities that are costly or difficult to be measured. These methods exploit the availability of data that are collected

by the machines or information systems during their operation and use them to estimate the costly measure, benefiting from regression modelling techniques. This allows for an estimate without metrology, reducing the number of product tests performed. These measurements are usually taken in metrology stations where multiple measurements on various products are performed after the process is finished. This often leads to functionality problems and, in some cases, the product is destroyed. The costs related to these issues are summed up with the already present measurement costs; for this reason, only a subset of the products or processes are tested.
Examples of literature about the development of techniques for Virtual Metrology are various, even in the field of Semiconductor Manufacturing. A lot of them are based on classical Machine Learning models such as prediction models based on regression algorithms [12] or models based on least square optimization of the kernel like Ridge Regression, Lasso and Efficient net [13]. In [14], tree-based approaches like RF are been also investigated. However, not only ML models are being used; Deep Learning approaches are spreading given their promising results in this and other fields. Artificial Neural Networks are widely adopted in VM like in [15], while more sophisticated architectures have been recently adopted [16] [17] [6].

2.3 The Dataset

The dataset used in this work corresponds to the same of the reference work [1], provided by Infineon experts.
Data is collected from two machines of the same type that are also running in parallel, with a selection of almost two years of activity where the data acquisition characteristics are not changed significantly. These samples contain physical quantities and process information which are collected by the sensors of the machines that operate during the process. They are stored for analysis for different behaviours, such as process control. After that, some samples get data quantities added thanks to a metrology tool that performs multiple measurements directly on the product. Such measurements are, for example, critical dimension and layer thickness; the latter are chosen as labels for the VM prediction task since it shows high correlations to the quantities' evolution during the etching process. This permits to have the data related to the analyzed problem: the samples of the evolution of the process with the corresponding outcome are available, making the use of statistical models for its analysis and resolution possible.

2.4 Deep Learning

2.4.1 The main topic of Machine Learning

The Machine Learning field refers to the study of computer algorithms that can learn a task with the use of data. In this manner, ML programs can perform tasks without being explicitly programmed; computers are brought to learn directly from the data examples provided instead of challenging a human to individuate the solution rule. This is performed by minimizing a performance measure called loss function, which permits training the model and learning the task as much as possible. This approach should ideally lead to the correct management of similar but unseen examples. In many cases, this method turns out to be more effective and efficient.
ML algorithms can be divided into categories depending on the data and the training approach used:

- **Supervised Learning**: Both an input X and the desired output Y are presented to the model, to learn a mapping function $f(x)$ such that $f(X) = Y$, representing the hypothetical rule of the task;

- **Unsupervised Learning**: No output is provided to the algorithm, which is left to discover the regularities in the data without feedback, learning its various properties and being able to discriminate between similar and different examples;

- **Semi-Supervised Learning**: The learning algorithm will work almost completely unsupervised, but with a small representative set of output samples available to guide the extraction of useful information;

- **Reinforcement Learning**: The learner (called the agent) is left in an environment where he can interact with it, making him discover patterns in the environment or perpetrating a goal with the guidance of rewards/punishments.

In this work, a supervised learning approach is used thanks to the availability of labelled data; supervised algorithms are preferable whenever possible, due to their simpler implementation and often better results. More precisely, the task refers to the learning of the rule that connects various quantities measured in the etching process to the layer thickness. This corresponds to a regression task since the output quantity is a real number, instead of a classification problem where the outputs are of finite values.

2.4.2 The subcategory of Deep Learning

Deep Learning refers to a subfield of Machine Learning that uses particular models called Artificial Neural Networks [18]. This kind of model is inspired by the human brain, by exploiting its ability to do extremely complex things with the efficient connection of many simple units, the neurons. So, the logic behind these models is to use a simple processing unit with limited learning capability and connect a lot of them to form a complex configuration. Each of these units can be of a different type depending on the behaviour of the model. For example, it can be a perceptron (also named fully-connected neuron) for general processing, an approximation of the neuron behaviour used in the first ANN models as in 2.1. In

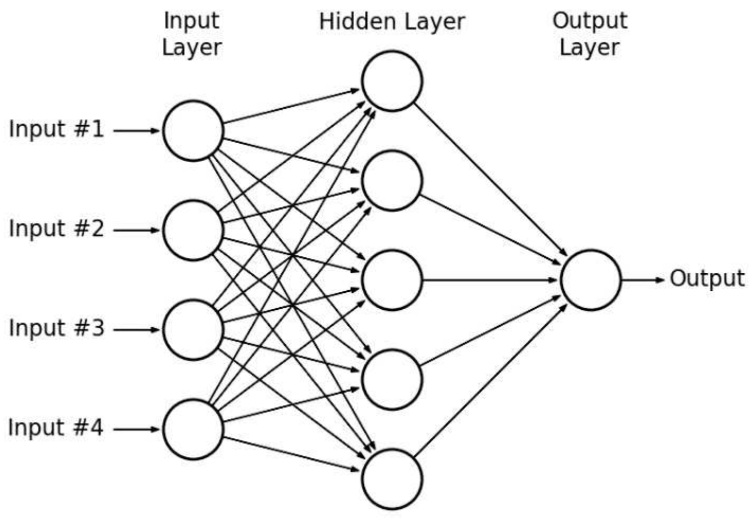


Figure 2.1: ANN architecture example from [3]

image processing, it can be a filter or, for temporal processing, a recurrent cell. Given the huge amount of units, some simplifications are required to reduce the computational effort since the connection of these units together is unfeasible. For this reason, these elements are grouped in layers that are stacked and connected. So, the elements of each layer are linked with the previous and the following one, having no internal connections. This simplifies the model configuration and permits the stack of a lot of these layers, making the network deep, hence the origin of the name Deep Learning.
Some examples of ANNs will be described in later sections.

2.5 Domain Adaptation

2.5.1 The main topic of Transfer Learning

Transfer Learning is the topic that deals with all those applications where the used statistical model has to face particular differences present in subproblems of a general category already learned. It refers to techniques that permit to reuse of general properties of the task learned previously and, starting from them, fit the knowledge in their particular aspects [19]. This is needed since a common assumption when dealing with statistical models is usually not present and leads to a drastic decrease in performance. It refers to the fact that all samples used for learning the task are independent and identically distributed (i.i.d) according to a probability distribution D [20]. So, the discrepancy between relative similar tasks that have differences in distributions, which worsen the performances, are addressed by Transfer Learning. This knowledge transfer is performed to improve the capabilities of a model. Some examples of applications are cancer subtype discovery(medical imaging) [21], building utilization [22], general game playing [23], text classification [24], digit recognition [25] and object recognition/detection [26].

2.5.2 The subcategory of Domain Adaptation

Domain Adaptation refers to a particular case of the more general topic of Transfer Learning. In this case, the domains under consideration have the same feature space but different distributions, while in general Transfer Learning feature space can differ. The domain used for extracting the knowledge takes the name of the source domain, while the domain that needs to be adapted is called the target domain. Domain Adaptation can be solved through three different methods [27]:

- **Instance re-weighting:** Source samples are re-weighted to fit better the samples in the target domain. For estimating those weights, a non-parametric minimization is performed to reduce the gap between the re-weighted source distribution and the target distribution;

- **Subspace alignment:** The optimization of a mapping function is learned to lead to a common subspace where the source and target domains have a similar distribution;

- **Deep alignment:** Deep Learning techniques can be used to align source and target domains, either by minimizing the distance between distributions or by adversarial domain alignment.

In this work, the latest solution is adopted using labelled data(source or target samples). Usually, this alignment is performed in Semi-Supervised and Unsupervised settings, but the use of labels can drastically reduce the task effort.

3 Virtual Metrology task with Deep Learning models

This chapter talks about the description of the models used in this work, 1DCNN regarding the reference work and TCN and LSTM regarding this one. Then the results of the regression task are presented with the methodology that leads to them.

3.1 Deep Learning models

3.1.1 One Dimensional Convolutional Neural Network model

1DCNN models take their name from a particular behaviour of the more general Convolutional Neural Network model. In this particular case, the convolution instead of being applied in two dimensions, for example in image data, is performed only in one dimension. The latter corresponds to the time dimension in our specific case to extract features from the evolution in time of the signals. Since we are dealing with time series another implementation must be in place: the convolution must be causal since in this way each feature depends only on the present and past information and does not include future information that would be conceptually wrong. More details about 1DCNN can be found in [4].

Convolutional Neural Networks

The CNN models are inspired by the connectivity model between neurons present in the animal visual cortex. For this reason, they were designed originally for image processing. Every cortical neuron responds to the stimulus of a precise characterization of the visual field called the receptive field, such as an edge with a certain orientation. The receptive fields of different neurons partially overlap in such a way as to cover all the characteristics of the visual field. In a convolutional layer, the kernel filters are equated to the neurons in the biological model where each of them generates a feature map being convoluted with the entire input. In the case of a bi-dimensional input I and a two-dimensional kernel filter K, the discrete convolution S of I and K can be expressed as:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n). \tag{3.1}$$

This feature map represents how much the filter is stimulated in every location of the input. Hence the target of the training of these filters is to set their parameters in a way that they can extract salient characteristics of the inputs.
Here in 3.1 an example of the CNN model to better understand his general elements. This

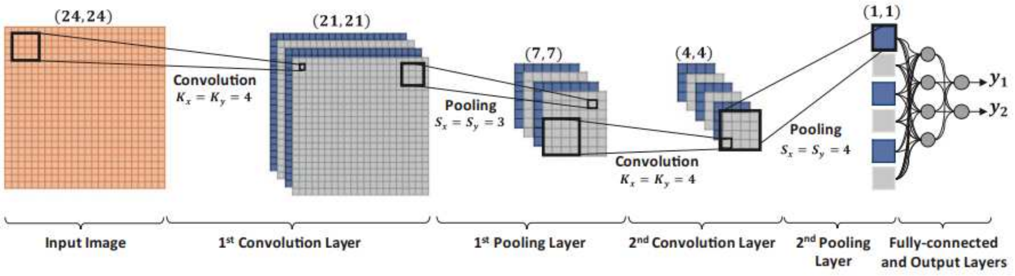


Figure 3.1: Example of a CNN architecture from [4].

simple network consists of two convolution and two pooling layers with 4 and 6 neurons. The output of the last pooling layer is processed by a single fully-connected layer and followed by the output layer that produces the classification output. The interconnections feeding the convolutional layers are assigned by weighting filters (w) having a kernel size of (K_x, K_y). The convolution takes place within the image boundaries; therefore, the feature map dimension is reduced by the $(K_x - 1, K_y - 1)$ pixels from the width and height, respectively. The subsampling factors (S_x, S_y) are set in advance in the pooling layers, elements used for reducing the dimension of the feature maps. The kernel sizes corresponding to the two convolution layers were set to $(K_x = K_y = 4)$, while the subsampling factors are set as $(S_x = S_y = 3)$ for the first pooling layer and $(S_x = S_y = 4)$ for the second one. These values make that the outputs of the last pooling layer (i.e. the input to the fully-connected layer) are scalars (1×1). The output layer consists of two fully-connected neurons corresponding to the number of classes to which the image is categorized.

3.1.2 Temporal Convolutional Network model

The TCN models refer to a family of models that exploit the characteristics of CNNs and adapt them to use time sequences. To satisfy this use, the convolutions in the architecture are causal, meaning that there is no information leakage from future to past. In addition, the architecture can take a sequence of any length and map it to an output sequence of the same length, just as with a Recurrent Neural Network model, the basic Deep Learning model for time sequence processing. To achieve the first point, the TCNs use convolutions where output at time t is convolved only with elements from time t and earlier in the previous layer or input. To accomplish the second point, the TCNs use a 1DCNN architecture where each feature map has the same length as the input layer. Also, zero padding of length (kernel size $- 1$) is added to keep the subsequent layers of the same length as the previous ones. For these reasons, as tell in [28], TCNs can be seen as the sum of 1DCNN and causal convolution. This would make the TCN model almost equal to the previous presented 1DCNN model but this is not true in practice. Indeed, TCNs refer to the architecture that implements other features in addition to the ones presented above. These features are the dilated convolution and the residual connections, taken from WaveNet model [5].

Dilated Convolution

A dilated convolution is a convolution where the filter is applied over an area larger than its length by skipping input values with a certain step. The network can perform the convolution on a coarser scale, like pooling or convolutions with a stride greater than 1, but the output has the same size as the input. Formally, for a $1 - D$ sequence input $x \in \mathbb{R}^n$ and a filter $f : \{0, ..., k - 1\} \rightarrow \mathbb{R}$, the dilated convolution operation F on element s of the sequence is defined as:

$$F(s) = (x *_d f)(s) = \sum_{i=0}^{k-1} f(i) \cdot x_{s-d\cdot i} \tag{3.2}$$

where d is the dilation factor, k is the filter size, and $s - d \cdot i$ accounts for the direction of the past. As a special case, dilated convolution with dilation of 1 yields the standard convolution. 3.2 depicts dilated causal convolutions for dilations 1, 2, 4, and 8. Stacked dilated

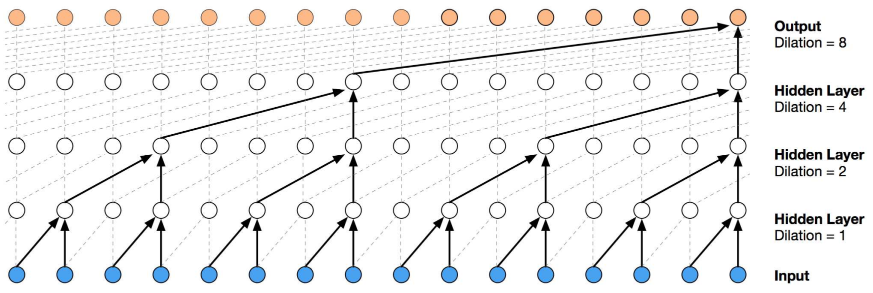


Figure 3.2: Stacked Dilated Convolution from [5].

convolutions enable networks to have very large receptive fields with just a few layers, while preserving the input resolution throughout the network as well as computational efficiency. Usually, the dilation doubles up to a certain limit in a layer and then it is repeated with the stacking of new layers. This leads to two benefits. First, exponentially increasing the dilation factor results in exponential receptive field growth with depth. Second, stacking these blocks further increases the model capacity and the receptive field size.

Residual Connections

The residual connections consist of a solution that permits the network's layers to learn modifications to the identity mapping rather than the entire transformation. This has repeatedly been shown to benefit very deep networks since it preserves more information during the flow into the model. In detail, the output features are determined by applying an activation function to the sum of the input and the transformed input:

$$output = Activation(x + F(x)). \tag{3.3}$$

3.1.3 Long-Short Term Memory network

The LSTM models are an extension of the more general Recurrent Neural Network models and improve some of its aspects through optimizations of the information process.
The RNN models are a particular kind of ANN that focuses on processing data that correlates in the time domain, then being able to process any time series data type. This is possible because the model introduces recurrent connections that permit the elaboration of the information at timestamp t along with the information stored through the elaboration of previous data. More specifically, the hidden representation of the data h_t is calculated using the current input x_t, the previous representation h_{t-1} and the current timestamp t in this way:

$$h_t = f(h_{t-1}, x_t, t). \tag{3.4}$$

This permits the network to remember past information, which makes them a great tool when working with temporal data or NLP.
LSTMs introduce some improvements in the management of the past information to overcome the lack of standard RNNs that limit their learning about short-term and not long-term dependencies. This is done by adding a set of gated units to the basic structure of an RNN to manage the store of the information [29]. The use of those gates drastically reduces the vanishing and exploding gradient problem which occurs in RNNs. In particular, those gates are of three different categories:

1 Input gates, which manage the input utilization from the memory cell;

2 Output gates, which weight how much the current value stored in the memory cell is present in output;

3 Forget Gates, which manage how much the current value stored has to be forgot.

More in detail, gates make use of the sigmoid function σ to determine when to add or remove information. The forget gate decide when to keep or discard information by looking at the current input x_t and the previous activation h_{t-1}:

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f) \tag{3.5}$$

where W_f is the weight matrix for the forget gate and b_f is the bias term. The input gate will determine which new information is going to be stored in the cell state. Which elements to

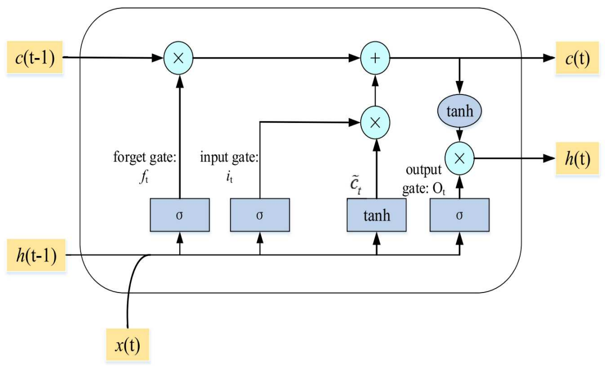


Figure 3.3: LSTM cell overview from [6].

store i_t is determined as follows:

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_u) \tag{3.6}$$

while the values to store $\hat{C}_t$ is computed using hyperbolic tangent:

$$\hat{C}_t = tanh(W_c[h_{t-1}, x_t] + b_c). \tag{3.7}$$

The cell update just multiplies the old state by the forgetting factor, and adds the new input information:

$$C_t = f_t \cdot C_{t-1} + i_t \cdot \hat{C}_t. \tag{3.8}$$

The output gate, which decide what to propagate forward in the output, is computed as:

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o) \tag{3.9}$$

where $h_t = o_t \cdot tanh(C_t)$. More details about these equations can be found in [30].

3.2 Virtual Metrology Regression task

As said previously, the regression task studied is the one that tries to predict the layer thickness given some quantities measured during the etching process. This is done as in the reference work [1], from which the benchmark model is taken as comparison starting point with some procedure characteristics. This is used as a reference to train and test the TCN and LSTM models.

