Explainable Prediction of Manufacturing Anomalies via Convolutional Models and Layer-wise Relevance Propagation

by

Zhijian Ling

A thesis submitted in conformity with the requirements
for the degree of Master of Applied Science

Department of Mechanical & Industrial Engineering
University of Toronto

Explainable Prediction of Manufacturing Anomalies via Convolutional Models and Layer-wise Relevance Propagation

Zhijian Ling
Master of Applied Science
Department of Mechanical & Industrial Engineering
University of Toronto
2024

Abstract

Anomaly prediction involves estimating defects in manufacturing products based on the conditions of the processing line, as recorded by sensors. This task is crucial as it can forecast future anomalies, thereby improving product quality and reducing investment costs. Explainable prediction provides intelligible explanations for the model's predictions, enabling manufacturing managers to understand the relationship between sensor information and the resulting anomalies, allowing them to mitigate their occurrence. In this project, we focused on predicting periods of high anomaly counts in a future horizon of 72 hours based on sensor data in the past 24 hours. Due to the success of convolutional kernel-based methods in capturing temporal relationships, we investigated two types of convolutional approaches: ROCKET-EBM, which utilizes a large number of random convolutions, and InceptionTime, which relies on trained convolutions. However, as these models were not inherently interpretable, we utilized a post-hoc explanation approach called Layer-wise Relevance Propagation (LRP). For ROCKET-EBM, we designed a LRP-based algorithm, while for InceptionTime, LRP was directly applied. In our experiments using real manufacturing data, both models outperformed our baseline models. Further, we evaluated the quality of explanations produced by LRP for both models. For the ROCKET-based model, we observe that LRP is able to identify the crucial features for the model. Due to the complexity of the InceptionTime model, we applied relevance score filtering to eliminate small intermediate relevance scores. We observed that the features identified through this process were sufficient to reproduce the model's performance.

Acknowledgements

First, I would like to sincerely thank my supervisor, Professor Eldan Cohen, for his constant guidance and support throughout my research. He allowed me the freedom to explore my interests while providing the direction needed. His mentorship has been invaluable, and I have gained a tremendous amount of knowledge and experience over the years under his guidance.

I also want to express my gratitude to my lab mates and the company I've worked with. Their collaboration and shared insights have greatly enriched my research and made the journey both productive and enjoyable.

Lastly, I want to thank my family and friends for their unwavering emotional support and love. Their encouragement has been essential in helping me stay focused and motivated throughout this journey.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Manufacturing plays a crucial role in the economic system and is closely connected to our daily lives. It involves transforming raw materials into products through a series of complex steps. Each step in the manufacturing process is highly interdependent and must meet stringent requirements; otherwise, the final product may be compromised if any step falls outside these constraints [36][58].

This project aimed to forecast the occurrence of scratches and foreign matter, referred to as anomalies, on products during the inspection process. In our manufacturing process 1.1, raw materials undergo a series of procedures along the processing line, which is composed of multiple stations. Each station performs specific tasks such as shaping, cutting, or treating the materials. After processing, the products are inspected for any anomalies. Sensors throughout the processing line record various parameters, such as temperature, pressure, speed, and weight. Some of these parameters are strongly correlated with the quality of the final products and are significant factors in identifying the causes of anomalies. Due to the continuous nature of the raw materials, they are not processed as discrete individual units, meaning there is no unique ID that can be tracked throughout the process. This introduces additional challenges in accurately detecting and linking anomalies to specific sections of the production line.

Figure 1.1: Manufacturing Pipeline

1.1 Anomaly Prediction and Recent Development

The number of scratches and foreign matter is highly correlated with the conditions on the manufacturing line. Anomaly prediction involves establishing a relationship between the Sensor Dataframe and the Anomaly Dataframe to identify and address potential issues.

Anomaly prediction aims to identify outliers or the frequency of outliers within the manufacturing dataset. In manufacturing, an "anomaly" refers to deviations in product quality that stray from expected norms. This technique is extensively used across various sectors, including the aerospace [77][56] and pharmaceutical industries [45][87]. With the rapid expansion of data, it becomes impractical to scrutinize and identify anomalies manually.

