Available online at www.sciencedirect.com

ScienceDirect

Procedia CIRP 107 (2022) 611–616

www.elsevier.com/locate/procedia

55th CIRP Conference on Manufacturing Systems

Early Quality Prediction using Deep Learning on Time Series Sensor Data

Amal Saadallah[a,], Omar Abdulaaty[a], Jan Büscher[b], Thorben Panusch[b], Katharina Morik[a], Jochen Deuse[b,c]

[a]*Artificial Intelligence Group, TU Dortmund University, Otto-Hahn-Str. 12, 44227 Dortmund, Germany*
[b]*Institute of Production Systems, TU Dortmund University, Leonhard-Euler-Str. 5, 44227 Dortmund, Germany*
[c]*Centre for Advanced Manufacturing, University of Technology Sydney, Sydney, Australia*

* Corresponding author. *E-mail address:* amal.saadallah@cs.tu-dortmund.de, jan.buescher@ips.tu-dortmund.de

Abstract

In manufacturing systems, early quality prediction enables the execution of corrective measures as early as possible in the production chain, avoiding thus costly rework and waste of resources. The increasing development of Smart Factory Sensors and Industrial Internet of Things has offered wide opportunities for applying data-driven approaches for early quality prediction in real-time using Machine Learning (ML). With the multiplication of applications, further requirements on the quality of predictive ML models covering multiple aspects such as accuracy, robustness and explainability have to be fulfilled to build trustworthy ML-based solutions. In this context, we investigate the task of early quality classification using a Convolutional Neural Network (CNN) on time series sensor data of an automotive real-world case study. To do so, a gradient-based heat-mapping explanation method for CNNs is computed to determine the most discriminative time series patterns and localize them in time. These patterns are subsequently used to achieve quality prediction in real-time as early as possible.

Keywords: Early Quality Prediction; Sensor Data; Time Series; Deep Learning; Heat-maps; Automotive Manufacturing.

1. Introduction

In manufacturing systems, quality deviations that are only detected at the end of the production chain, may potentially result in high amounts of rejected products which require laboriously and costly rework or need to be scrapped [10]. To prevent such events, early quality prediction have to be achieved. Hence, corrective actions are expected to have the largest impact if they are executed as early as possible in the process, avoiding thus costly rework and waste of resources through further processing of defective components [6, 13].

A key requirement for early quality prediction is the full coverage of the product quality in the early stages of the manufacturing process or within the execution of some stages. Nowadays, in the context of industry 4.0, the linkage of production environment through Information and Communication Technologies (ICT) to cyber-physical systems with the goal of monitoring, controlling and optimizing complex manufacturing systems, enables real-time capable approaches for process data acquisition, analysis and knowledge discovery [12, 10]. This is achieved in practice by collecting and analyzing sensor data. As a result, data-driven approaches for predicting process quality in real-time and deriving adequate process control interventions in a timely manner, can be developed [9, 6, 3, 5]. Sensors are able to generate mass quantities of sensor data as responses to some types of inputs from the physical production environment. Most often, each of these data points are captured at specific time stamps, effectively transforming sensor data into time series data that can be analyzed across this additional dimension [12, 13].

Machine Learning (ML) algorithms trained on sensor data are used to gain knowledge that can be generalized and used to predict unknown future events, i.e. new data points. Quality prediction that is performed using ML, is referred in Industry 4.0 as model-based quality prediction [10]. To do so, the description of the product or process quality and all the related information should be done at the first step, especially in highly complex dynamic production systems with non-linear interactions between their steps. The second step consists of building and training a predictive model that maps the available quality-related information, e.g, operating states sensor data, process input parameters, to the resulting process/product quality [12]. This model can be used afterwards for predicting the expected quality given

10.1016/j.procir.2022.05.034

a set of input values. Several industrial applications utilize already existing ML methods and algorithms to solve actual problems in manufacturing from an engineering point of view [10]. These applications cover a wide range of industrial fields, including electronics [10], metal [6, 13, 5], and process industries [9, 3]. Likewise, the adopted ML solutions are not limited to a specific family of models but include amongst others Artificial Neural Networks (ANNs) [9], Support Vector Machines (SVMs) [13], and Decision Trees (DTs) [6]. Most of the aforementioned works focused on quality prediction at the end of the production chain with the goal of reducing the costs of quality inspection assigned by humans or special machines [10, 6, 13]. However, only few studies have achieved and investigated the impact of early quality prediction in the very early stages of multi-staged processes or within the execution of the process [9, 5].