3.2.1 Performance metrics

For the performance comparison more metrics are being used: Mean absolute error (MAE), maximal residual error (ME), explained variance regression score (EV) and R-square (coefficient of determination) regression score function (R2). Mean absolute error is used not only for measuring performance but also as training loss. The equations are:

$$MAE(y^{true}, y^{pred}) = \sum_{i=1}^{n} |y_i^{true} - y_i^{pred}| \tag{3.10}$$

$$ME(y^{true}, y^{pred}) = \max_i |y_i^{true} - y_i^{pred}| \tag{3.11}$$

$$Var(y) = \sum_{i=1}^{n} (y_i - \bar{y})^2 \tag{3.12}$$

$$EV(y^{true}, y^{pred}) = 1 - \frac{Var(y^{true} - y^{pred})}{Var(y^{true})} \tag{3.13}$$

$$R2(y^{true}, y^{pred}) = 1 - \frac{\sum_{i=1}^{n}(y_i^{true} - y_i^{pred})^2}{Var(y^{true})} \tag{3.14}$$

where y^{true} are the dataset labels, y^{pred} are the predictor outcomes, $\bar{y}$ stands for the mean value and Var for the variance, the square of the standard deviation. R2 and EV are equal if the mean error is 0, then both are taken into account to assure an unbiased evaluation. For MAE and ME lower value is better, instead for EV and R2 higher value is better with 1 as upper bound limit.

3.2.2 Dataset details

The 33 measured quantities contained in the dataset are coming from two different equipment, corresponding to the source and the target domains respectively. Each time series has the layer thickness as the label, not considered in the dimension of the dataset. The raw sensor signals got some preprocessing steps; an outlier removal procedure is applied, then constant features are deleted and, finally, data get normalized with a min-max scaler. Also, an equal distributed upsampling of timestamps was used since time series have different lengths caused by the endpoint detection present in the process, which cuts the series. In this way, time series are generated of equal length, a characteristic that must be present to use data with

the models. Additional feature engineering approaches are not applied since the models face the raw time series directly. However, the data collection is been limited to only one process to simplify the dataset, avoiding prediction on multiple ones. Therefore, only one label and one group of products are considered for the prediction.
The B dataset is composed of 1271 samples which form the source domain, the one that should consent to the extraction of the solution to the VM problem. The A dataset is composed of 1209 samples which form the target domain, the one that should be adapted for being solved with the solution developed for the source domain dataset B. Each time sequence has a length of 1024 given by the equal distributed upsampling performed in preprocessing steps. Then, the number of quantities measured is 33, as told previously.

Cross-validation

A different number of datasets are created for cross-validation, a procedure used to validate the final solutions, while only one of these datasets is used for the intermediate steps. This method consists of applying the resolution of the problem on different distributions of the dataset into training, validation and test sets, intending to average the results and reduce the dependence on the specific distribution of the dataset. The training set consists of samples that the model uses for learning the specific task. The validation set consists of samples that guide the training into a more general solution. The test set is used to evaluate the final performances.
For this specific task, 5 different datasets are created. The data is divided into 5 groups where 4 of them are used as training samples. The other one is divided equally into validation and test sets. This happens before the normalization, avoiding information leakage from training data distribution to validation and test ones. The min-max scaler parameters are calculated on training samples and applied directly to the other. In this way, the training set doesn't contain information about the normalization parameters of the others.

Datasets dimensions

In summarizing, the datasets have the following dimensions, excluding the labels:

- **Train dataset A**: $[968 : 1024 : 33]$;
- **Validation dataset A**: $[120 : 1024 : 33]$;
- **Test dataset A**: $[121 : 1024 : 33]$;

- **Train dataset B**: $[1017 : 1024 : 33]$;
- **Validation dataset B**: $[127 : 1024 : 33]$;
- **Test dataset B**: $[127 : 1024 : 33]$.

3.2.3 Benchmark predictor model

The 1DCNN predictor model taken as the reference has the following elements:

1. 1DCNN layer with 16 filters followed by a batch normalization layer;
2. 1DCNN layer with 8 filters followed by a max-pooling layer with 4 as pool dimension and a batch normalization layer;
3. Fully connected layer with 16 units applied to the flattened version of the features extracted previously, followed by a batch normalization layer;
4. Output layer(1 fully connected unit) that use sigmoid as activation function.

The 1DCNN layers have a kernel size of 17 and a stride of 2. The stride halves the time dimension at every layer. 1DCNN layers and the fully connected layer use Exponential Linear Unit (ELU) as the activation function, a slightly modified version of the more common Rectified Linear Unit (ReLU). This configuration owns about $20k$ parameters, resulting in a simple and lightweight model.

3.2.4 Training procedure parameters maintained

Some of the characteristics of the training procedure are kept untouched. This is done because the assumption is that the reference work has already encountered an optimization process and so there is no need to touch the parameters of the benchmark training. Given this, some parameters have to remain the same for pursuing that since the work focuses on comparing models. Otherwise, the differences in the training procedure could lead to a not meaningful comparison.
The first of these parameters is the training epochs, fixed at 300, in a way to train the model for the same amount of iterations. The second is the optimizer, which guarantees that the optimization process happens with the same tool. This is the Adam optimizer[31], one of the most used thanks to the excellent results of his adaptive learning rate techniques. Then,

the learning rate value and the learning rate schedule are maintained the same, with the usual aim of preserving the optimization process equality; the learning rate started at 0.00001 and undergoes an exponential learning rate decrease after 50 epochs. The third is the schedule of the early stopping, which has $min_\delta = 0.00001$ and $patience = 20$. Finally, the batch size is fixed to 32; this parameter influences the calculation of the gradient in the optimization process and, for this reason, is kept fixed. However, this is a more flexible parameter that in general can be optimized without messing with the comparison, but in this work, we choose to keep the same.

3.2.5 Hyperparameters tuning

In ML and also DL models, some of their characteristics are not fixed and determine different behaviours depending on their configuration. The problem with these hyper-parameters is that there isn't a rule that relates them to the problem, so their choice should follow research. The latter aims to find out the configuration that better suits the problem, solving it in the best way possible. However, in DL we should consider the computational complexity since some of these parameters govern the complexity of the model. In this way, the research is focused not only on the better model but on a trade-off between better and simpler models, although the problem resolution has more weight.
The hyper-parameters research can be done in different ways:

- **Experience choice**: the values are chosen by experience matured with the model or with the work on similar problems;
- **Grid search**: the tested values come from the combination of the equidistant parameter lists that allows exploring a portion of the parametric space, with the computational burden depending on the density and size of the explored space;
- **Random search**: the values come from sampling predetermined ranges of parameters to explore the parametric space, usually in a uniform way, with the drawback of not being able to explore optimal regions of the space;
- **Improved Random Search**: The limitations of random search seek to be mitigated with techniques that guide the exploration of the parametric space by exploiting the results of previously tested instances, for example using the Bayesian rule.

In this work, the second method is mainly used, with the choice of intervals aided by the first method. These will be shown in the following sections relating to specific models.

3.2.6 TCN predictor model

The TCN predictor model has two different configurations, the first hyper-parameter in consideration. The first one uses the last temporal features extracted by the TCN layers to make the regression prediction. So, this is composed of a series of TCN layers where the last one returns only the features of the last timestamp that is fed directly into the output layer. In this way the architecture results simpler but has the receptive field limited in the last part of the sequence; if the network is not so deep the features used for the prediction aren't originated from the whole sequence, which should limit the capabilities of the configuration. Instead, the second configuration uses a series of TCN layers to extract a time sequence of features that is suddenly reduced with a 1DCNN layer which has kernel size and stride of 5, reducing by 5 the time dimension. The reduction using 1DCNN is done to avoid the huge amount of parameters that will be required using a fully connected layer on the whole features. The reduced time sequence is flattened and sent to a fully connected layer with 16 units before arriving at the output layer. In this way the features processed are taken from the whole sequence but the model is more complex. In both configurations, the activation function of the output layer is a linear activation instead of a Sigmoid one. This is done because the vanishing gradient problem arises with the use of the Sigmoid function.
The other hyper-parameters are inspired from the example [32], which are all those possible regarding the TCN model if we exclude regularization techniques. The activation function for both TCN layers and fully connected one is added as it is a parameter not present in the reference example. The optimization through a grid search is done in two steps. The first step parameters are:

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

The choice of grid parameters is adopted firstly to explore a portion of the parametric space and then to use the results to explore further if these prompt to do so. This permits avoidance testing of a series of configurations that prove bad behaviour from the beginning. The second step parameters are:

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

In the second step, the more complex configurations were discarded, and we explored the kernel size with the dilated convolution depth. Finally, the parameter of the number of filters was finished.
Given the satisfactory results, it was not necessary to further expand the research on the parameters. This approach is a mixture of grid search and experience choice, the latter mainly guided by literature examples and the results achieved.

3.2.7 LSTM predictor model

As done for the TCN predictor model, the first hyper-parameter for the LSTM predictor model consists of different configurations, in this case, three. The first one uses the last temporal features extracted by the LSTM layers to make the regression prediction. As for the TCN predictor, the last one returns only the features of the last timestamp that is fed directly into the output layer. In this case, there isn't the problem of the receptive field; for how LSTM architecture is made, the last features originated from the information of the whole sequence. However, for long time sequences as in this case, there could be problems related to learning long term dependencies. It means that such ones could be obfuscated by short term ones, possibly resulting in more or less the same receptive field problem for different reasons. The second one is also in the same spirit as the TCN predictor; it uses stacked LSTM layers to extract a time sequence of features that is then reduced with a 1DCNN layer by a factor of 5 as for TCN. The reasons that motivate this are the same. The reduced time sequence is delivered to the same fully connected layer with 16 units that in this case have Leaky ReLU as fixed activation function, as for the 1DCNN layer. The third configuration adds after the results obtained in the first two, further exploring the architecture types.

While the second configuration is motivated by examples like in [7] showed in 3.4, the third one reverses the LSTM and 1DCNN elements, like in [33]. In this way, instead of reduc-

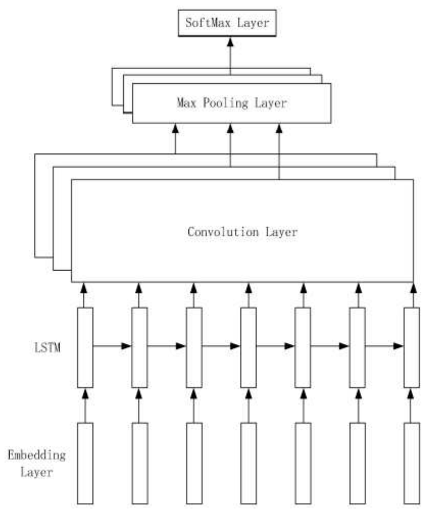


Figure 3.4: LSTM-CNN architecture example from [7]

ing the size of the LSTM features, the input is reduced in size before extracting the features due to the LSTM layers. This leads to the simplification and lightening of the model, as the 1DCNN levels are less demanding from a computational point of view and putting them at the beginning allows the LSTM elements to process lower dimensional data. As for TCN, in all configurations, the activation function of the output layer is a linear activation instead of a Sigmoid one, for the same reason.
The other hyper-parameters are taken from [34]. These parameters are:

1 Number of LSTM layers: from 1 to 3;

2 Number of units per layer: 25, 50 or 100;

3 Dropout rate: 0, 0.1 or 0.25.

Dropout [35] is a regularization technique that consists of deactivating with a certain probability (the dropout rate) the units of the layers during training. This leads the units to learn the general characteristics and actively participate in the analysis of the different characteristics. This is because it is possible that some units are not involved in some training steps and it is not possible to rely on them, preventing an excessive specialization of the units.
It was not necessary to further expand the research on the parameters except for the improvement done with the third configuration; this was tested with the same other parameters.

3.3 Results obtained on the Virtual Metrology regression task

All the results that came out from the regression task described in the previous section will be shown in this one. The study is carried out using the Python programming language, the most used concerning ML and DL problems, with the support of the Keras library [36] for the tools related to the construction and training of DL models and the scikit-learn library [37] for evaluation metrics. The analysis is performed using only the first division of the dataset and the statements are strengthened using CV at the end. This allows using fewer computational resources during the study with the drawback of depending on the particular data distribution, which could produce less significant results. However, this drawback is checked and mitigated using CV once selected the final configuration.

3.3.1 Benchmark predictor

We start showing and analysing the behaviour of the benchmark model. The model, as previously mentioned, has a low number of parameters due to its simplicity and lightness; this is usually preferred since a simpler model allows not to waste computational resources and to be exploited more, given the presence of fewer parameters that have to learn the task. The latter is learned from the model, as the performance metrics in 3.1 show; the MAE and ME are low for the validation source dataset(B) and the test dataset. In addition, the training history in 3.5 shows the decrease in the training and validation losses, meaning that the model is doing what is expected.

Evaluation of the benchmark predictor				
Dataset	MAE	ME	EV	R2
Source (B) validation	0.0667	0.310	-	-
Source (B) test	0.0705	0.256	0.822	0.816
Target (A) test	0.227	0.484	0.738	-0.573

Table 3.1: Benchmark predictor: performance evaluation

However, some defects are present; the simplicity of the model has its drawback since it makes the training more unstable, causing spikes in the validation loss given by the impact on the change of the parameters which, being few, have a greater influence on the modification of the forecast. This could cause the early stop criterion to intervene too soon during the training procedure, worsening the final performance. In the same way, these few amounts

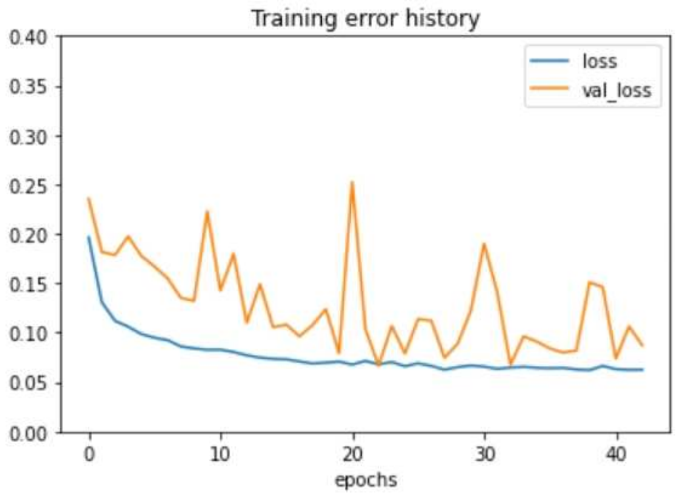


Figure 3.5: Benchmark predictor: training history

of parameters influence the predictions which appear to be strongly related to the dataset presented during training.

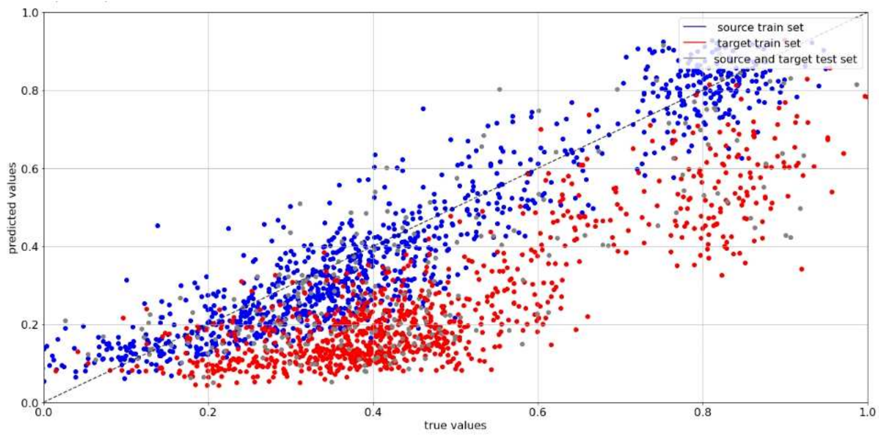


Figure 3.6: Benchmark predictor: predictions plot

As we can see in 3.6, the predictions are good only for the source domain(blue dots) while the ones regarding the target domain(red dots) are really bad. The test samples(grey dots) follow the distribution of their respective provenance, behaving in the same way as the training set. Perfect predictions belong to the centre dotted line, the place where true values (on the x-axis) match their own predicted values (on the y-axis). The predictor was not expected to do well in the target domain but it's clear that the model is not capable at all of generalizing into similar data, demonstrating how much the simplicity of the model makes it dependent on the specific task learned. This also can be seen in the performance metrics results, where the numbers are good for the B dataset while they are bad for the A-one; this is particularly evident in the R2 score. These results invite the exploration of further models to refine these aspects, which may also be due to the model's incompatibility in easy managing temporal sequences.

3.3.2 TCN predictor

To improve the shortcomings of the benchmark, the first model chosen is TCN based.

Hyper-parameters optimization and selection

As previously described, a hyper-parameter procedure is made for finding the best parameters of the explored parametric space. The performances on the validation data are used to compare them, averaging the results on the best performing configurations given the parameter in the examination; the MAE and ME of the 6 best models are averaged to provide the metrics for the comparison. The number of models considered is chosen to also include some of them that do not work very well, avoiding focusing only on good models to perform a richer evaluation.
The first step of this optimization gives the results in 3.2. These motivate the need for further exploration of the parameters and the justification of the non-unique optimization attempt. First of all, the more complex architectures prove to work bad, meaning that this aspect will still lead to wrong behaviour, regardless of the other parameters. This will cause a waste of resources in case of the further exploration of the configurations with more than 1 layer and more than 1 block, highlighting the usefulness of more than one optimization attempt. Then comparing the model type and the activation function gives a clear result regarding the best solution, instead of the number of filters parameter that tends to improve as the number increases. Therefore, the second optimization aims to dispel these remarks, exploring further the number of filters and testing simpler configurations through the dilated convolution depth parameter, with the addition of kernel size as a new variable.