Factories must invest significant effort and resources to optimize these processes and enhance product quality. Due to the numerous steps involved, detecting when a particular step falls outside the required range can be challenging. Recently, machine learning and deep learning have gained strong momentum in manufacturing. These techniques improve quality by building complex relationships with data. By leveraging machine learning and deep learning, manufacturers can significantly reduce investment in testing while efficiently enhancing product quality.

Anomaly prediction has garnered significant attention from researchers, leading to the development of numerous algorithms. Recent advancements in machine learning and deep learning have further enhanced the effectiveness of these algorithms in anomaly prediction applications. Based on the type of data, modeling techniques can be classified into supervised [85][62], semi-supervised [63][57], and unsupervised [27] approaches. These algorithms achieve high accuracy in identifying anomalies but suffer from a significant lack of interpretability. Most of the models used are black-box models. In manufacturing, it is crucial for managers

who are not experts in machine learning and deep learning to understand and trust the models. Therefore, interpretability is essential in this context.

1.2 Problem Definition

This project aimed to forecast the future number of anomalies and identify the production processes closely related to the increasing counts. To achieve this, we built models that predicted future anomaly counts using historical sensor data. Additionally, we employed a post-hoc technique to explain the model and identify the features most closely related to high anomaly counts.

The goal of this project is to develop an accurate and explainable long-term manufacturing anomalies prediction model. The model aims to establish the relationship between sensor data collected at each station along the processing line and the anomalies detected during the final inspection without the need for tracking individual items to link their sensor data to specific anomalies. To ensure interpretability, techniques will be applied to explain the model's predictions by identifying the most relevant features contributing to each anomaly forecast.

Given the strong performance of convolutional models in time series data for capturing temporal relationships, we focused on convolutional models for our task. In this thesis, we investigated the performance of both random convolutional models and trained convolutional models. Random convolution involves randomly generating the kernel, padding, and dilation parameters. Typically, a large number of random convolutions are required to ensure that some of them effectively capture the patterns within the data. Subsequently, a model is used to refine the information generated by these random convolutions. In contrast, trained convolutional models are optimized through gradient backpropagation with fixed padding and dilation parameters. Due to computational limitations, the number of trained convolutions is usually small. We utilized Layer-wise Relevance Propagation (LRP), a widely used post-hoc explanation technique, to explain the model's predictions. LRP is efficient and easy to implement.

In this thesis, we make the following contributions:

1. We constructed a random convolutional model inspired by ROCKET for predicting manufacturing anomalies. The model employs an Explainable Boosting Machine (EBM) as the head, referred to as ROCKET-EBM. Additionally, we designed an LRP-based algorithm to explain the predictions of ROCKET-EBM. ROCKET-EBM outperformed baseline models, and the top features identified by the LRP-based algorithm were confirmed to be critical to the model's predictions.

2. We utilized the state-of-the-art trained convolutional model, InceptionTime, for this task and employed LRP to explain the model's predictions. We showed that InceptionTime outperformed traditional neural networks. We identified limitations in the use of LRP in this setting and integrated relevance score filtering into the LRP algorithm, resulting in improved explanations.

1.3 Outline

The organization of the thesis is as follows:

Chapter 2 provides a brief background on time series analysis and modeling, along with an explanation of techniques of machine learning and deep learning.

Chapter 3 outlines the manufacturing setting, the selection of manufacturing parameters, and data preprocessing.

Chapter 4 presents an approach based on random convolution kernels, a non-neural network model, to predict anomalies and presents an LRP-based explainability mechanism for the model.

Chapter 5 focuses on trained convolutional neural networks to predict future anomalies combined with LRP-based explanations for the model's predictions.

Chapter 6 summarizes the findings and outlines future work.

Chapter 2

Background

2.1 Time Series Analysis

A time series is a data collection tracking a sample's behavior over a given period, defined as a sequence of data points indexed in time order, $\{(x_1, t_1), (x_2, t_2), \cdots, (x_n, t_n)\}$, where x_t is the measurement taken at time t. Time series data are prevalent in various fields, including finance [49], manufacturing [72], and climatology [25]. Time series research encompasses a wide array of topics, including time series forecasting. The aim is to predict future values by leveraging historical data and exogenous variables. These exogenous variables are external factors that impact the time series, such as seasonal variation, holidays, and economic conditions. Another critical area of time series research is anomaly detection [15], which involves identifying unusual patterns or outliers within the data to explore their causes or implications. The complexity of time series arises from its combination of significant information and noise, making analysis challenging. This has led to extensive research efforts to extract meaningful insights from such data. Time series analysis is generally divided into two categories: univariate, which examines a single data sequence, and multivariate, which deals with multiple interdependent sequences.