More recently, Deep Neural Networks (DNNs) have been successfully applied in the context of process quality prediction with high accuracy [9]. Their success is mainly due to their ability to learn new complex enriched features representations in an automated manner from the input data [14]. Thus, they achieve a good performance in solving a wide variety of complex tasks such as quality prediction where the expected process/product quality is affected by multi-interacting features in complex manufacturing settings [10, 6]. DNNs are therefore known to be complex black-box models that mostly give better accuracy in their predictions, but with very low interpretability [8]. However, likewise high prediction accuracy, both interpretability and explainability can be of the same importance and sometimes even more substantial for quality prediction related applications [7]. Hence, understanding the model's predictions, i.e. "Why" a certain quality label is predicted, and the model's dynamics, i.e. which model's parameters are taking into account or if the model contains any bias, can be of great help for subsequent decision-making by process experts on process optimization, such as adjusting process parameters, early stopping of the process if desired quality standards will not be reached, etc. The explainability of DNNs is an active field of research and several methods have been presented in the literature [8] including visualization-based approaches [11].

In this work, we employ a one dimensional Convolutional Neural Network (1D-CNN) to predict as early as possible the quality of a screwing process in a real-world automotive industry use case. The quality of a bolt is described by eight discrete labels indicating whether it is "defective" or "non-defective". In case of "defective", seven types of defects can be identified. The quality label is described by a time series feature. The learning task can formally be described as a multi-class Time Series Classification (TSC). We devised a training mechanism such that a 1D-CNN is trained on the training set composed of the full-length time series features. Then, a heat-mapping-based explanation method is used to highlight the most important discriminative pattern on the time series on a validation set. These maps are used not only for providing explanations, but also to determine where the discriminative patterns can be localized so that a reliable early quality prediction can be achieved by means of sampling, i.e. deriving shorter length of the time series window to be used for early prediction of the quality. The 1D-CNN is retrained using the input time series with the new length. We demonstrate how this training mechanism can be used to achieve better prediction accuracy using carefully selected time series subsequences on the described use case. In the remainder of this paper, we describe the proposed approach in Section 2. In Section 3, we describe the use case and the acquired data. Section 4 is dedicated to the experimentation part where the experimental set-up is described and the results are discussed. The last section concludes the paper.

2. Methodology

2.1. Notations and Definitions

Let the input data denoted as a multi-set $\mathcal{D} = (X, Y) = \{(x^{(i)}, y^{(i)})\}_{1 \leq i \leq N}$ which consists of $|\mathcal{D}| = N$ input data points $x^{(i)}$ and their corresponding label $y^{(i)}$. Each input data point $x^{(i)}$ consists of a univariate time series. A univariate time series is a sequence of data points, measured typically at successive points in time spaced at uniform time intervals. Each data point $x^{(i)}$ can be denoted by $x^{(i)} = \{x_1^{(i)}, x_2^{(i)}, \cdots, x_n^{(i)}\}$, where $x_t^{(i)} \in \mathbb{R}$ is the value of $x^{(i)}$ at the time instant t. A time series subsequence can be defined as a sequence of consecutive points which are extracted from the time series x starting from time instant t and can be denoted as $s_t^l(x) = \{x_t, x_{t+1}, \cdots, x_{t+l-1}\}$, where l is the length of the subsequence.

In general, the common goal of predictive ML models is to approximate some true function $f^* : \mathcal{X} \rightarrow \mathcal{Y}$, where $\mathcal{X}$ denotes the input feature space and $\mathcal{Y}$ is the target variable to be predicted. In our case, the predictive ML model is built to predict a discrete quality label given an input univariate time series. Therefore, $\mathcal{X} = \mathbb{R}^d$ is the input time series feature space, where the dimension d can be equal to n if the whole recorded time series is used for training the predictive model or to l if only time series subsequences of length l are alternatively used. $\mathcal{Y} = \{c_1, c_2, \cdots, c_k\}$ is the target variable that can take k possible values mutually exclusive. The learning task can be formalized as a multi-class TSC. The prediction of $\mathcal{Y}$ is carried out by the application of a model $f_\theta : \mathcal{X} \rightarrow \mathcal{Y}$. Usually, a fixed structure of f_θ is chosen that is parametrized by some vector $\theta \in \mathbb{R}^p$. Learning from data $X \in \mathcal{X}$ consists then of basically fitting θ to X, so that $f_\theta \approx f^*$.