First step of TCN predictor hyper-parameter optimization		
Hyper-parameter	MAE	ME
Model type		
TCN only	0.0859	0.298
TCN + 1dcnn	**0.0734**	**0.260**
Activation function		
ReLU	0.0837	0.302
Leaky ReLU	**0.0747**	**0.253**
Number of TCN layers		
1	**0.0694**	**0.269**
2	0.127	0.477
Number of dilated convolution blocks		
1	**0.0694**	**0.269**
2	0.112	0.424
4	1.92	6.84
Number of filters		
16	0.103	0.348
32	0.0893	**0.293**
64	**0.0790**	0.311

Table 3.2: TCN predictor: hyper-parameter optimization, first step

Given the results in 3.2 and 3.3, the chosen configuration can be described, with the motivations behind the selection, not dependent only on numerical results in some cases. The architecture with the combination of TCN for extracting temporal features and 1DCNN to combine and reduce them is chosen as it performs the best. For the same reason, the Leaky ReLU activation function is selected among ReLU. The same happens for the number of layers and blocks of 1 each. Leaky ReLU probably permits to mitigate the presence of dead units(units that are not learning), a limitation occurring with using ReLU activation when the values of the neurons are negative, justifying its better results. The number of filters is set to 64, as the path of the improving performance with the increasing numbers stops with the second optimization attempt since the parameter of 128 didn't do better. Regarding the kernel size, even if the results are evident, there is a drawback; the complexity of the model increases enormously with a larger size than an analysis of the trade-off between performance and complexity is due. In this case, the task results guided the choice since this aspect is more important than the others and the complexity is manageable in the specific task, not requiring an exaggerated time for training. So, 50 is chosen for the kernel size even if it leads to 5 times more the training time compared to the model with the kernel size of 3 and an amount of 1.5 million parameters. Instead, the dilated convolution depth didn't give useful information, so the 6 parameter is chosen as the receptive field is wider than the 4 parameter, which is preferable for extracting more general features.

Second step of TCN predictor hyper-parameter optimization		
Hyper-parameter	MAE	ME
Model type		
TCN only	0.0752	0.252
TCN + 1DCNN	**0.0534**	**0.216**
Number of filters		
64	**0.0586**	**0.207**
128	0.0589	0.223
Kernel size		
3	0.0755	0.259
15	0.0685	0.247
50	**0.0555**	**0.215**
Dilated convolutional depth		
4	**0.0575**	**0.214**
6	0.0600	0.215

Table 3.3: TCN predictor: hyper-parameter optimization, second step

Summarizing, the TCN architecture chosen for the regression task as the following characteristics:

- **Model type**: TCN + 1DCNN;
- **Activation function**: Leaky ReLU;
- **Number of TCN layers**: 1;
- **Number of dilated convolution blocks**: 1;
- **Number of filters**: 64;

- **Kernel size**: 50;
- **Dilated convolution depth**: 6.

Regression task results

The improvements with the use of TCN architecture are visible. The task is learned from the model in a better way compared to the benchmark results in 3.1, as it can be seen in 3.4; the MAE and, in particular, ME are lower for the validation and the test source dataset. In addition, the training history in 3.7 is more stable, even if it starts from higher values given by the greater complexity. This permits the training to last longer than the benchmark, improving the overall behaviour.

Evaluation of TCN predictor				
Dataset	MAE	ME	EV	R2
Source (B) validation	0.0531	0.185	-	-
Source (B) test	0.0560	0.164	0.895	0.893
Target (A) test	0.129	0.357	0.720	0.377

Table 3.4: TCN predictor: performance evaluation

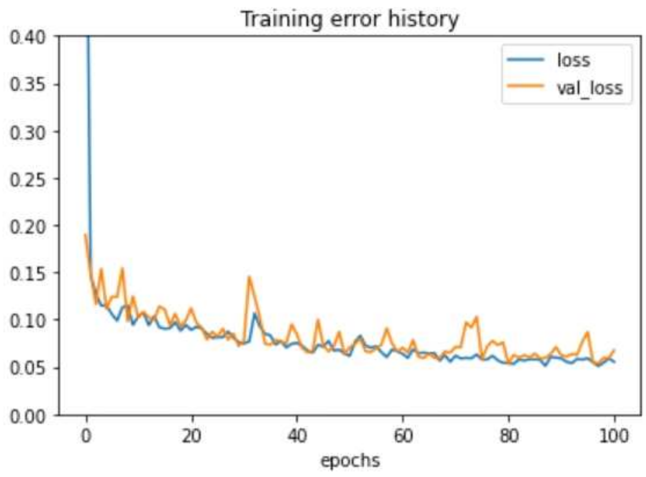


Figure 3.7: TCN predictor: training history

Therefore, the major complexity solves some issues encountered in the benchmark model; along with the more stable training and the better results on the learned regression task, the

TCN architecture proves to extract more general properties. Indeed, the results are decent also for the target domain, demonstrating a better generalization capability. This can be seen from all the metrics and also from the predictions in 3.8. In this case, the target domain pre-

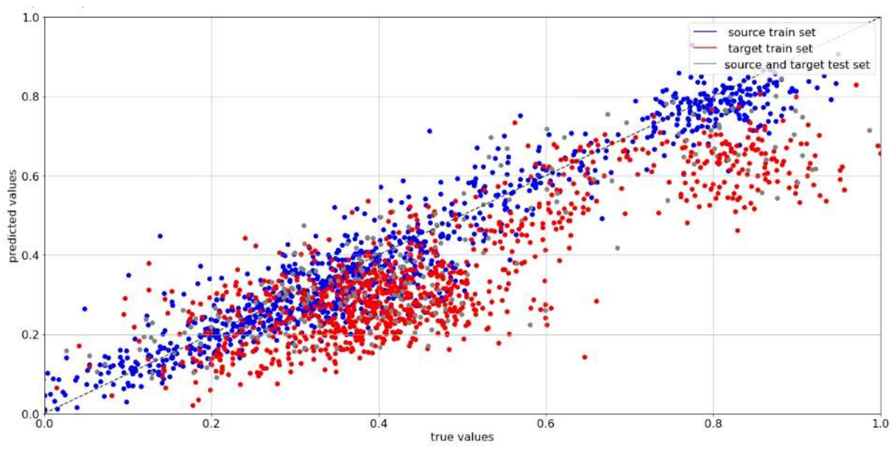


Figure 3.8: TCN predictor: predictions plot

dictions(red dots) are closer to the optimal prediction line compared to the benchmark, further showing the claim regarding the best generalization properties. The source domain predictions(blue dots) do the same showing the better performances of the architecture. This is followed by the test sets(grey dots) that behave the same as the training ones.
The huge complexity of the model seems the only drawback of the TCN architecture that proves to generalize better than the benchmark, along with the better results. However, this model doesn't solve the regression task for the target domain even with its generalization properties, an aspect that will be treated later on and requires other methods.

3.3.3 LSTM predictor

The other architecture chosen is based on LSTM, providing an alternative with the same purposes as TCN.

Hyper-parameters optimization and selection

As previously described, a hyper-parameter procedure is made for finding the best parameters of the explored parametric space. The performances on the validation data are used to compare the parameters, averaging MAE and ME of the 6 best models in the same behaviour of the TCN architecture. This optimization gives the results in 3.5. These, differently from

TCN, show a stability property in the LSTM architecture given that most parameters behave very similarly; the results are almost the same except for the model type. This justifies the stop in the hyper-parameters search procedure since there is no clear direction for testing new parameters, with no need to improve performance. Hence, the LSTM architecture chosen for the regression task has the following characteristics:

- **Model type**: 1DCNN + LSTM;
- **Number of LSTM layers**: 1;
- **Number of units**: 50;
- **Dropout rate**: no dropout.

LSTM predictor hyper-parameter optimization		
Hyper-parameter	MAE	ME
Model type		
LSTM only	0.0753	0.300
LSTM + 1DCNN	0.0543	0.207
1DCNN + LSTM	**0.0470**	**0.187**
Number of LSTM layers		
1	0.0489	**0.169**
2	0.0481	0.184
3	**0.0476**	0.200
Number of hidden units		
25	0.0511	0.188
50	0.0476	0.193
100	**0.0474**	**0.181**
Dropout rate		
No dropout	**0.0469**	0.187
0.1	0.0482	0.189
0.25	0.0486	**0.180**

Table 3.5: LSTM predictor: hyper-parameter optimization

The 1DCNN followed by LSTM configuration is chosen as it performs the best, the only clear choice among the parameters. The number of LSTM layers is chosen for the principle of selecting the simplest model if they perform similarly, as for the picking of the absence of dropout. This, however, is not applied for the number of hidden units, where 50 is chosen to allow the model to have greater capabilities in extracting the features necessary to solve the task. There is no problem having a more complex model since it remains simpler than TCN, with 200k parameters instead of 1.5 million. However, there is no need to complicate it too much for no reason; this will happen using 100 for the number of hidden units without proof of a performance improvement. Choosing 50 instead of 25 is intended to give the model more room to manoeuvre on the problem.

Regression task results

Like the TCN model, the task is learned better than the benchmark results in 3.1, as it can be seen in 3.6; the MAE and ME are lower for the validation and the test source dataset. The stability improvements also appear there, with the more stable training history in 3.9, which permits the training behaviour to improve.

Evaluation of LSTM predictor				
Dataset	MAE	ME	EV	R2
Source (B) validation	0.0535	0.177	-	-
Source (B) test	0.0624	0.165	0.900	0.892
Target (A) test	0.141	0.338	0.689	0.302

Table 3.6: LSTM predictor: performance evaluation

The extraction of more general properties takes place there too, with the LSTM architecture proving to demonstrate better generalization capability than the benchmark. This can be seen from all the metrics and the predictions in 3.10, presenting decent results also in the target domain. The better performances of the architecture compared to the benchmark are shown with the source domain predictions(blue dots) that are closer to the optimal prediction line and with the target domain predictions(red dots) doing the same. The test sets(grey dots) behave as the training ones.
Overall, the LSTM model appears to be more balanced than the TCN model, with architecture more stable concerning parameter modification and the lower complexity, however, showing similar performances.

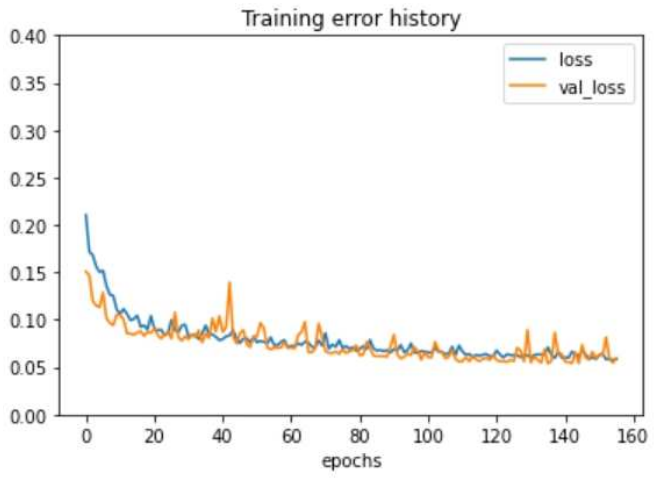


Figure 3.9: LSTM predictor: training history

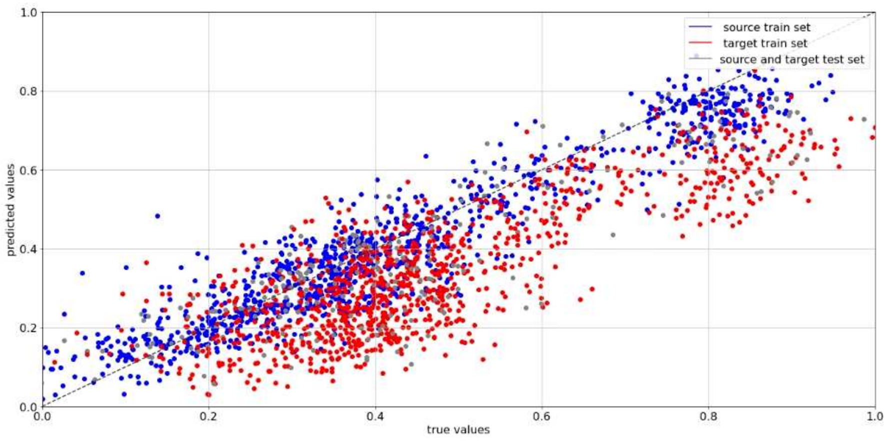


Figure 3.10: LSTM predictor: predictions plot

3.3.4 Cross-validation results

The cross-validation results are shown in this subsection, with some final consideration about the resolution of the regression task.
The CV, as described previously, is performed to confirm the statements regarding the single results, asserting that there isn't a particular behaviour given by the specificity of the distribution of the samples used. For this reason, all the selected models described above are trained on 5 different arrangements of the data to obtain the mean results (with the standard deviation indicated by the $\pm$ symbol) that are invariant to the different folds of the dataset. The results are shown in 3.7; they reflect the previous ones, which means that there isn't an

upheaval with the analysis on different disposals of the dataset and the models, therefore, behave the same on all the 5 folds. So the previous claims are confirmed, with the TCN and LSTM predictors solving the regression task better than the benchmark. Finally, we can also compare the TCN model and the LSTM one in this task; the greater complexity of the TCN model allows it to perform the best in the regression problem, which is more evident by observing the target domain A where the results are much better. This proves the better generalization properties of the TCN model compared to the LSTM one, sacrificing, however, the simplicity and lightness aspect of the model.

Cross-validation results of the regression task					
Dataset	Predictor	MAE	ME	EV	R2
Source (B) validation	Benchmark	0.0711 ± 0.006	0.286 ± 0.07	-	-
	TCN	**0.0557** ± 0.004	**0.251** ± 0.06	-	-
	LSTM	0.0589 ± 0.004	0.279 ± 0.08	-	-
Source (B) test	Benchmark	0.0721 ± 0.004	0.249 ± 0.02	0.845 ± 0.01	0.837 ± 0.02
	TCN	**0.0549** ± 0.003	0.214 ± 0.05	**0.904** ± 0.01	**0.902** ± 0.02
	LSTM	0.0584 ± 0.005	**0.191** ± 0.02	0.896 ± 0.02	0.892 ± 0.02
Target (A) test	Benchmark	0.223 ± 0.1	0.509 ± 0.1	0.684 ± 0.1	-0.712 ± 1
	TCN	**0.130** ± 0.02	**0.397** ± 0.07	**0.735** ± 0.02	**0.345** ± 0.1
	LSTM	0.174 ± 0.03	0.426 ± 0.02	0.682 ± 0.04	-0.113 ± 0.2

Table 3.7: Cross-validation results of the regression task

4 Domain Adaptation method with Deep Learning models

This chapter talks about the Domain Adaptation method used for solving the problem of making the solution for the regression task more general and scalable. We introduce the configuration of this method, the DBAM approach, with its components and the adversarial training principle used, taken from the Wasserstein GAN. Finally, we present the results obtained in this task.

4.1 DBAM architecture

DANN-Based Alignment Model (DBAM) was introduced in the reference work [1, 2] and bases its development, as the name says, on Domain Adversarial Neural Network (DANN). Neural Networks have become a standard solution for performing Domain Adaptation in recent years, where the first use of DANN was presented in [38]. In [38], the adaptation is based on a deep feature extractor meaning that the Domain Adaptation is performed on the features space to produce similar features for both the domains. Instead, in [2] the alignment is performed using an aligner model with an autoencoder structure. This allows for interpretability of the aligned data, with the possibility of using different features size between source and target domains. In this case, the Domain Adaptation is performed directly in the data space, being able to compare the samples of the two domains instead of relying

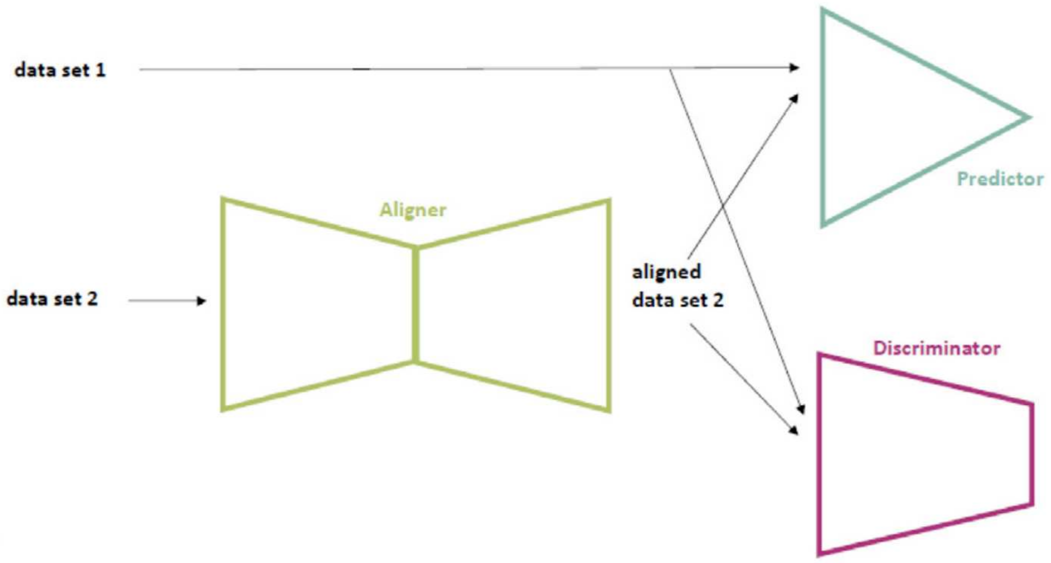


Figure 4.1: Graphical representation of the DBAM architecture from [1]

only on features.
The three main components of the DBAM architecture, which base on the GAN model, are:

1. A prediction model P (also called predictor), which performs the main task, a regression one in this case;
2. An aligner model A (also called aligner), used to perform the Domain Adaptation part adapting data from a target domain to a source domain;
3. A discriminator model D (also called discriminator), which is necessary to distinguish between domains in the adversarial training procedure.

Regarding the single components architecture, one of the predictors is known already since it was presented previously for the various solutions, with the others that will be described in detail later. To briefly know what to expect, the discriminator model has the same structure as the predictor as it performs a similar task with different behaviour since it needs to analyze the data and produce a single number as the predictor does. Instead, two predictors' components form the aligner model, one used in the same way as the predictor and one mirrored. The first is used for extracting features using multiple outputs instead of only one for the predictor model's case. The other is used to reconstruct the data from the extracted features, making them, in this case, look more like target domain data.

4.1.1 Autoencoder architecture

To better understand the behaviour and structure of the aligner architecture, we briefly describe the autoencoder model. This one is a Neural Network architecture that aims to extract a representation of the data taken into consideration, usually in a lower-dimensional form. This is pursued with two components, the encoder and the decoder; the first is a NN architecture that has the role of compressing the data in a lower-dimensional representation, named code or bottleneck. The second does the opposite thing, reconstructing the samples from the code. For this reason, this part usually is made by mirroring the encoder, but it is not required. This kind of model is included in the Unsupervised Learning category since

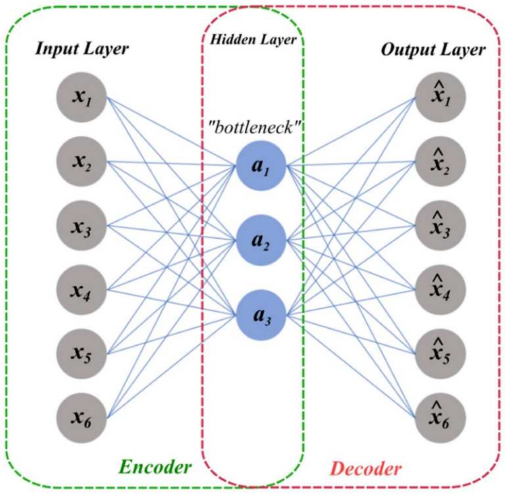


Figure 4.2: Autoencoder architecture example from [8]

the output, in this case, is the input itself, so there isn't a need for labels. This training aims to learn to extract data features that permit data reconstruction, information likely to represent all the useful information in a compressed form.

4.1.2 GAN model

Generative Adversarial Network (GAN) [39] is a particular architecture which introduced into Deep Learning the concept of Adversarial Training to learn and perform a particular task, usually related to the generation of new data. The generation and manipulations of images are one field where this kind of approach is used a lot like this example [40]. This architecture employs two different pieces called generator and discriminator. The generator has the role of producing believable artificial data, while the discriminator has the role of distinguishing between real data and generated data. Both the generator and the discriminator are ANN with arbitrary architecture. As usually happens in DL, the entire system is trained by using backpropagation [41], with the weights of the generator that are updated to maximize the training loss. This is correlated to the inability of the discriminator to perform its job. In contrast, the discriminator weights are updated to minimize such loss. The maximization of this metric means that the generated data is brought to be better as it can deceive the discriminator more. Instead, minimization means that the discriminator is better at distinguishing between real and generated data. In this manner, the learning dynamic is a min-max two-players game based on Game Theory, with a value function that one agent (the discriminator D) tries to minimize while the other agent (the generator G) tries to maximize:

$$min_G max_D V(D,G) = \mathbb{E}_{x \sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))] \quad (4.1)$$

where $D(x)$ is the probability that x originated from the true data distribution. Discriminator parameters are updated by ascending its stochastic gradient:

$$\nabla\theta_d = \frac{1}{m}\sum_{i=1}^{m}\left[logD(x^{(i)} + log(1 - D(G(z^{(i)})))\right] \quad (4.2)$$

while generator parameters are updated by descending its stochastic gradient:

$$\nabla\theta_g = \frac{1}{m}\sum_{i=1}^{m} log(1 - D(G(z^{(i)}))) \quad (4.3)$$

where $z^{(i)}$ is a sample in a minibatch of size m. The game, which follows the claims of Game Theory, ends in an equilibrium which is a maximum for one player's strategy and a minimum for the other player's strategy. If this happens, the training of a GAN is performed successfully as the discriminator cannot discern between real and generated samples once the equilibrium state is reached, correctly guessing about 50% of the time, meaning a random choice.

4.1.3 Wasserstein Generative Adversarial Networks

It is common in GANs to face issues such as training instability and mode collapse because the optimization is a competition between two parts that, if one prevails, leads to the training purpose failing as it can't reach the equilibrium. A variation of GAN called Wasserstein GAN (WGAN)[42] was proposed to solve those issues by replacing the discriminator model with a critic, which acts as a helper to estimate an approximation of the Wasserstein distance between source and target distributions. The WGAN discriminator is named critic since it doesn't discriminate between real and artificial samples but instead outputs a score regarding the realness of the samples. In this manner, it evaluates how much the data seems to come from each of the two data distributions. Indeed, one of the major differences between GAN and WGAN is removing the sigmoid function in the last layer of the discriminator since the output could be an arbitrary number and not a value between 0 and 1 representing a probability. The critic, therefore, seeks to maximize the distance between its output on real samples and its output on artificial samples. At the same time, the generator tries to maximize the discriminator output for target samples. Formally, the loss function for the critic can be written as follows:

$$\nabla w = \frac{1}{m} \sum_{i=1}^{m} \left[f(x^{(i)}) - f(G(z^{(i)})) \right] \tag{4.4}$$

while the loss for the generator is:

$$\nabla \theta = \frac{1}{m} \sum_{i=1}^{m} f((G(z^{(i)}))). \tag{4.5}$$

To enforce a Lipschitz constraint on f, weight clipping is applied to restrict the maximum weight value of f. Lipschitz constraint limits the gradient of the discriminator making it worse but providing more gradient information which helps train the generator; this improves the performance of the GAN in general. However, the weight clipping utilization can result in poorly generated outputs and failure to converge. [43] propose an alternative which adds a penalization term to the norm of the critic gradient compared to its input. Formally, the gradient penalty is obtained as in equation 4.6:

$$gp = \lambda(||\nabla \hat{x} f(\hat{x})||_2 - 1)^2 \tag{4.6}$$

where $\hat{x}$ is uniformly sampled from the real data distribution and the artificial data distribution, and λ is a weight parameter for the gradient penalty. The gradient penalty term is then added to the critic loss in 4.4. Another aspect to note is that the critic must be updated more times with respect to the generator (e.g., 10 times more as suggested in [42]) to avoid the generator prevails on it and ensure critic convergence.

Since WGANs seek to minimize the distance between distributions, the idea of using them in the Domain Adaptation field is immediate.

4.2 Adversarial Domain Adaptation task

In Adversarial Domain Adaptation, the network's objective is not to generate new samples but to standardize samples from the target domain to appear more similar to the source domain elements. The discriminator will try to separate source and target samples. The generator will try to fool the discriminator by adapting target samples. Ideally, this will train a generator model capable of shifting samples from target to source domain. The latter is obtained with an overall training procedure similar to what was previously seen for Wasserstein GANs [42] with a loss coefficient added regarding the supervised setting part. As for Wasserstein GANs, a balance between discriminator and aligner is obtained through iterative training. The discriminator is first trained a certain number of additional times, with the aligner's weights fixed. Then, the aligner is trained while the discriminator's weights are fixed. Moreover, the aligner undergoes a warm-up training procedure which is suggested to ensure a good initialization. Since the predictor is assumed to be a trained model, its weights are kept frozen during the whole DBAM training.
This procedure is performed as in the reference work [1], from which the benchmark model is taken as the comparison starting point with some procedure characteristics. In this work, the configurations tested progressively change from the benchmark elements to the TCN and LSTM ones. It starts by changing the predictor model, keeping the WGAN elements as the benchmark. Continue introducing the new aligner models and completing the modification with the discriminator part. It allows us to see progressively the improvements or the problems encountered in each element. Before that, a parenthesis opens to show the limitations and problems of using the predictive model alone to perform the Domain Adaptation part, justifying the use of an ad-hoc procedure.

4.2.1 Training procedure parameters maintained

The training is performed with some parameters that don't change compared to what is described in [1]. It happens for the same motivation of the regression task to preserve the possibility of comparing the new solutions with the benchmark. These are:

- **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;
- **Critic gradient penalty factor**[43]: fixed to 10.

Furthermore, the warm-up training of the aligner was kept the same, as it doesn't add valuable comparison information but is useful only to see if the model is working. It has 100 epochs with Adam optimizer, the learning rate of 0.0001 and the learning rate schedule and early stopping criterion as the training of the regression task.

4.2.2 Training procedure hyper-parameters

In this case, some aspects of the training procedure have to be tuned, since the training configuration is much more complex than the one of the regression task. The models that should be trained are two and also depend on the already trained predictor, therefore needing some tweaks to make the training effective. However, it is counterproductive to change too many parameters since the testing requires considerable time. For this reason, they are limited to two:

1 Adversarial loss weight (alpha): chosen from the set {0.1, 0.25, 0.5, 1, 5};

2 Discriminator training extra steps: chosen from the set {5, 10, 20}.

The first one refers to the balancing between the use of the unsupervised loss, the loss of the adversarial setup, and the supervised one, the predictor's loss; the overall loss is equal to the sum of the two, with one of the two components multiplied by alpha or it's inverse depending on the value of alpha. If the latter is lower than 1 means that the training is focused more on finding an alignment that preserves the performance of the prediction, so the predictor's loss is multiplied by the inverse of alpha to enhance it. With alpha bigger than 1, the model focuses more on the quality of the domains' alignment, so the adversarial loss is multiplied by alpha to enhance it. This parameter is always tested, choosing the configuration that balances the training depending on the predictor characteristics. The benchmark uses 5 for this parameter.
The second one is changed only if needed after the alpha results, otherwise kept at the original

value of 20. This parameter balances the training ratio between the aligner and the discriminator, an aspect needed to keep the competition alive between the two parts, avoiding that one suppresses the ability of the other. The decreasing of this value is needed in the presence of a complex discriminator model that is more capable of its task and so needs less training to compete, being able to save time too. The latter could also be the reason for the decrease in this parameter if a compromise on training time is necessary.

4.2.3 Benchmark DBAM elements

Aligner

The 1DCNN aligner model taken as reference has the following elements:

1. 1DCNN layer with 32 filters, 7 as kernel size and stride of 2;
2. Average pooling layer with 2 as pooling dimension;
3. 1DCNN layer with 16 filters, 7 as kernel size and stride of 2;
4. Upsampling layer with 3 as upsampling parameter;
5. 1DCNN layer with 32 filters, 21 as kernel size and stride of 1;
6. Upsampling layer with 3 as upsampling parameter;
7. 1DCNN layer with 33 filters, 33 as kernel size and stride of 1.

The 1DCNN layers have ReLU as an activation function, use the valid padding parameter in the convolution(which is not causal) and have also a dropout rate of 0.15. The components from 1 to 3 compose the encoder, while the others form the decoder. The time dimension of the input is halved 3 times, so the first dimension of the code is $1024/8 = 128$ which becomes 124 for the convolution operation in these steps. Instead, the feature dimension is 16, the number of filters of the last 1DCNN layer of the encoder. The last 1DCNN layer uses the number of filters equal to the dimension of the input features, to reconstruct its exact dimension, and uses a linear activation function without dropout. This configuration, as the predictor counterpart, results in simple and lightweight with about $57k$ parameters.

Discriminator

The 1DCNN discriminator model taken as reference has the following elements:

1. 1DCNN layer with 32 filters, 17 as kernel size and stride of 1;
2. Maximum pooling layer with 4 as pooling dimension;
3. 1DCNN layer with 16 filters, 17 as kernel size and stride of 1;
4. Maximum pooling layer with 4 as pooling dimension;
5. Fully connected layer with 512 units applied to the flatten version of the features extracted previously;
6. Fully connected layer with 256 units;
7. Fully connected layer with 128 units;
8. Fully connected layer with 64 units;
9. Fully connected layer with 32 units;
10. Output layer(1 fully connected unit) that use linear activation function.

1DCNN and the fully connected layers use Leaky ReLU as an activation function. It should be noted that in this configuration, differently from the predictor's one, the output is linear as the critic requires to express a number indicating the quality of the sample analyzed. The discriminator owns about $726k$ parameters, resulting in a more complex model because of the fully connected part.

4.2.4 TCN DBAM elements

Aligner

The TCN autoencoder structure for the aligner takes most of the characteristics of the configuration in [44]. Then, 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 the valid padding parameter. As described in [44], the TCN layers for the reconstruction possess a dilated convolution depth that is inverted, in a way that the end output can exploit more of the compressed features, using all of them after the upsampling. Instead,

two parts that are not adopted are the use of the convolution layer at the end of each TCN layer and the combination of the subsequent outputs evolving after each TCN layer. The first is avoided since the intention is to study only the TCN elements while the second requires lots of layers to be useful, which are not present. A common characteristic of the TCN layers, in this case, is the use of ReLU instead of Leaky ReLU, used in the predictor counterpart. In this particular task, it doesn't make sense to have negative numbers since all the signals are positive and the reconstruction is applied to them. Another characteristic of these layers is that they use only one dilated convolution block.
Hyperparameter optimization, in this case also, is performed in the warm-up training part, used mainly to see if different configurations are working. The performances, in this case, aren't very important for the DBAM training results, but of course, the aligner needs to be a working architecture. The combination of parameters tested for this architecture are:

1. Number of TCN layers: 1 or 2;
2. Number of filters: 8, 16 or 32;
3. Kernel size: 15 or 50;
4. Type of compression: Average pooling or 1DCNN pooling.

The last parameter is chosen to see if a learned pooling performs better than a fixed one, with both of them with 16 as the compression factor. This model is the starting architecture, which will be modified if necessary based on the results achieved.

Discriminator

The architecture of the TCN discriminator was not developed, as the aligner results did not make it necessary, as will be seen later. However, a general idea for imagining the architecture could be the union of the feature extraction part of the TCN predictor with the feature processing part of the benchmark discriminator. The LSTM discriminator uses this idea later.

4.2.5 LSTM DBAM elements

Aligner

The LSTM autoencoder structure for the aligner takes inspiration from [8, 45]; a series of 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.

Differently from [8, 45], the whole sequence of features is retained and compressed instead of producing a single vector(the last output of the LSTM elaboration); this is because the degree of compression would be too stringent and would deteriorate the performance, as previously seen with the predictor's counterpart. Anyway, it is done the same way for the TCN aligner, which shares the behaviour of not using other types of layers. It is done for the same reason, testing only the specific type of layers without using 1DCNN ones. Instead, one different aspect is in the decompression part: as showed in [45], the feature code is not upsampled but repeated N times to restore the original dimension. This choice is made since it suits more the elaboration of the LSTM layer, which can have available all the sequence of features in the progression of the elaboration, instead of relying on a series of repeated static features, as happens with the upsampling.
Also in this case an hyperparameter optimization is performed, for the same reason of the TCN aligner. The combination of parameters tested for this architecture are:

1 Number of LSTM layers: 1 or 2;

2 Number of units: 25 or 50;

3 Dropout rate: 0, 0.1 or 0.25;

4 Type of compression: Average pooling or 1DCNN pooling.

As for the TCN aligner, the last parameter owns 16 as the compression factor. This model is the starting architecture, which will be modified if necessary based on the results achieved.

Discriminator

Combining the LSTM predictor and the benchmark discriminator inspires the LSTM discriminator architecture. For this reason, the model is built using the first part of the LSTM predictor for the features extraction part and the fully connected part of the benchmark one for the features elaboration. However, we implement two other changes. The first is to add a 1DCNN layer to reduce further the sequence size, subsequently processed by the LSTM layer. The second modification is the fewer units present in the LSTM layer. These two are both forced by the higher computational complexity of the LSTM layer, which enormously increases the required time in the DBAM training. Without these and then with the same feature extraction part of the predictor model, each epoch requires more than 10 minutes to be addressed, which is almost $3-4$ times the configuration with the two changes and 10

times the benchmark DBAM one. The reduction of the LSTM units also makes the feature dimension similar to the benchmark one, so it's not increasing the parameters regarding the fully connected part. In summarizing, the architecture is composed by:

1. 1DCNN layer with 50 filters, 5 as kernel size and stride of 5(divides by 5 the dimension);
2. 1DCNN layer with 50 filters, 3 as kernel size and stride of 3;
3. LSTM layer with 16 units and without dropout;
4. Fully connected layer with 512 units applied to the flatten version of the features extracted previously;
5. Fully connected layer with 256 units;
6. Fully connected layer with 128 units;
7. Fully connected layer with 64 units;
8. Fully connected layer with 32 units;
9. Output layer(1 fully connected unit) that use linear activation function.

1DCNN layers and also the fully connected ones use Leaky ReLU as activation function, like the models they are inspired by. The discriminator has nearly the same amount of parameters as the benchmark, but with the LSTM layer being more computationally demanding.

4.3 Results obtained on the Domain Adaptation task

This section will show all the results achieved in the Domain Adaptation task described in the previous ones. As for the regression task, the study is carried out using the Python programming language supported by the Keras library [36] and the scikit-learn library [37]. The analysis is performed using only the first division of the dataset. In the end, the most important statements are strengthened using cross-validation once selected the final configuration.

4.3.1 The need of a Domain Adaptation ad-hoc procedure

First of all, we want to motivate how it is necessary to adopt an ad-hoc solution instead of solving the problem using the predictor only. For this reason, the TCN and LSTM predictor models found in the previous chapter encounter another two training procedures, the

two most immediate to try to adapt the resolution of the problem to both domains. The first one is to train the models using the overall dataset, presenting to them both the source domain(B) and the target domain(A). The other one is to extend the training of the models with the target domain. In the latter case, the models are already capable of solving the regression task on the source domain, so the training adds after it. Instead, in the first case, the models are freshly initialized.
The first training procedure is performed in the same way as the one already described, present in the previous chapter. The only difference is that all the dataset parts used in the training procedure are doubled in dimension since they include the data from the A domain.