Convolutional Neural Networks (CNNs) are a special class of predictive models. Their name stems from their structure that is built based on Deep Neural Network (DNNs) structure and the principle of convolutions. Deep Learning incorporates new representations of the raw data in $\mathcal{X}$ through transformations $\phi : \mathbb{R}^d \rightarrow \mathbb{R}^{d'}$ that map the raw features to another space $\phi(\mathcal{X}) = \mathbb{R}^{d'}$, into the learning process, i.e., $\phi = \phi_\theta$, fitted to some data. DNNs learn ϕ_θ in multiple levels of abstraction. Each level corresponds to one layer of the network. The overall DNN can be viewed as nested layer transformations $f(x) = f^{(l)}(f^{(l-1)}(\cdots f^{(1)}(x))$ where l is the depth of the network. Each layer $f^{(i)} : \mathbb{R}^{d^{(i-1)}} \rightarrow \mathbb{R}^{d^{(i)}}$ applies two operations. First, an affine transformation that consists of a weighted sum or convolution, is computed. Then, an activation function, e.g., tanh or ReLU

activation, is applied. Each layer is parametrized by the weights used in the affine transformation, while the activation function is usually fixed. These weights are learnt automatically using an optimization method that minimizes an objective loss function. In order to estimate the loss/error of some given weights values, a differentiable loss function is then defined. The most common used loss function in DNNs for classification task is the categorical cross-entropy loss [2]: $L(x) = -\sum_{j=1}^{k} y_j log(\hat{y_j})$ with $\hat{y_j}$ denotes the probability of x having the class y equal to class c_j out of k classes in the data set, noted y_j for simplicity and L the loss of classifying the input time series x. Similarly, we can define the average loss when classifying the whole training set of $\mathcal{D}$ by: $J(\omega) = \frac{1}{N}\sum_{i=1}^{N} L(x^{(i)})$ with ω is the set of weights to be learned by the network. The loss function is minimized to learn the weights in ω using a gradient descent method: $w = w - \alpha\frac{\partial L}{\partial w}|\forall w \in \omega$ with α is the learning rate of the optimization algorithm. By subtracting the partial derivative, the model is actually auto-tuning the parameters $w \in \omega$ in order to reach a local minimum of J. Since the partial derivative with respect to certain parameters $w \in \omega$ can not be in most of the cases computed directly, a chain rule of derivative is employed using the backpropagation algorithm [4].

A layer that computes a complete weighted sum in the affine transformation stage has to fit lots of weights because every neuron in the layer is connected there to every neuron in the previous layer. The major drawback of such fully connected layers is their tendency to overfitting. Convolution is used as an alternative that shares weights across all connections. The continuous operator defines the convolution of two functions $f, g : \mathbb{R}^d \rightarrow \mathbb{R}^{d'}$: $(f * g)(x) = \int f(x-a)g(a)\,da$.

2.2. 1D-CNN Architecture

In the case of univariate time series, a convolution can be viewed as applying and sliding a filter over the time series. Opposingly to the case of image data, the filters operate only on one dimension (time) instead of two dimensions (width and height). The filter can also be considered as a generic non-linear transformation of the raw time series. A general form of applying the convolution on the time series x at a centered time stamp t is given by: $C_t = f(K * s^l_{t-\frac{l}{2}}(x) + b), \forall t \in [1, n]$, with K is a filter of length l, $s^l_{t-\frac{l}{2}}(x)$ is the subsequence of the time series x centred at time t, b is a bias parameter and f is a non-linear function such as the Rectified Linear Unit (ReLU). Several filters are usually applied to the time series. An intuition behind applying several filters on an input time series would be to learn multiple discriminative features useful for the classification task [2]. In order to learn automatically the values of the filter K, the convolution should be followed by a discriminative classifier. It should be noted that some pooling operations such as average or maximum computation can also be applied after the convolution to reduce the length n of the time series by aggregating over a sliding window. Batch normalization operation can also be performed to accelerate the network convergence. In the context of time series, the batch normalization is performed over each channel, preventing thus the internal covariate shift across one mini-batch training of time series [15]. The final discriminative layer L takes the new representation of the input time series, i.e. resulting from the applied convolutions and give a probability distribution over the class variables in the data set using a softmax operation: $\hat{y_j}(x) = \frac{e^{f^{(L-1)}*w_j+b_j}}{\sum_{z=1}^{k} e^{f^{(L-1)}*w_z+b_z}}$, with $\hat{y_j}$ denoting the probability of x having the class y equal to class c_j out of the k classes and w_j the set of weights (and the corresponding bias b_j) for each class j connected to each previous activation in layer $L-1$.

The architecture of the CNN used in this work is inspired from [15]. The basic block is composed from a convolutional layer followed by a batch normalization layer and a ReLU activation layer. The convolution operation is fulfilled by 1-D kernels. The blocks are repeated four times with varying at each time the number of applied filters and their corresponding length. The basic convolution block exclude any pooling operation to prevent overfitting. Batch normalization is applied to speed up the convergence. After the convolution blocks, the features are fed into a Global Average Pooling (GAP) layer. However, opposingly to [15], instead of directly connecting the GAP layer to the final softmax layer, we feed its output to three dense layers. Even though this strategy would lead to increasing the number of weights, it results in better generalization [2].

2.3. Grad-CAM for Extracting explanations

Heat maps are one of the tools for providing visual explanations for CNNs, applied widely in computer vision related applications. Grad-CAM is one of the most popular methods for producing heat maps. Grad-CAM is a generalization of Class Activation Mapping (CAM), a method that does not require a particular architecture. The "Grad" in Grad-CAM stands for "gradient". Grad-CAM is being widely used since it has been proven to successfully pass commonly used sanity checks, which are devised to check whether the heat map is truly providing insights into what the model is doing or not. Grad-CAM can be transferred from the computer vision to time series domain [2]. Grad-CAM is applied to an already-trained neural network after training is completed and the parameters are fixed. We feed the time series input into the network to calculate the Grad-CAM heat-map for that input belonging to a given class of interest. The output of Grad-CAM is a "class-discriminative localization map", i.e. a heat-map where the hot part corresponds to a part in the input that is of the most importance for assigning a particular class to this input by the classifier. For example, in the case of images, this part corresponds to most important regions in the image for classifying an object within the image. In our case, it highlights the input time series subsequence that is of high relevance for deciding the class of the whole time series. The length and the beginning of the subsequence are then automatically decided by the Grad-CAM.

In order to obtain this map for any class $c_j, \forall j \in [1, k]$, the gradient of y^{c_j} before applying the softmax (i.e. the score of the class c_j) with respect to feature maps $A_f, \forall f \in [1, f_{maps}]$ of the last convolutional layer, is computed. These gradients flowing back are global average-pooled to obtain the neuron impor-

tance weights $\alpha_f^{c_j} = \frac{1}{Z}\sum_u \frac{\partial y^{c_j}}{\partial A_f^u}$, where Z is the total number of units u in A_f. Note that in the $2D$ case, the activation unit u has $2D$ coordinates $\{i, j\}$. In our case, it has only $1D$ dimension corresponding to the time dimension. This weight $\alpha_f^{c_j}$ represents a partial linearization of the deep network downstream from A_f, and depicts the importance of this feature map for the class c_j. Afterwards, a weighted combination of a forward activation maps is computed: $L_{Grad-CAM}^{c_j} = ReLU(\sum_{f=1}^{f_{maps}} \alpha_f^{c_j} A_f)$. The ReLU is applied to remove the negative contributions since we are mainly interested in the input part that have a positive influence on the class of interest c_j, i.e, in the case of images, pixels whose intensity should be increased in order to increase the distinguishability of c_j. Without this ReLU, localization maps sometimes highlight more than just the desired class and achieve lower localization performance [11]. $L_{Grad-CAM}^{c_j}$ is used to find the subsequence in the input time series that have mainly contributed to the decision of the network for the class c_j.