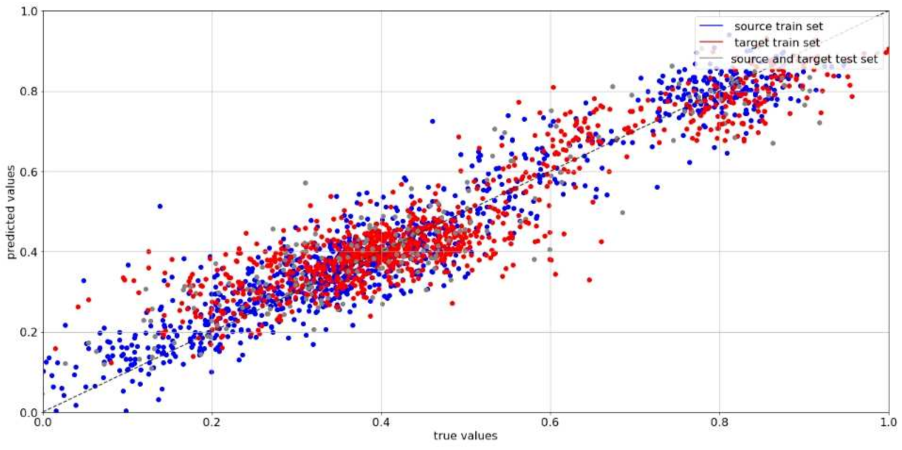


Figure 4.3: Predictions plot of the TCN predictor trained on the combination of A and B data

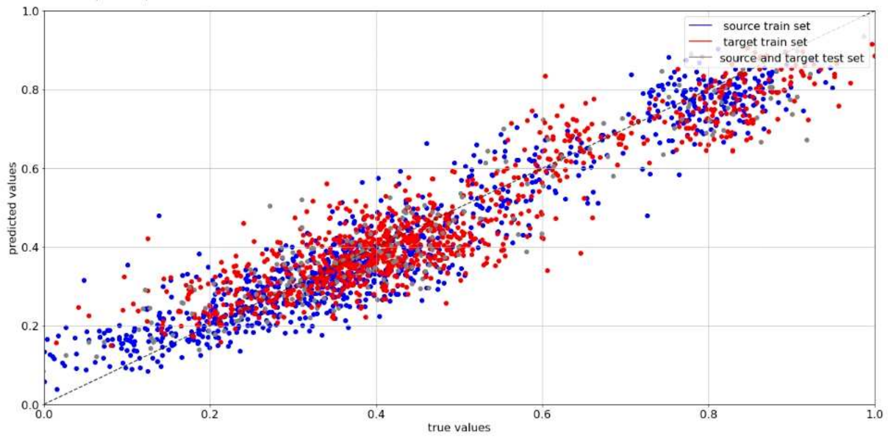


Figure 4.4: Predictions plot of the LSTM predictor trained on the combination of A and B data

The performance results on both the procedures can be seen in 4.1 and 4.2. In this case, as we can see from the prediction plots in 4.3 and 4.4 and from the results, the task is resolved for either the source and the target domains.

TCN predictor's results on the different procedures					
Dataset	Procedure	MAE	ME	EV	R2
Source(B)	Original	0.0560	**0.164**	0.895	0.893
test	First	**0.0537**	0.173	**0.904**	**0.895**
	Second	0.104	0.400	0.806	0.615
Target(A)	Original	0.129	0.357	0.720	0.377
test	First	0.0627	0.263	0.825	0.825
	Second	**0.0565**	**0.202**	**0.857**	**0.857**

Table 4.1: TCN predictor: results on the different procedures

LSTM predictor: results on the different procedures					
Dataset	Procedure	MAE	ME	EV	R2
Source(B)	Original	0.0624	0.165	0.900	0.892
test	First	**0.0533**	**0.158**	**0.903**	**0.903**
	Second	0.161	0.413	0.784	0.219
Target(A)	Original	0.141	0.338	0.689	0.302
test	First	**0.0640**	**0.246**	0.832	**0.825**
	Second	0.0664	0.261	**0.848**	0.817

Table 4.2: LSTM predictor's results on the different procedures

Instead, the second training procedure is a reduced version of the already described one present in the previous chapter. This one aims to add the knowledge to solve the task for the A domain. For doing this, the models are trained on the A domain for fewer epochs since we expect the model to exploit the common characteristics of the two domains already learned from the B data. The epochs, in this case, are 100, but we also expect that the training stops early because of the latter described point. In this case, as we can see from the prediction plots in 4.5 and 4.6 and from the results in 4.1 and 4.2, the task is resolved for the target domain, but the model experiences performance degradation in the source domain, forgetting how to resolve the task for the source domain. With these two approaches, we can state mainly two things. First of all, in an incremental environment, we can't use this method; the addiction of another dataset in the training procedure cause, in the first case, the doubling of the training data, so the doubling of the training time. The epochs are the same

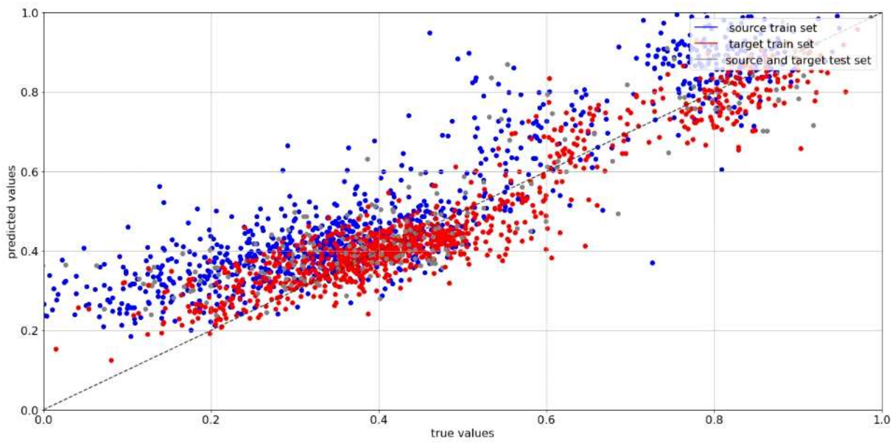


Figure 4.5: Predictions plot of the TCN predictor trained on A data after the training on B data

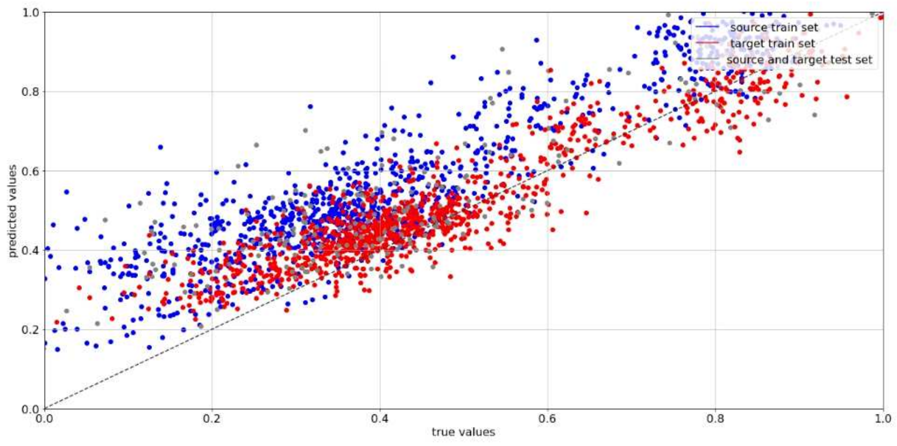


Figure 4.6: Predictions plot of the LSTM predictor trained on A data after the training on B data

in each of them, but with the samples processed doubled. This factor, if the domains were more numerous, would increase accordingly. Instead, in the second case, the new presented samples make the models forget how to solve the task in the previous domain, ruining the previous work done. The second statement is that we can't reuse the previous work done. In the first case, the training starts from scratch, using only the characteristics of the model from the previous results. These, however, do not guarantee good results, especially dealing with amounts of data wider than before. In the second case, further operations ruined the work done previously.

These aspects justify the need for a specific solution to address the domain adaptation task, ensuring that previous results can be adopted without modification to multiple domains without problems. This work addresses this aspect through DBAM, which exploits the already trained predictor without modification and uses the aligner model for the DA part. This choice is preferable to using the DANN model, which involves the additional training of the predictor with the alignment performed on the feature space. The DBAM, instead, permits reliance on a steady model in the prediction part and benefits from the possibility of visually interpreting the results of the aligner since it works on the signal domain instead of the features one. Anyhow, a problem persists as, for each new domain, adapting the old aligner or training a new one is a difficult choice that has to be considered.

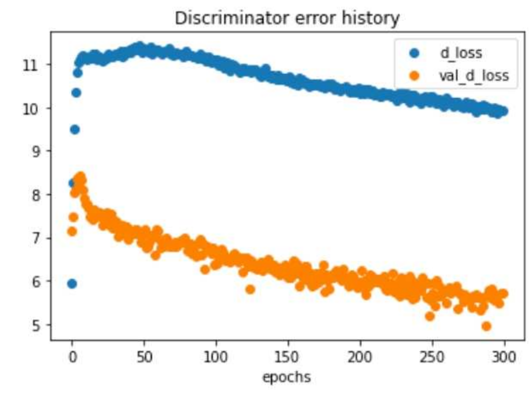
4.3.2 Benchmark DBAM

First of all, we want to present the results achieved from the benchmark, in a way to see the detailed characteristics of his training results to be directed towards subsequent comparisons. The evaluation metrics used for this approach refer to the predictor performances used on the target domain data that have been aligned. These are reported in 4.3, with the results of the previous chapter where the predictor was tested on the original target domain data.

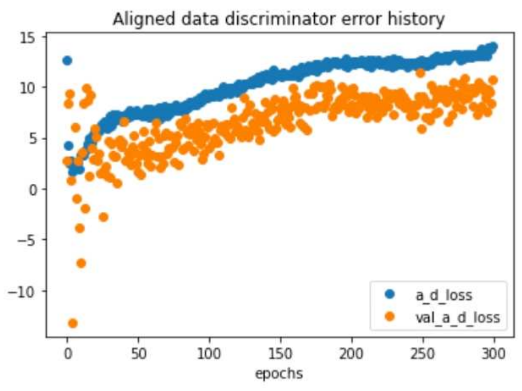
Evaluation of the benchmark DBAM				
Dataset	MAE	ME	EV	R2
Target(A) validation	0.117	0.400	0.590	0.421
Target(A) test	0.118	0.463	0.565	0.393
Target(A) test (no DBAM)	0.227	0.484	0.738	-0.573

Table 4.3: Benchmark DBAM: performance evaluation

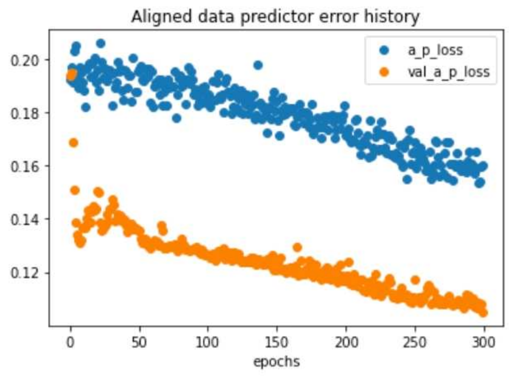
As we can see from the results, the improvements are obtained, mainly in MAE and R2. This one can also be seen in the predictions of 4.9, which are better than the results of 3.6, reported in the precedent chapter. It refers only to red points since blue points are the same as the predictor model doesn't change from the previous chapter, as the source data. However, there are visible issues related to the higher values that aren't predicted very well, so there is room for further improvement.
Another aspect considered is the training history, which makes it possible to see if something went wrong during the learning phase. In the DBAM procedure, as in every adversarial training setup, the outcome of the training metrics isn't straightforward. In this case, we should consider multiple losses, with some of these metrics that don't need to be minimized or maximized. The relevant aspect is to evaluate the stability of the training, even if we also have a



(a) First metric

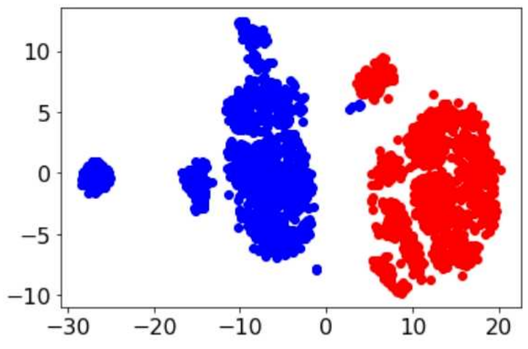


(b) Second metric

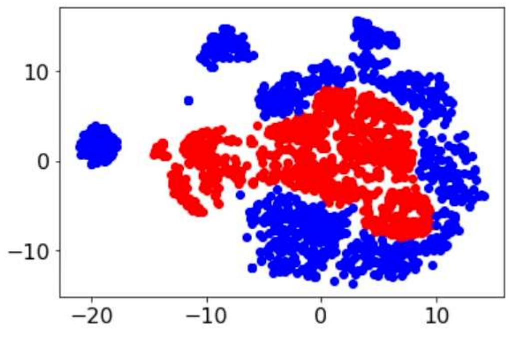


(c) Third metric

Figure 4.7: Benchmark DBAM: training history



(a) Before alignment



(b) After alignment

Figure 4.8: Benchmark DBAM: t-SNE plot of train data

metric that shows the predictor's performance. We choose the three most important to evaluate this stability, among the various metrics concerning the different DBAM components

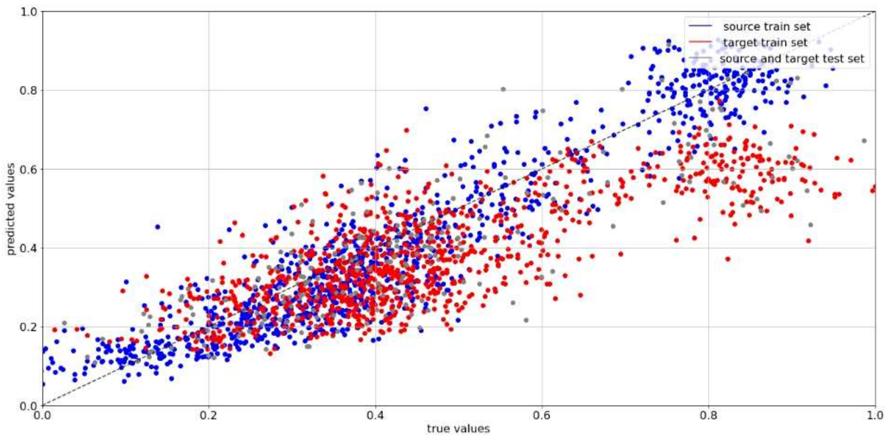


Figure 4.9: Benchmark DBAM: predictions plot

that act on the different parts of the dataset.
In all the metrics, the blue points are the training data results, while the orange points are the validation ones. The first one regards the error of the discriminator on the combination of source and target domains, with the target domain incurring the alignment, which is useful for understanding if the training behaviour is stable. A continuous and steep decrease of this metric means that the discriminator is not challenged by the generator, with the training not performed as needed. This aspect also happens if the values are steeply increasing, meaning that the aligner is predominating. Instead, in the presence of lots of oscillations, the training becomes unstable with the models that are not learning something useful. These aspects are the same for the second metric, which is the part of the discriminator error connected to the aligned target domain only. With this separate part, we can see the isolated behaviour of the aligner to see if it is working. The specific values of these metrics are not so relevant since the important thing is the shape they assume. Instead, the third metric values are important since they represent the evolution of the performances of the predictor; in this plot, we should see that the numbers are decreasing since the plot shows the MAE of the predictions over the data. In the benchmark case, we can see that the training is performing well since the overall discriminator error is stable and also slightly decreasing(the discriminator is improving), and the discriminator error on the aligned data is steady with also a slight increase(the aligner is improving) and the predictor error is decreasing.
Another important aspect is the alignment evaluation since the numerical results with the training error histories can not give the information necessary for the correct behaviour of the DBAM. This one is performed using the t-SNE[46] plot, which indicates the distribu-

tion of the data. This one is applied to the features extracted from the predictor, the output data of the last layer before the fully connected part in the model. What we should expect is the separation of the features of the source and target domains in the beginning, as the predictor should extract different features. Then, after the alignment, these should overlap as the two domains become similar. It's happening with the benchmark DBAM, as shown in 4.8. As usual, the blue points are from the source domain and the red ones are from the target one.

With the benchmark results, we saw an example of DBAM configuration with the specific aspects we considered to evaluate it. These will be used immediately afterwards in the study of the other configurations.

4.3.3 Insertion of the new predictors

The first improvement studied is the replacement of the benchmark predictor with the TCN and LSTM models studied in the previous chapter. In this case, the adversarial training components remain the same as in the benchmark, so we analyse the impact of the new models applied only in the prediction part.

TCNpredictor-DBAM configuration

We show the results on the configuration having only the TCN predictor as the difference from the benchmark configuration, named TCNpredictor-DBAM for simplicity. We can find the alpha parameter optimization in 4.4, where the parameter 0.1 performs the best.