2.4. Important Time Series Subsequences Identification

The produced heat maps by the Grad-CAM can be used to identify the length and the temporal location of the most discriminative subsequences within the input time series. This would help also to check whether a reliable decision on performing early quality prediction can be made or not. In other words, these maps can be used to decide if the early subsequences in the input time series are on average the most important subsequences for correctly assigning the time series to its true class. To do so, we split $\mathcal{D}$ into two sets, $\mathcal{D}_{train}$ and $\mathcal{D}_{val}$. $\mathcal{D}_{train}$ is used to train the 1D-CNN. $\mathcal{D}_{val}$ where the class labels for each time series are assumed to be known, is used for producing the heat maps. Note that by means of random data shuffling, the data split and the 1D-CNN training are repeated until we ensure the condition C_1 corresponding to the two following points:

- All the classes $c_j, \forall j \in [1, k]$ are represented in $\mathcal{D}_{val}$.
- At least one time series sample from each class is correctly classified by the 1D-CNN.

Then, we select time series samples from $\mathcal{D}_{val}$ that are correctly classified by 1D-CNN, i.e. time series samples $x^{(s)} \in \mathcal{X}_{\mathcal{D}_{val}}$ that fulfill the condition C_2: $\hat{y}_{x^{(s)}}^{(s)} = y^{(s)}$. Denote with $X^{(s)}$ all the the series $x^{(s)} \in \mathcal{X}_{\mathcal{D}_{val}}$ fulfilling C_2.
For each selected time series $x^{(s)} \in X^{(s)}$ with $y^{(s)} = c_j$, we compute $L_{Grad-CAM}^{c_j}$ to highlight the most important subsequence $s_{t_i}^l(x^{(s)})$ of length l starting at time t_i, for assigning the correct class membership to $x^{(s)}$. Then, we compute the average length l_s of all the computed subsequences starting from $t_i = t_0$, where t_0 is the initial instant of the process time series generation. Finally, we compare l_s to the average length l_m of the original input time series in $\mathcal{X}_{\mathcal{D}}$: $l_m = \frac{1}{N}\sum_{x \in \mathcal{X}_{\mathcal{D}}} length(x)$. We verify afterwards if the condition C_3 is fulfilled: $l_s < l_m and l_m - l_s \geq \tau$ where τ is the admissible time duration required to perform process optimization, e.g. corrective measures, process parameters adjustment. τ is a user-defined hyperparameter as it is application-dependent and an interaction with domain experts is required to set up its value. If the condition C_3 is fulfilled, it is possible then to reliably perform early quality prediction before the termination of the process. The 1D-CNN is then retrained on the input time series subsequences of $\mathcal{D}_{train}$ of length l_s. This operation can be viewed as an input feature selection where only l_s input time series points are fed to the 1D-CNN. The above steps are summarized in Algorithm 1.

Parameters: Data set: $\mathcal{D}$; Admissible time duration for process adaption: τ.
1: Repeat $\mathcal{D}$ split into $\mathcal{D}_{train}$ and $\mathcal{D}_{val}$ and the 1D-CNN training until C_1 is fulfilled.
2: Select time series samples from $\mathcal{D}_{val}$ that fulfill C_2 and put them in $X^{(s)}$.
3: **for** $x^{(s)} \in X^{(s)}$ **do**
4: We compute $L_{Grad-CAM}^{y(x^{(s)})}$ to highlight $s_{t_i}^l(x^{(s)})$.
5: Set up all the t_i to t_0.
6: Calculate the lengths of the subsequences starting from t_0 including the computed $s_{t_i}^l(x^{(s)})$.
7: **end for**
8: Compute the average length l_s over the computed above lengths.
9: Verify the validity of C_3: $l_s < l_m$ and $l_m - l_s \leq \tau$
10: If C_3 is fulfilled, the 1D-CNN is retrained on the input time series subsequences of $\mathcal{D}_{train}$ of length l_s.

Algorithm 1: Important **T**ime **S**eries **S**ubsequences **I**dentification: **ITSSI**

3. Case Study

Bolted connections are commonly used mechanical connections in industrial products. The non-linear tightening process of the bolts is generally distributed into four different zones that can be visualized using the torque diagram, which is a commonly used monitoring tool for the quality of bolted connections in industry. The four zones are the rundown, alignment, elastic clamping and post yield zone. By strengthening and plastic deformation of the bolts, a clamping force is generated between the connected parts. However, under realistic process conditions many different complex factors influence the quality of the part connections: e.g. connected parts might move during the screwing, the equipment condition might be deficient, or the quality of the bolts might differ leading to quality problems in the final product. When the occurrence and cause of the fault in the bolt connection is detected at an early stage in the process, countermeasures can be taken by the worker so that potential quality problems can ideally be avoided. However, monitoring methods are mainly based on the comparison of the torque with a preset standard value, ignoring the analysis of the torque diagram. As a result, however, various errors remain hidden. For this reason, the use of more sophisticated methods such as data mining is desirable to achieve an improvement of the analysis quality within the bolting process. A historical data set covering 233138 industrial screwing cases representing torque sequences from the same workplace and product type is used as a

case study. The data set consists of eight different classes, with one class representing the "non-defective" cases while the remaining depict seven different types of defects. Only 358 cases are labelled as "defective" which induces a highly class imbalanced multi-class classification problem. The sensory data is described by a univariate time series feature of the normalized torque that is measured during the screwing process.

4. Experiments

4.1. Experimental set-up

The available data set is split in 75% for training and 25% for testing. The 75% are split between 75% for $\mathcal{D}_{train}$ and 25% for $\mathcal{D}_{val}$. 10-folds cross validation procedure is employed for the evaluation of our method. The reported results are the averaged metrics values over all the folds. Since the data set reveals high imbalance ratios towards some classes, both random oversampling and under-sampling are employed to mitigate this issue [1]. More precisely, we under-sampled the "non-defective" class and over-sampled the seven different defect classes. In addition, the time series of the data have different lengths, zero-padding is employed to bring the input time series to the same length. For the 1D-CNN, all convolutions have a stride equal to 1 with a zero padding to preserve the exact length of the time series after the convolution. The first convolution contains 128 filters with a filter length equal to 10, followed by a second convolution of 128 filters with a filter length equal to 8, then a third convolution of 256 filters with a filter length equal to 5 which in turn is fed to a fourth and final convolutional layer composed of 128 filters, each one with a length equal to 3. The three dense layers are composed of 500, 300, and 100 neurons, respectively, each one using ReLU activation function. The number of neurons in the final softmax classifier is equal to 8, i.e. the number of classes in the data. The model's weights are learnt using a variant of Stochastic Gradient Descent (SGD), namely Adam. The number of training epochs is set to 2000.

4.2. Evaluation Metrics

To evaluate the achieved results, the confusion matrix is utilized. This matrix represents all prediction results of a given model for multi-class as follows: The *Recall* metric calculates

		Predicted Value		
		Class A	Class B	Class C
Actual Value	Class A	Aa	Ab	Ac
	Class B	Ba	Bb	Bc
	Class C	Ca	Cb	Cc

Table 1: Confusion Matrix

the proportion of actual positives that were correctly identified. For example, for class A:

$$Recall(A) = \frac{TruePredictedA}{TotalTrueA} = \frac{Aa}{Aa+Ab+Ac}$$

The *Precision* describes what proportion of positive identifications are actually correct. The *F1-score* represents the trade-off between Precision and Recall. They are calculated as follows for class A:

$$Precision(A) = \frac{TruePredictedA}{TotalPredictedAsA} = \frac{Aa}{Aa+Ba+Ca}$$

$$F1 - score = 2 \times \frac{Precision*Recall}{Precision+Recall}$$

The metrics are calculated per class. There are two ways to compute their average: *Macro Average* (*M.Avg*), which is a simple arithmetic mean of the metrics per class, and *Weighted Average* (*W.Avg*), which is a weighted mean based on the number of samples per class. For example:

$$M.AvgRecall = \frac{\sum RecallPerClass}{TotalNumberOfClasses} = \frac{Recall(A)+Recall(B)+Recall(C)}{3}$$

$$W.AvgRecall = \frac{\sum(RecallPerClass*NumberOfSamplesPerClass)}{TotalNumberOfSamples}$$