TCNpredictor-DBAM: alpha parameter optimization				
Parameter	MAE	ME	EV	R2
0.1	**0.0900**	**0.317**	**0.689**	**0.659**
0.25	0.0974	0.367	0.611	0.579
0.5	0.0910	0.370	0.669	0.649
1	0.109	0.467	0.478	0.460
5	0.121	0.463	0.384	0.371

Table 4.4: TCNpredictor-DBAM: alpha parameter optimization

However, the results on validation data are not enough, as we should have confirmation regarding the domain alignment. This confirmation is obtained with the t-SNE plot, shown in 4.11. In this case, the t-SNE plots corroborate the statements of the TCN predictor done in

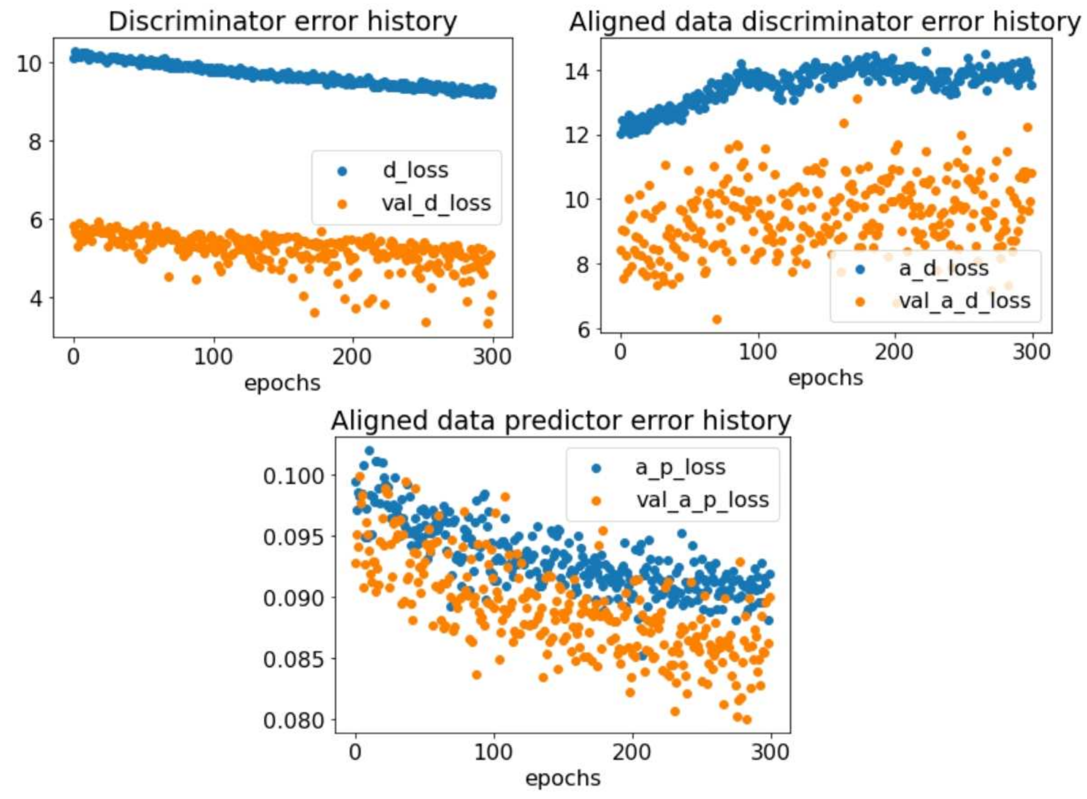
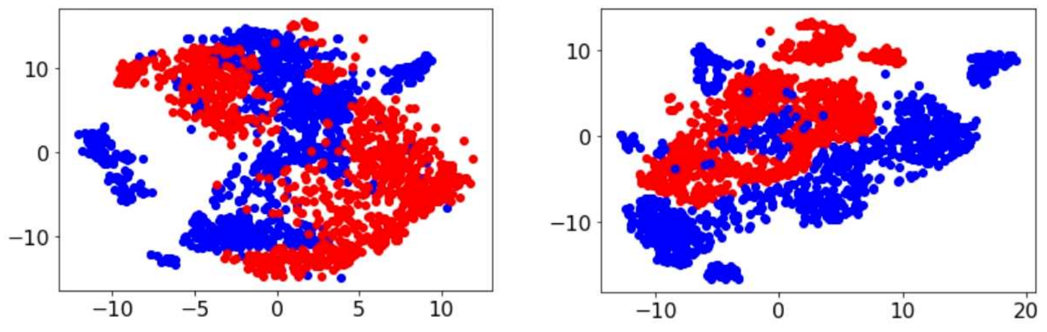


Figure 4.10: TCNpredictor-DBAM: training history



(a) Before alignment **(b)** After alignment

Figure 4.11: TCNpredictor-DBAM: t-SNE plot of train data

the previous chapter. The greater generalization capabilities can also be seen there since the two domains are overlapped before the alignment, which serves only to refine this task. This aspect can also justify the good performances with an alpha parameter of 0.1; the predictor performance is the main focus, but this does not cause problems in the domain alignment as

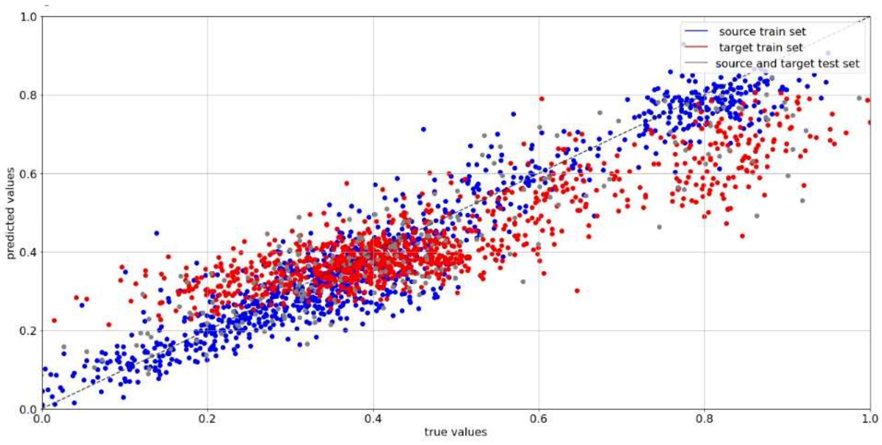


Figure 4.12: TCNpredictor-DBAM: predictions plot

the generalization capabilities of the TCN predictor aid this alignment.
The correct behaviour of the task can also be seen on the training history in 4.10, as the stability of the training is visible. This one permits us not to explore further the discriminator extra steps parameter, as the configuration is working, so we don't need to reduce the training time since it is the same as the benchmark. Finally, the prediction plot in 4.12 is more concentrated towards the target line than the benchmark, demonstrating the configuration improvements again. Later, we will show the final test results with the benchmark and LSTM results.

LSTMpredictor-DBAM configuration

We show the results on the configuration having only the LSTM predictor as the difference from the benchmark configuration, named LSTMpredictor-DBAM for simplicity. We can find the alpha parameter optimization in 4.5, where the parameter of 0.1 performs the best.

LSTMpredictor-DBAM: alpha parameter optimization				
Parameter	MAE	ME	EV	R2
0.1	**0.0780**	**0.340**	**0.739**	**0.736**
0.25	0.0833	0.349	0.711	0.707
0.5	0.101	0.385	0.574	0.560
1	0.101	0.421	0.561	0.556
5	0.106	0.460	0.523	0.518

Table 4.5: LSTMpredictor-DBAM: alpha parameter optimization

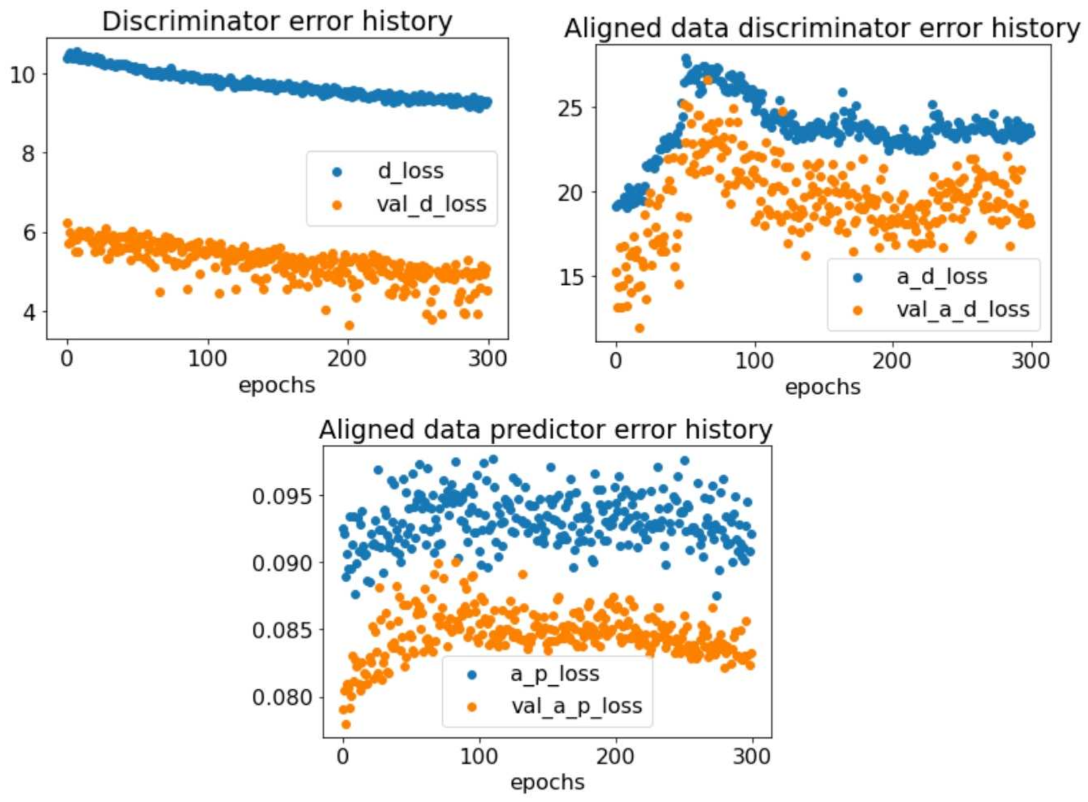
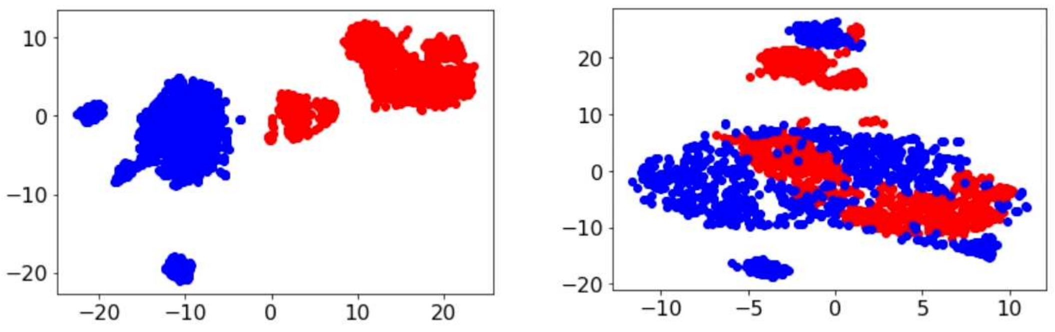


Figure 4.13: LSTMpredictor-DBAM: training history



(a) Before alignment

(b) After alignment

Figure 4.14: LSTMpredictor-DBAM: t-SNE plot of train data

However, the alignment is not happening with this parameter, so we should discard this configuration. Instead, the one with 0.25 as the alpha parameter is working, as we can ascertain on the t-SNE plot in 4.14. In this case, the lack of generalization properties makes it necessary to weigh more the loss related to the alignment.

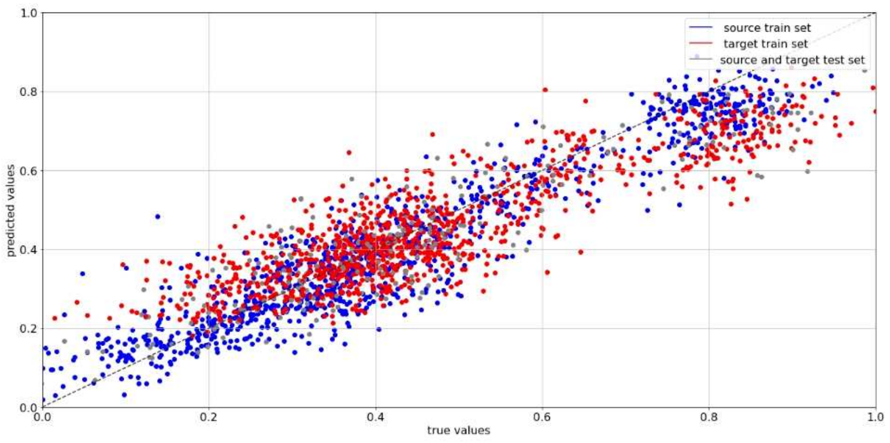


Figure 4.15: LSTMpredictor-DBAM: predictions plot

The correct behaviour of the task can also be seen on the training history in 4.13, as the stability of the training is visible. This one permits us not to explore further the discriminator extra steps parameter, as the configuration is working, so we don't need to reduce the training time since it is the same as the benchmark. Finally, the prediction plot in 4.15 is more concentrated towards the target line than the benchmark, demonstrating the configuration improvements again. The sequent section will show the final test results with the benchmark and TCN results.

Final results

The test results of the different configurations with the new predictor added are compared in 4.6.

Test results of DBAMs with new predictors				
DBAM model	MAE	ME	EV	R2
Benchmark DBAM	0.118	0.463	0.565	0.393
TCNpredictor-DBAM	0.0881	0.387	0.687	0.646
LSTMpredictor-DBAM	**0.0822**	**0.320**	**0.713**	**0.707**

Table 4.6: Test results of DBAMs with new predictor

As we can see, inserting a new predictor improves overall performance. Additionally, the new predictors demonstrate improved readiness to facilitate data alignment, as the required alpha parameter is less than the value of 5 used in the benchmark. This one benefits the prediction results, which are the best for the LSTM architecture, looking at the test results and the predictions in 4.15. The TCN configuration has difficulty in the alignment due to its better generalization properties, a side effect that still allows it to beat the benchmark.

4.3.4 Insertion of the new aligners

The second improvement studied is the replacement of the benchmark aligner with the TCN and LSTM models. In this case, the only component that remains the same as in the benchmark is the discriminator model. With this configuration, we analyze the impact of the new alignment architecture.

TCNaligner-DBAM configuration

We show the results on the configuration with the TCN aligner, named TCNaligner-DBAM, for simplicity. With this new aligner, the evidence of problems emerges at the beginning by looking at the warm-up training behaviour. The architecture introduces a huge amount of

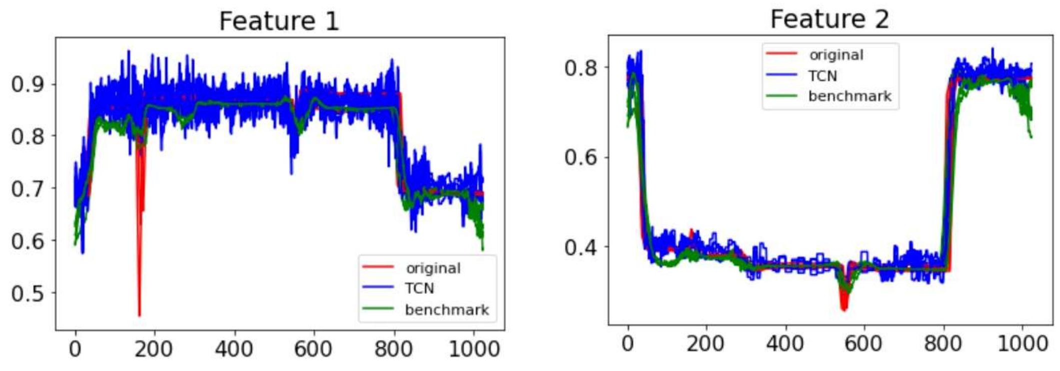


Figure 4.16: TCN aligner: signals examples after warm-up training

noise in the signals reconstruction, as we can see for some signals of the A domain in 4.16. In this figure, some signal samples are plotted in the original form and reconstructed with the pre-trained aligners, which act as simple autoencoders. This bad behaviour happens with all the hyper-parameters tested for this model, demonstrating that the problem isn't derived from the specific parameters. This noise also makes DBAM training problematic as it becomes very unstable as we can see from 4.18, especially for the aligner. This aspect prevents proper training of this DBAM architecture, which generates bad signals to pass to the predictor, as for the signals of the B domain in 4.17 after the training. As for the previous case, some signal samples are plotted in the original form and aligned with the trained models.

Given these results, the objective becomes to understand how to reduce the noise origin.

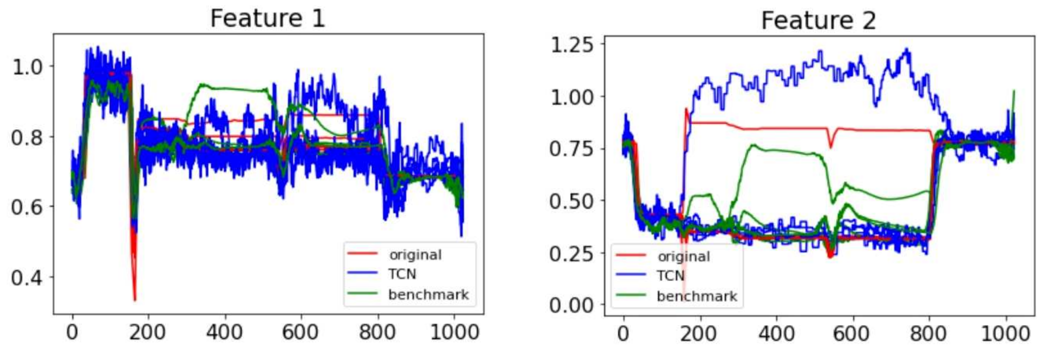


Figure 4.17: TCNaligner-DBAM: signals examples after training

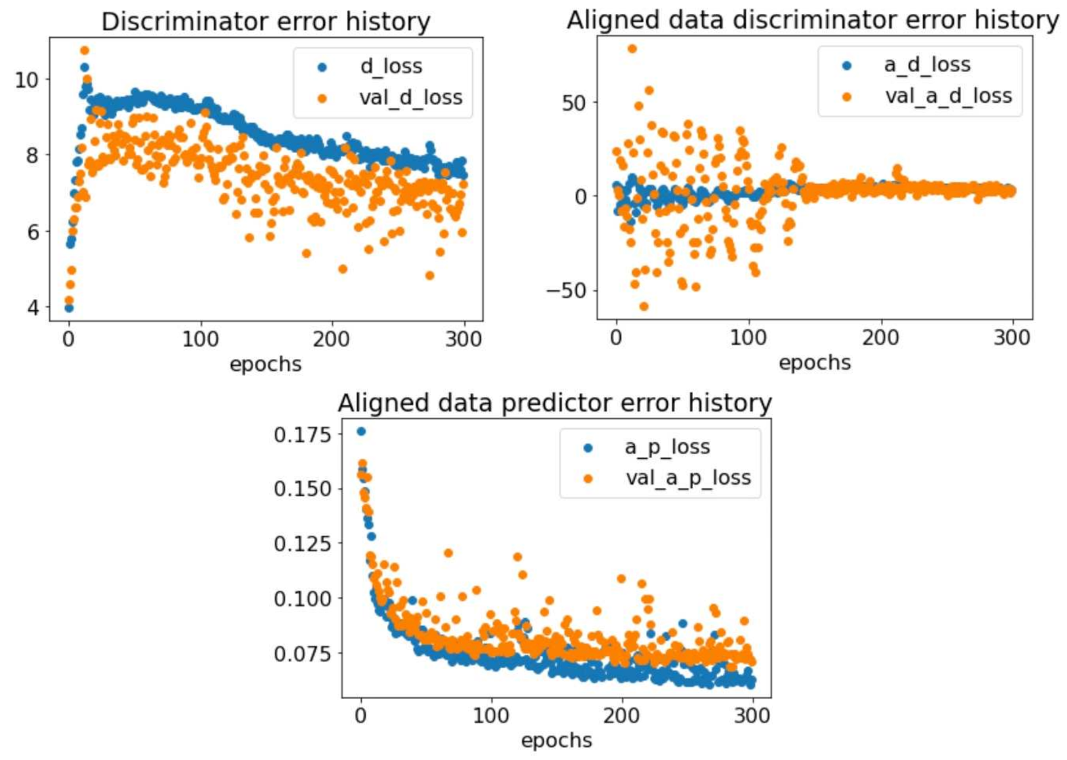


Figure 4.18: TCNaligner-DBAM: training history

The first experiment focuses on a complex 1DCNN structure, extending and complicating the benchmark elements in a way to see if the noise shows up with a complex architecture in general: the noise did not show up, meaning that the problem doesn't come from the complexity of the architecture but a particular aspect of the TCN aligner.