The *Micro Average* (*Mi.Avg*) is a metric that is used to calculate the metric for the whole model. For example:

$$Mi.AvgRecall = \frac{\sum TruePredictedPerClass}{\sum TotalTruePerClass}$$

Following the same idea, the macro, weighted and micro averages definitions can be extended to Precision and F1-Score. However, in the case of micro average, it should be noted that based on the definition, the following equation is valid:

$$Accuracy = Mi.AvgRecall = Mi.AvgPrecision = Mi.Avg\text{F1-Score} = \frac{TotalTruePrediction}{TotalNumberOfSamples}$$

4.3. Results and Discussion

The results are presented over two different perspectives: (1) Table 2 encloses descriptive statistics of the results on the prediction performance of the 1D-CNNs trained on the full length time series data and on the identified subsequences by Algorithm 1, denoted CNN_{full} and CNN_{ITSSI}, respectively. The length of the input time series for the CNN_{full} is 727 time steps. However, the average length l_s derived following the procedure explained in Algorithm 1 is 250 time steps. So, CNN_{ITSSI} is trained on the input subsequence of length l_s; (2) Figure 1 shows the localization of some examples of the identified subsequences for different classes and the differences between them.

Metric	Precision		Recall		F1-score	
	CNN_{full}	CNN_{ITSSI}	CNN_{full}	CNN_{ITSSI}	CNN_{full}	CNN_{ITSSI}
M.Avg	58%	**66%**	77%	77%	63%	**68%**
W.Avg	**99%**	98%	94%	**97%**	96%	**97%**
Mi.Avg	–	–	–	–	94%	**97%**

Table 2: Average Performance Comparison of CNN_{ITSSI} Vs. CNN_{full}

Following Table 2, it can be seen that feeding the most important, i.e, discriminative time series patterns to the 1D-CNN improves its predictive performance. Since the data is highly imbalanced, the micro and the weighted average measures are biased towards the majority classes that are more accurately predicted by the classifier and much more represented in the data set. Better overview of the performance on the minority classes can be seen using the macro measures that reflect the averaged measures independently from their representation in the

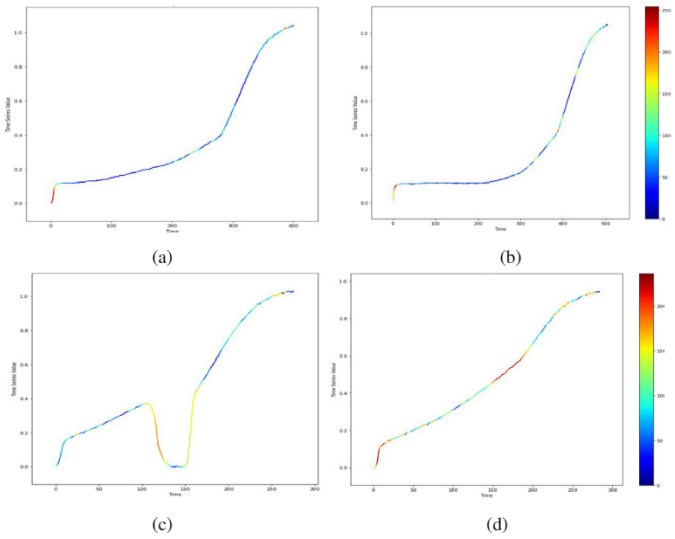

Fig. 1: Examples of heat-maps produced by the Grad-CAM: (a-b) Two examples of input time series for the "non-defective" class. (c-d) Two examples belonging to two different "defect" classes. The x-axis is the time, while the y-axis is the recorded torque value as the time series value. Red color is used for highlighting the most important subsequences and the blue for less important parts.

data. A clear improvement in the Precision is achieved by the CNN_{ITSSI}, while preserving similar Recall. This results in an increase of 8% in the F1-score compared to the CNN_{full}. With respect to the CNN_{ITSSI}, the prediction accuracy is improved using shorter time series inputs (i.e. lower dimensionality). The dimension is reduced up to 65% from 727 time points to 250. Therefore, our method can also be viewed as input feature selection (i.e. particular time series points selection), which also helps in reducing the general resources consumption required for model training.

From practice point of view, relying only on the first 250 time steps to make the decision within the execution of the process enables to save resources of further processing of products that are not expected to reach the standard quality requirements (i.e. containing some types of defects). In addition, if C_3 is fulfilled with $l_m = 500$ and $l_s = 250$, process optimization and intervention can happen to adjust the process so that expected quality deviations can be corrected.