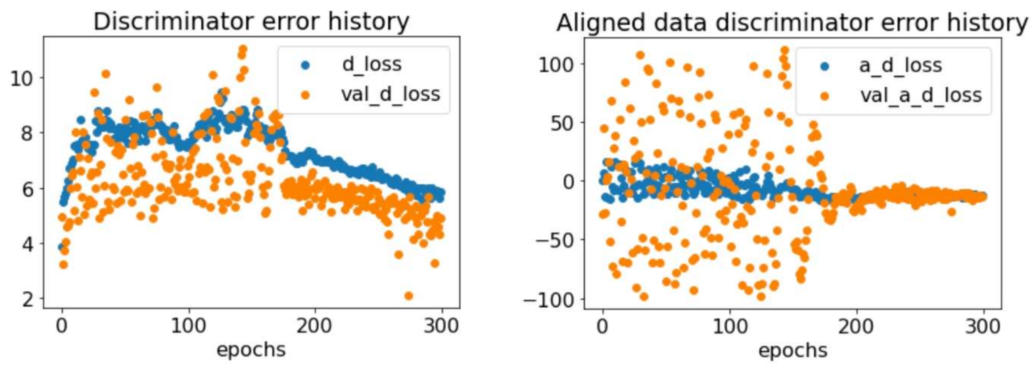


Figure 4.19: TCNaligner-DBAM: training history of the improved architecture

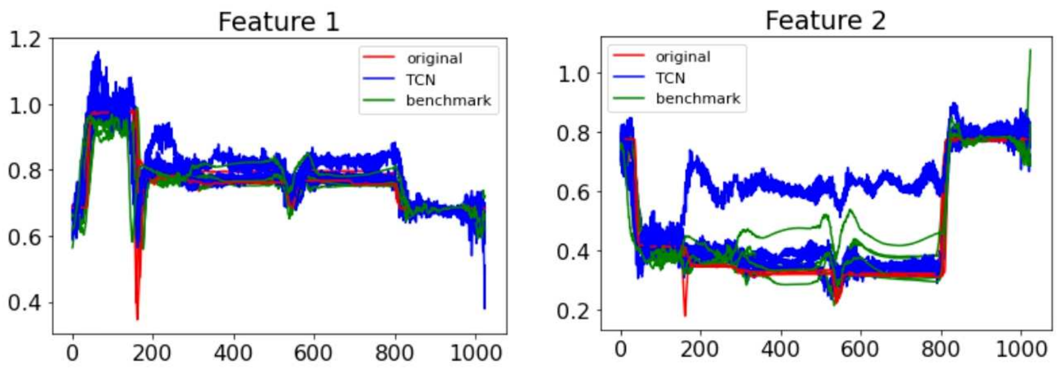


Figure 4.20: TCNaligner-DBAM: signals examples of the improved architecture

This statement leads us to use 1DCNN layers before TCN to reduce the signal dimension before the TCN layers computations and see if this reduces the noise. This idea is based on the example of the predictor architecture with 1DCNN followed by the LSTM layers[33] which, however, performs worse than the standard TCN architecture when looking at hyper-parameters optimization results and noise in the warm-up training. This one also happens by trying to invert the position of the 1DCNN and TCN components, an aspect based on TCN predictor architecture. The next choice is oriented to reduce the influence of the TCN elements, with the use of TCN layers followed by 1DCNN layers in the encoder part and the use of only 1DCNN elements in the decoder part. The behaviour improved a bit, but not sufficient since most of the noise is still present and the DBAM training, tested in this case, shows the same instability problems. The last attempt was to simplify the TCN layer by diminishing the dilated convolution depth and using the repeat function to upsampling the compressed features, as used in the LSTM aligner[45]. Also, in this case, the results are almost the same as the previous point; this leads to thinking that the TCN layers are not suited for building an effective aligner architecture. The training history of the last configuration tested is shown in 4.19 and the signals in 4.20. These results conclude the analysis of the TCN-DBAM elements since the discriminator study without a functioning aligner would be meaningless.

LSTMaligner-DBAM configuration

We show the results on the configuration with the LSTM aligner, named LSTMaligner-DBAM for simplicity. In the warm-up training the chosen aligner has the following parameters, given the results of the optimization procedure:

- **Number of LSTM layers**: 1;
- **Number of units**: 50;
- **Dropout rate**: 0;
- **Type of compression**: Average pooling.

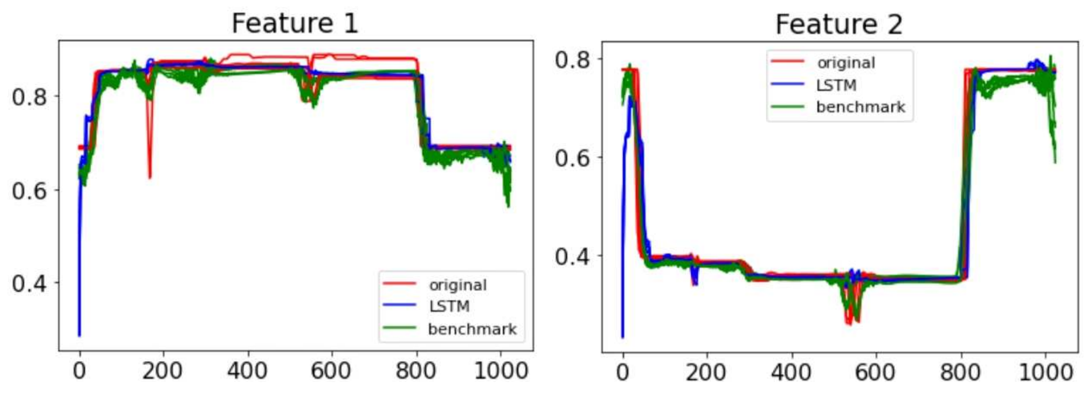


Figure 4.21: LSTM aligner: signals examples after warm-up training

The signals, as we can see from 4.21, result more regular than the benchmark, demonstrating that the LSTM aligner is doing his job well. Regarding the DBAM architecture, we can find the alpha parameter optimization in 4.7, where the parameter 0.25 performs the best.

LSTMaligner-DBAM: alpha parameter optimization				
Parameter	MAE	ME	EV	R2
0.1	0.0872	0.401	0.704	0.663
0.25	**0.0817**	0.364	**0.708**	**0.707**
0.5	0.0823	0.366	0.705	0.703
1	0.0863	0.377	0.686	0.685
5	0.0879	**0.351**	0.684	0.670

Table 4.7: LSTMaligner-DBAM: alpha parameter optimization

LSTMaligner-DBAM: discriminator extra steps parameter optimization				
Parameter	MAE	ME	EV	R2
5	0.0878	0.379	0.670	0.670
10	0.0827	**0.361**	0.702	0.702
20	**0.0817**	0.364	**0.708**	**0.707**

Table 4.8: LSTMaligner-DBAM: discriminator extra steps parameter optimization

In this case, we investigate the discriminator extra steps parameter as introducing a more complicated aligner causes the elongation of the training time. The objective is to see if the performance can retain even if the discriminator participation is lowered. This aspect doesn't happen since the performance degrades as reported in 4.8, also presenting more difficulties in the alignment. The other aspects of the configuration with 0.25 as the alpha parameter and 20 as the discriminator extra steps parameter show that the architecture is working. This includes the training history in 4.22, the predictions plot in 4.24 and the t-SNE plot in 4.23. Finally, the test results are presented in 4.9. These compare the actual configuration with the benchmark and the LSTMpredictor-DBAM configuration.

LSTMaligner-DBAM: test results				
DBAM model	MAE	ME	EV	R2
Benchmark DBAM	0.118	0.463	0.565	0.393
LSTMpredictor-DBAM	**0.0822**	**0.320**	**0.713**	**0.707**
LSTMaligner-DBAM	0.0856	0.342	0.693	0.691

Table 4.9: LSTMaligner-DBAM: test results

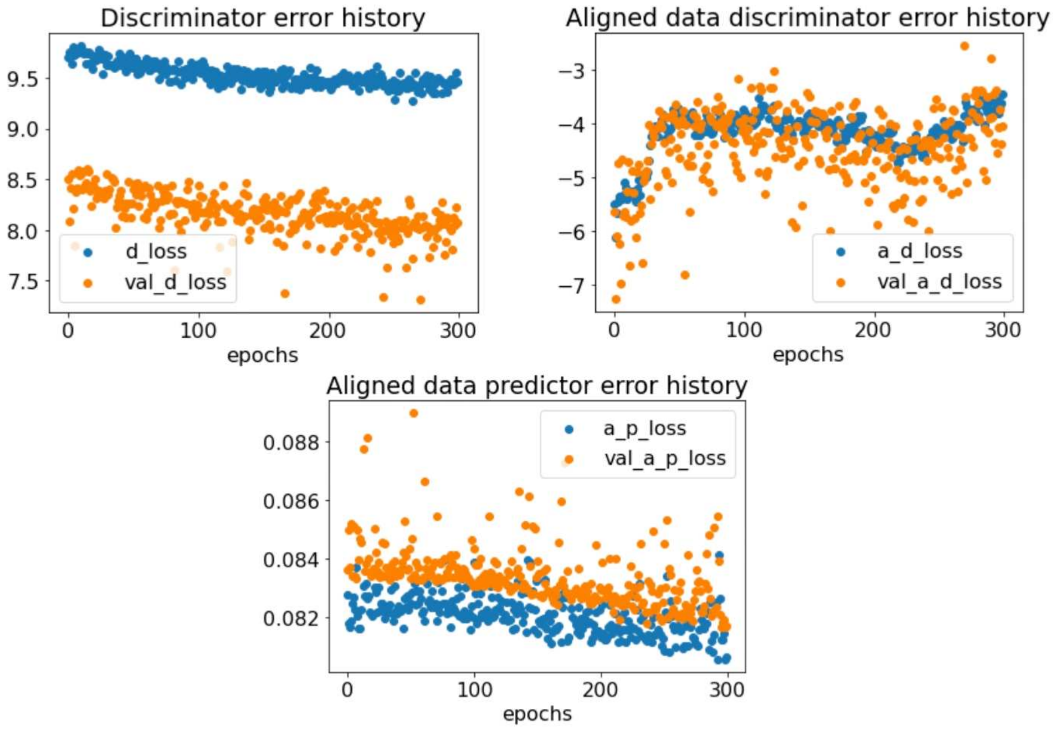
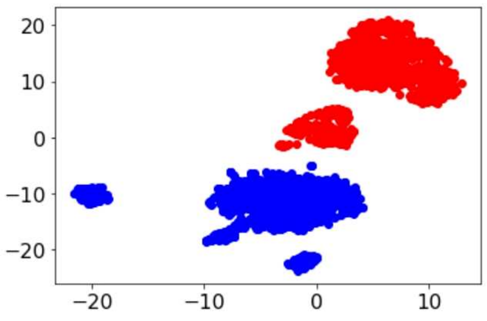
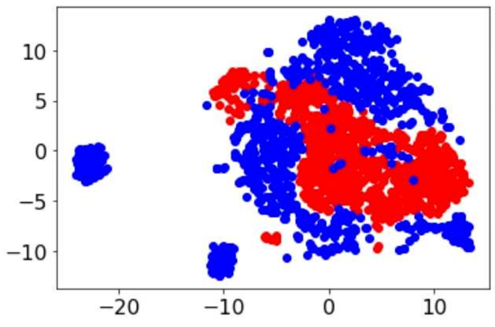


Figure 4.22: LSTMaligner-DBAM: training history



(a) Before alignment



(b) After alignment

Figure 4.23: LSTMaligner-DBAM: t-SNE plot of train data

It is clear that the benchmark is beaten in this case too, but introducing the aligner does not improve the performance, which remains the same. However, an improvement can be seen in the signals generated by the aligner in 4.25 which are smoother than the benchmark aligner although failing to capture abrupt movements. The latter behaviour does not change in the LSTMpredictor-DBAM configuration since it depends only on the architecture of the aligner, then the new predictor does not affect it. Therefore, with the LSTMaligner-

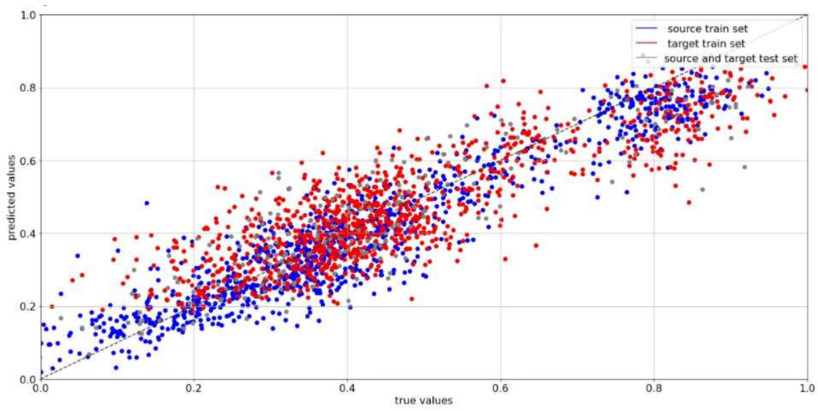


Figure 4.24: LSTMaligner-DBAM: predictions plot

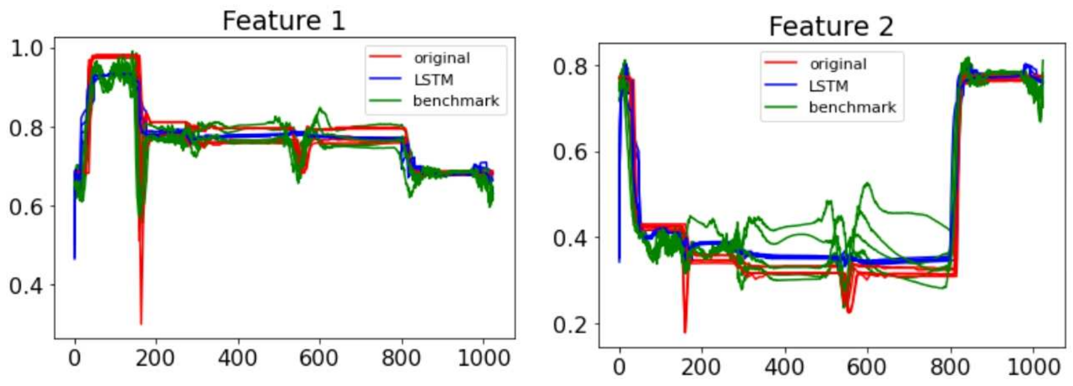


Figure 4.25: LSTMaligner-DBAM: signals examples after training

DBAM configuration, the aligned signals are more regular than the LSTMpredictor-DBAM, at the expense of a greater computational complexity given by the more complicated aligner architecture.

4.3.5 Insertion of the new discriminator

The last improvement studied is replacing the benchmark discriminator with the LSTM model, applied only for the LSTM configuration as the TCN one does not work. With this addition, all parts of the configuration own LSTM elements.

LSTMdiscriminator-DBAM configuration

We show the results on the configuration described above, named simply LSTMdiscriminator-DBAM. We can find the alpha parameter optimization in 4.10, where the parameter of

0.25 performs the best. Given the previous results on the LSTMpredictor-DBAM and the LSTMaligner-DBAM, the 0.1 alpha parameter is not tested.

LSTMdiscriminator-DBAM: alpha parameter optimization				
Parameter	MAE	ME	EV	R2
0.25	**0.0873**	**0.367**	**0.679**	**0.675**
0.5	0.0881	0.369	0.670	0.670
1	0.0918	0.373	0.650	0.648
5	0.0951	**0.367**	0.630	0.626

Table 4.10: LSTMdiscriminator-DBAM: alpha parameter optimization

In this case, the discriminator extra steps parameter is investigated. The new discriminator enormously increases the required time in the DBAM training, as said previously in its architecture description. This aspect induces seeking solutions to improve it, achieved by lowering the discriminator extra steps parameter. Furthermore, the discriminator is also more complex, so reducing its influence on training can better balance the competition between the aligner and become beneficial for training. It occurs as a performance improvement is present, as reported in 4.11, so the value of 5 is chosen for this parameter.

LSTMdiscriminator-DBAM: discriminator extra steps parameter optimization				
Parameter	MAE	ME	EV	R2
5	**0.0861**	0.384	**0.685**	0.679
10	0.0869	**0.358**	**0.685**	**0.681**
20	0.0873	0.367	0.679	0.675

Table 4.11: LSTMdiscriminator-DBAM: discriminator extra steps parameter optimization

The other aspects of the configuration with 0.25 as the alpha parameter and 5 as the discriminator extra steps parameter show that the architecture is working. This includes the training history in 4.26, the predictions plot in 4.28, the t-SNE plot in 4.27 and the signals in 4.29. The introduction of the more complex discriminator makes, at first, the aligner training a bit more unstable, but it takes the right path after a while.

LSTMdiscriminator-DBAM: test results				
DBAM model	MAE	ME	EV	R2
Benchmark DBAM	0.118	0.463	0.565	0.393
LSTMaligner-DBAM	**0.0856**	0.342	**0.693**	**0.691**
LSTMdiscriminator-DBAM	0.0917	**0.324**	0.676	0.663

Table 4.12: LSTMdiscriminator-DBAM: test results

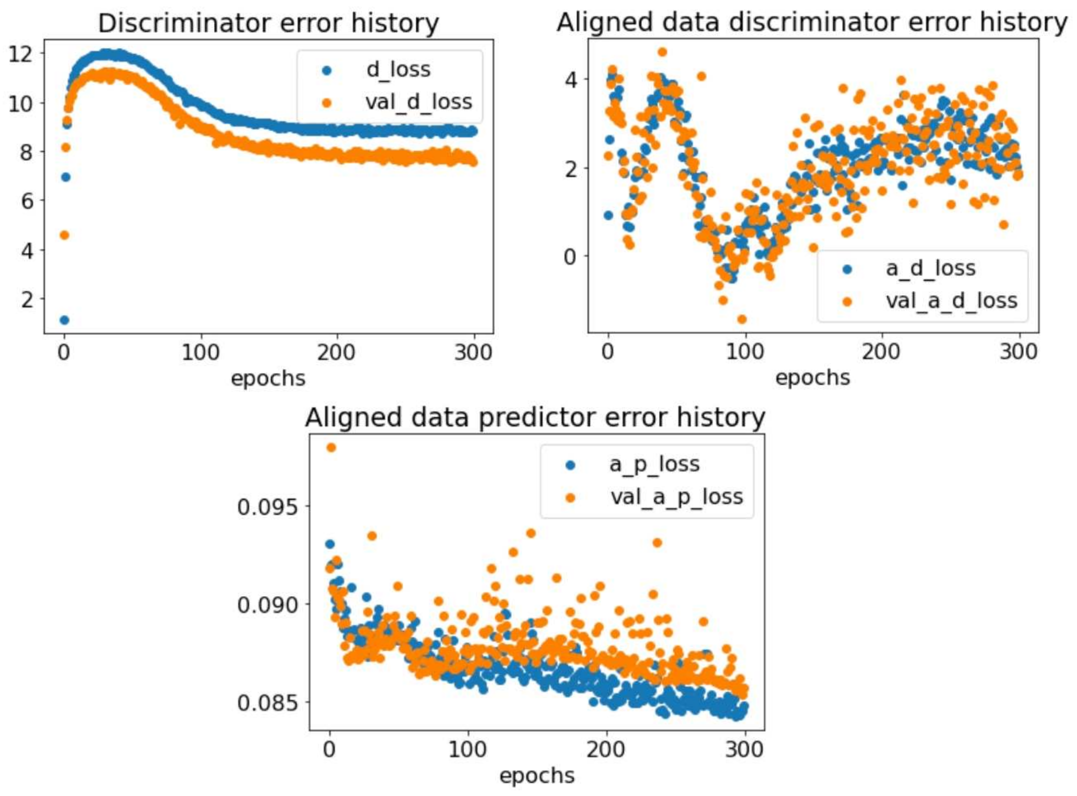
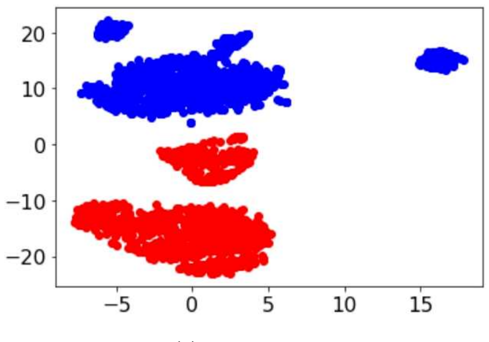


Figure 4.26: LSTMdiscriminator-DBAM: training history

Finally, the test results in 4.12 show that the configuration beats the benchmark also in this case. However, introducing the new discriminator doesn't improve the results compared to the LSTMaligner-DBAM configuration. Instead, the training is longer and more complicated, thus making use of this configuration inadvisable.


(a) Before alignment

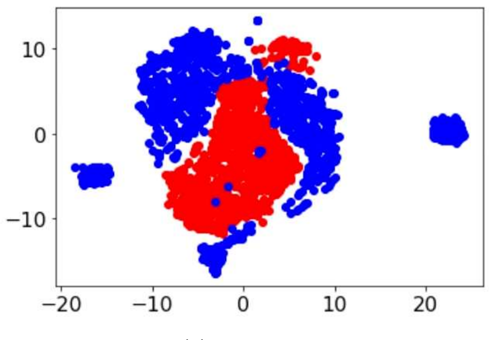

(b) After alignment

Figure 4.27: LSTMdiscriminator-DBAM: t-SNE plot of train data

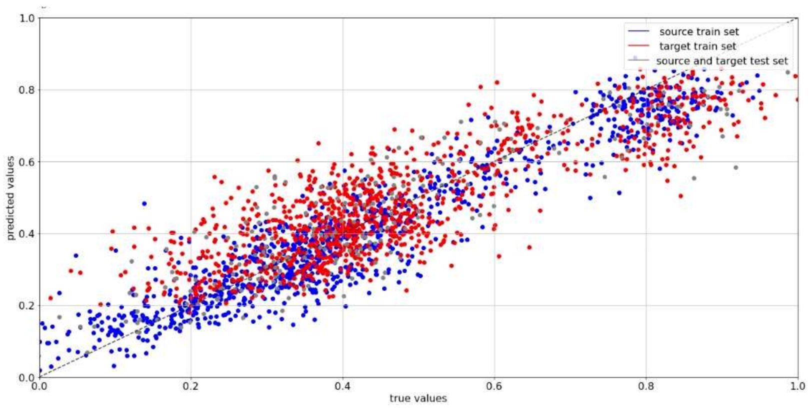


Figure 4.28: LSTMdiscriminator-DBAM: predictions plot

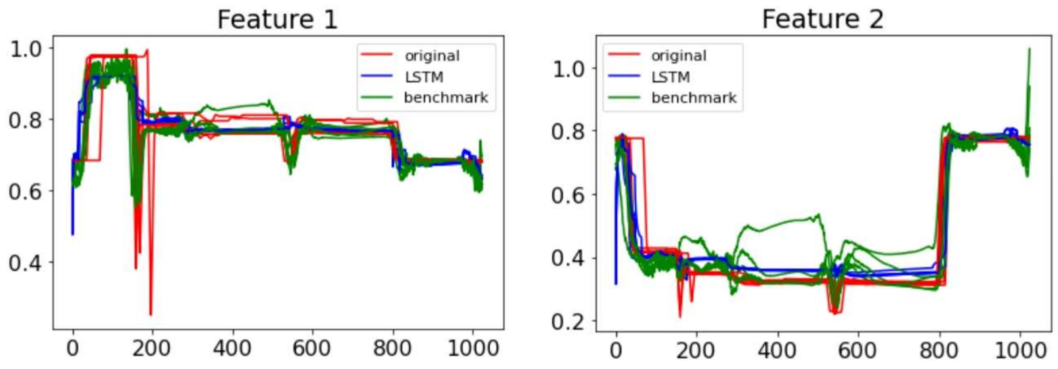


Figure 4.29: LSTMdiscriminator-DBAM: signals examples after training

4.3.6 Results recap and cross-validation results

Among all the configurations, some improvements result better than the others. Changing the predictor is the most beneficial improvement, making the performance much better than

the benchmark in both the TCN and LSTM case. However, the TCN, while generalizing better, makes it more problematic to achieve a stable final configuration. Its properties make alignment more difficult using the benchmark aligner, with problems worsening with the addition of the TCN aligner, which does not work. This aspect leads to the absence of the TCN discriminator test and the consideration of TCN failing to perform this task. Instead, LSTM elements worked well with the introduction of an aligner which generates more stable signals. However, this has the same prediction performance compared to using the aligner benchmark, making the choice of the type indifferent if there is no preference on the shape of the signals. Even the LSTM discriminator gives the same results as the benchmark. However, its training is longer and more complex, making its use not recommendable.
These considerations lead to the decision to apply CV to the LSTMaligner-DBAM configuration. The shape of the signals is not relevant in the research that focuses more on using new architectures, thus preferring the architecture with more LSTM elements.

Cross-validation results of the Domain Adaptation task				
DBAM model	MAE	ME	EV	R2
Benchmark DBAM	0.110 $\pm$ 0.008	0.423 $\pm$ 0.07	0.506 $\pm$ 0.05	0.462 $\pm$ 0.07
LSTMaligner-DBAM	**0.0820** $\pm$ 0.007	**0.305** $\pm$ 0.05	**0.712** $\pm$ 0.03	**0.708** $\pm$ 0.02

Table 4.13: Cross-validation results of the Domain Adaptation task

The average test results(with the standard deviation indicated by the $\pm$ symbol) of the cross-validation procedure, reported in 4.13, confirm the previous statements as the models have almost identical performances compared to the results of the first data arrangement. This aspect proves that the previous models do not depend on the particular samples used for their formation. However, compared to the results of the reference research, these have discrepancies as the benchmark is performing worse than them. Details in the implementation or procedure not reported in that work may have determined these differences, such as, for example, the version of the python libraries or the data usage. Nevertheless, these discrepancies don't influence the comparison results as, in this work, the same method is applied to every architecture type.

5 Conclusion

The main purpose of this work was to improve the architecture of the reference research [1, 2] with the use of more sophisticated models such as TCN and LSTM. These are models more suited to deal with temporal information, proving it also in this research.
In the first part of the work, the results of the predictors showed superiority over the benchmark, improving the reliability of the estimate of the measure sought, reaching the desired goal. These improvements would translate into cost savings and increased productivity in a process environment.
In the second part of the work, the use of the Domain Adaptation procedure showed its benefits; introducing the DBAM methodology that can standardize data distributions belonging to similar environments is crucial to permit the exploitation of all the capabilities of the prediction model. With this task more complex than the previous one, the tested TCN models failed to reach the goal and compete with the benchmark. This, instead, was performed by the LSTM models which were capable of solving this task, however not improving compared to the benchmark elements, except for some minor aspects.
In any case, the various architectures have not been explored beyond the aspects necessary for comparison, which means that there is room for improvement as, for example, the LSTM models in DBAM still have potential. The latter may be the first hint for future work, including the further exploration of types of neural network models. Another aspect to consider for future work is implementing this method in a production line since this research bases its results on a controlled and simplified environment. It means it is needed to confirm the

actual performance and benefits in a real application example.
The main contributions of this work concur deepening a prediction task with the assistance of Domain Adaptation, a field still in its early stages with limited applications and scarce literature backing it.

References

[1] N. Gentner, M. Carletti, A. Kyek, G. A. Susto, and Y. Yang, "Dbam: Making virtual metrology/soft sensing with time series data scalable through deep learning," *Control Engineering Practice*, vol. 116, p. 104914, 2021.

[2] N. Gentner, A. Kyek, Y. Yang, M. Carletti, and G. A. Susto, "Enhancing scalability of virtual metrology: a deep learning-based approach for domain adaptation," in *2020 Winter Simulation Conference (WSC)*. IEEE, 2020, pp. 1898–1909.

[3] H. Hassan, A. Negm, M. Zahran, and O. Saavedra, "Assessment of artificial neural network for bathymetry estimation using high resolution satellite imagery in shallow lakes: Case study el burullus lake." *International Water Technology Journal*, vol. 5, 12 2015.

[4] S. Kiranyaz, O. Avci, O. Abdeljaber, T. Ince, M. Gabbouj, and D. J. Inman, "1d convolutional neural networks and applications: A survey," *Mechanical Systems and Signal Processing*, vol. 151, p. 107398, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0888327020307846

[5] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. W. Senior, and K. Kavukcuoglu, "Wavenet: A generative model for raw audio," *ArXiv*, vol. abs/1609.03499, 2016.

[6] X. Yuan, L. Li, and Y. Wang, "Nonlinear dynamic soft sensor modeling with supervised long short-term memory network," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 5, pp. 3168–3176, 2020.

[7] J. Zhang, Y. Li, J. Tian, and T. Li, "Lstm-cnn hybrid model for text classification," in *2018 IEEE 3rd Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*, 2018, pp. 1675–1680.

[8] A. Sagheer and M. Kotb, “Unsupervised pre-training of a deep lstm-based stacked autoencoder for multivariate time series forecasting problems,” *Scientific Reports*, vol. 9, p. 19038, 12 2019.

[9] G. S. May and C. J. Spanos, *Fundamentals of semiconductor manufactoring and process control*. John Wiley Sons, Inc., Hoboken, New Jersey, 2006, vol. 1.

[10] G. A. Susto, S. Pampuri, A. Schirru, G. D. Nicolao, and S. F. McLoone, “Automatic control and machine learning for semiconductor manufacturing: Review and challenges,” in *The 10th European Workshop on Advanced Control and Diagnosis(ACD)*, 2012.

[11] J. Ringwood, S. Lynn, G. Bacelli, B. Ma, E. Ragnoli, and S. Mcloone, “Estimation and control in semiconductor etch: Practice and possibilities,” *Semiconductor Manufacturing, IEEE Transactions on*, vol. 23, pp. 87 – 98, 03 2010.

[12] G. A. Susto and A. Beghi, “Least angle regression for semiconductor manufacturing modeling,” in *2012 IEEE International Conference on Control Applications*, 2012, pp. 658–663.

[13] C. Park and S. B. Kim, “Virtual metrology modeling of time-dependent spectroscopic signals by a fused lasso algorithm,” *Journal of Process Control*, vol. 42, pp. 51–58, 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0959152416300294

[14] C.-H. Chen, W.-D. Zhao, T. Pang, and Y.-Z. Lin, “Virtual metrology of semiconductor pvd process based on combination of tree-based ensemble model,” *ISA Transactions*, vol. 103, pp. 192–202, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0019057820301397

[15] S. Kang, “On effectiveness of transfer learning approach for neural network-based virtual metrology modeling,” *IEEE Transactions on Semiconductor Manufacturing*, vol. 31, no. 1, pp. 149–155, 2018.

[16] K. Lee and C. Kim, “Recurrent feature-incorporated convolutional neural network for virtual metrology of the chemical mechanical planarization process,” *J Intell Manuf*, vol. 31, p. 73–86, 2020.

[17] X. Wu, J. Chen, L. Xie, L. L. T. Chan, and C.-I. Chen, "Development of convolutional neural network based gaussian process regression to construct a novel probabilistic virtual metrology in multi-stage semiconductor processes," *Control Engineering Practice*, vol. 96, p. 104262, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S096706611930214X

[18] Y. Bengio, I. Goodfellow, and A. Courville, *Deep learning*. MIT press Massachusetts, USA:, 2017, vol. 1.

[19] S. Ben-David, J. Blitzer, K. Crammer, A. Kulesza, F. Pereira, and J. W. Vaughan, "A theory of learning from different domains," *Machine learning*, vol. 79, no. 1, pp. 151–175, 2010.

[20] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An introduction to statistical learning*. Springer, 2013, vol. 112.

[21] E. Hajiramezanali, S. Z. Dadaneh, A. Karbalayghareh, M. Zhou, and X. Qian, "Bayesian multi-domain learning for cancer subtype discovery from next-generation sequencing count data," *32nd Conference on Neural Information Processing Systems (NeurIPS 2018), Montréal, Canada*, 2018.

[22] I. B. Arief-Ang, F. D. Salim, and M. Hamilton, "Da-hoc: Semi-supervised domain adaptation for room occupancy prediction using co<sub>2</sub> sensor data," in *Proceedings of the 4th ACM International Conference on Systems for Energy-Efficient Built Environments*, ser. BuildSys '17. New York, NY, USA: Association for Computing Machinery, 2017. [Online]. Available: https://doi.org/10.1145/3137133.3137146

[23] B. Bikramjit and P. Stone, "General game learning using knowledge transfer," *IJCAI*, 2007.

[24] R. Rajat, A. Y. Ng, and D. Koller, "Constructing informative priors using transfer learning," *Twenty-third International Conference on Machine Learning*, 2006.

[25] D. S. Maitra, U. Bhattacharya, and S. K. Parui, "Cnn based common approach to handwritten character recognition of multiple scripts," in *2015 13th International Conference on Document Analysis and Recognition (ICDAR)*, 2015, pp. 1021–1025.

[26] U. Budak, A. Şengür, A. B. Dabak, and M. Çibuk, “Transfer learning based object detection and effect of majority voting on classification performance,” in *2019 International Artificial Intelligence and Data Processing Symposium (IDAP)*, 2019, pp. 1–4.

[27] L. Zhang, “Transfer adaptation learning: A decade survey,” *CoRR*, vol. abs/1903.04687, 2019. [Online]. Available: http://arxiv.org/abs/1903.04687

[28] S. Bai, J. Z. Kolter, and V. Koltun, “An empirical evaluation of generic convolutional and recurrent networks for sequence modeling,” *ArXiv*, vol. abs/1803.01271, 2018.

[29] S. Hochreiter and J. Schmidhuber, “Long Short-Term Memory,” *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 11 1997. [Online]. Available: https://doi.org/10.1162/neco.1997.9.8.1735

[30] C. Olah, “Understanding lstm networks,” 2015. [Online]. Available: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

[31] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” 2017.

[32] Y. Cao, Y. Ding, M. Jia, and R. Tian, “A novel temporal convolutional network with residual self-attention mechanism for remaining useful life prediction of rolling bearings,” *Reliability Engineering System Safety*, vol. 215, p. 107813, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0951832021003355

[33] R. Mutegeki and D. S. Han, “A cnn-lstm approach to human activity recognition,” in *2020 International Conference on Artificial Intelligence in Information and Communication (ICAIIC)*, 2020, pp. 362–366.

[34] N. Reimers and I. Gurevych, “Optimal hyperparameters for deep lstm-networks for sequence labeling tasks,” 2017. [Online]. Available: https://arxiv.org/abs/1707.06799

[35] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: a simple way to prevent neural networks from overfitting,” *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[36] F. Chollet *et al.*, “Keras,” https://keras.io, 2015.

[37] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[38] Y. Ganin, E. Ustinova, H. Ajakan, P. Germain, H. Larochelle, F. Laviolette, M. Marchand, and V. Lempitsky, "Domain-adversarial training of neural networks," 2016.

[39] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," *Advances in neural information processing systems*, vol. 27, 2014.

[40] G. Antipov, M. Baccouche, and J.-L. Dugelay, "Face aging with conditional generative adversarial networks," 2017. [Online]. Available: https://arxiv.org/abs/1702.01983

[41] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural computation*, vol. 1, no. 4, pp. 541–551, 1989.

[42] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein gan," 2017.

[43] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville, "Improved training of wasserstein gans," *CoRR*, vol. abs/1704.00028, 2017. [Online]. Available: http://arxiv.org/abs/1704.00028

[44] M. Thill, W. Konen, H. Wang, and T. Bäck, "Temporal convolutional autoencoder for unsupervised anomaly detection in time series," *Applied Soft Computing*, vol. 112, p. 107751, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1568494621006724

[45] D. Li, L. Li, X. Li, Z. Ke, and Q. Hu, "Smoothed lstm-ae: A spatio-temporal deep model for multiple time-series missing imputation," *Neurocomputing*, vol. 411, 05 2020.

[46] L. van der Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of Machine Learning Research*, vol. 9, no. 86, pp. 2579–2605, 2008. [Online]. Available: http://jmlr.org/papers/v9/vandermaaten08a.html