Figure 1 shows some examples of the produced heat-maps by the Grad-CAM for different classes. The most important subsequences are highlighted in red, while the least important are present in dark blue. It can be seen that the "non-defective" class presents some regularity in the patterns. The most important subsequences are almost localized in the beginning of the process. Clear different patterns can be distinguished in the highlighted subsequences for the "defective" classes. Their localization in average confirms also the validity of the derived length $l_s = 250$ by the **ITSSI** Algorithm 1. The most important subsequence localization can be further optimized by tuning also t_i instead of setting it to t_0 and by ensuring equal length sequences to be fed to the 1D-CNN using zero padding.

5. Concluding Remarks

In this paper, a novel approach for important time series subsequences identification for time series classification is presented. This is achieved using a CNN network and a gradient based heat-mapping approach, namely Grad-CAM. The method demonstrates a satisfactory performance not only in improving the classification accuracy, but also in achieving early quality prediction in a real-world use case in the automotive industry.

Acknowledgements

This work is supported by Deutsche Forschungsgemeinschaft (DFG) within the Collaborative Research Center SFB 876 "Providing Information by Resource-Constrained Analysis", and the Federal Ministry of Education and Research of Germany as part of the competence center for machine learning ML2R (01—S18038A).

References

[1] Chawla, N.V., 2009. Data mining for imbalanced datasets: An overview. Data mining and knowledge discovery handbook , 875–886.
[2] Fawaz, H.I., Forestier, G., Weber, J., Idoumghar, L., Muller, P.A., 2019. Deep learning for time series classification: a review. Data Mining and Knowledge Discovery 33, 917–963.
[3] Finkeldey, F., Saadallah, A., Wiederkehr, P., Morik, K., 2020. Real-time prediction of process forces in milling operations using synchronized data fusion of simulation and sensor data. Engineering Applications of Artificial Intelligence 94, 103753.
[4] LeCun, Y., Bottou, L., Bengio, Y., Haffner, P., 1998. Gradient-based learning applied to document recognition. Proceedings of the IEEE 86, 2278–2324.
[5] Lieber, D., Konrad, B., Deuse, J., Stolpe, M., Morik, K., 2012. Sustainable interlinked manufacturing processes through real-time quality prediction, in: Leveraging Technology for a Sustainable World. Springer, pp. 393–398.
[6] Lieber, D., Stolpe, M., Konrad, B., Deuse, J., Morik, K., 2013. Quality prediction in interlinked manufacturing processes based on supervised & unsupervised machine learning. Procedia Cirp 7, 193–198.
[7] Lv, Z., Han, Y., Singh, A.K., Manogaran, G., Lv, H., 2020. Trustworthiness in industrial iot systems based on artificial intelligence. IEEE Transactions on Industrial Informatics 17, 1496–1504.
[8] Molnar, C., 2020. Interpretable machine learning. Lulu. com.
[9] Saadallah, A., Finkeldey, F., Morik, K., Wiederkehr, P., 2018. Stability prediction in milling processes using a simulation-based machine learning approach. Procedia CIRP 72, 1493–1498.
[10] Schmitt, J., Bönig, J., Borggräfe, T., Beitinger, G., Deuse, J., 2020. Predictive model-based quality inspection using machine learning and edge cloud computing. Advanced engineering informatics 45, 101101.
[11] Selvaraju, R.R., Cogswell, M., Das, A., Vedantam, R., Parikh, D., Batra, D., 2017. Grad-cam: Visual explanations from deep networks via gradient-based localization, in: Proceedings of the IEEE international conference on computer vision, pp. 618–626.
[12] Stolpe, M., 2016. The Internet of Things: Opportunities and challenges for distributed data analysis. SIGKDD Explorations 18, 15–34.
[13] Stolpe, M., Blom, H., Morik, K., 2016. Sustainable industrial processes by embedded real-time quality prediction, in: Computational sustainability. Springer, pp. 201–243.
[14] Utgoff, P.E., Stracuzzi, D.J., 2002. Many-layered learning. Neural computation 14, 2497–2529.
[15] Wang, Z., Yan, W., Oates, T., 2017. Time series classification from scratch with deep neural networks: A strong baseline, in: 2017 International joint conference on neural networks (IJCNN), IEEE. pp. 1578–1585.