2.1.1 Univariate Time Series

The characteristics of univariate time series analysis include trends, seasonality, and outliers [50]. Trends reflect long-term changes in the level of a time series, and seasonality shows repeating cycles with a fixed frequency. Moreover, trends and seasonality are related to stationarity. A stationary time series preserves the statistical properties, such as mean, variance, and autocorrelation, over time. However, a time series with trends and seasonality changes values depending on the time, which is a non-stationary time series. A time series can be decomposed into trends, seasonality, and residual components. The seasonal decompose technique [24] applies the moving average to transform a time series in the form of an additive model, Time Series = Trend + Seasonality + Remainder, or multiplication model, Time series = Trend × Seasonality × Remainder.

Outliers are points that are significantly different from the pattern. Techniques such as K-Means [34] and Box plots [21] are employed to identify outliers while examining the remainder components from the seasonal decomposition provides an alternative method. In addition, outliers in a time series may not only be individual anomalies but also can be a sequence of data points deviating from the patterns [52]. For example, if a time series segment does not exhibit seasonality observed in the series, this segment can be identified as a sequence of anomalies.

A shapelet, proposed by Ye and Keogh [94], is a short pattern that can be identified as a representative of a class in time series classification. Shapelets are essential in time series analysis, as they help capture the similarity between different time series. Several algorithms have been developed to detect the shapelets, such as Fast shapelets [70] and W-TSS [53], which utilizes a wavelet-based method.

2.1.2 Multivariate Time Series

In the analysis of multivariate time series, it is essential to consider not only the properties within individual time series but also the interrelationships among them. When the time series share the same time indexes, the similarity between two time series can be measured by Euclidean Distance. However, challenges arise when comparing time series with different lengths. Dynamic Time Wrapping(DTW) [66] offers an approach that aims to find the optimal match of two time series to tackle this problem. The DTW algorithm computes the minimal distance needed to align each point in the time series to achieve the best match. A time series can contain various pieces of information, some of which may not be connected with those from other time series.

2.2 Time series Forecasting

Time series forecasting involves predicting a future value or a sequence of values using historical data. This is inherently a sequential prediction problem. Time series forecasting is a critical tool across various domains, including meteorology [2][47], finance [64][14], and healthcare [19][69], where accurate prediction can optimize resource allocation.

Time series forecasting has evolved from statistical methods, such as smoothing-based techniques and ARIMA [10], to machine learning approaches like linear regression. Recently, a breakthrough came with the advent of deep learning, dramatically improving the accuracy of time series forecasting [88]. In particular, the transformer model, achieving remarkably high performance in language, has proved exceptionally effective in time series forecasting [26]. This has led to the development of various transformer-based models for time series forecasting. The following discussion is structured into three main sections: the first covers statistical techniques, the second shows RNN-based models, and the third highlights transformer-based models.

2.2.1 Statistical Techniques

Statistical techniques for time series forecasting, though traditional, remain effective. With less computational demand, their performance is still satisfactory. The widely used smoothing-based technique for time series forecasting is the moving average [35]. This method involves a sliding window that calculates predictions based on the preceding values. It can potentially capture the general pattern of the data. However, the moving average approach can hardly deal with sophisticated time series. Its simplicity can lead to the omission of significant details like seasonality. Meanwhile, outliers can primarily affect the predictions using the moving average technique.

ARIMA [10] was developed to address these shortcomings. ARIMA, which stands for AutoRegressive Integrated Moving Average, has advanced significantly. Despite its relative simplicity compared to machine learning models, ARIMA achieves remarkable improvements. ARIMA is a model that combines AutoRegressive(AR) and Moving Average(MA) along with a differencing step(integrated). The AR models the relationship between the observation and lag values. Then, the MA component forecasts using the residual errors. However, standard ARIMA models may struggle with seasonal data. In such cases, Seasonal ARIMA(SARIMA) [10] is more appropriate, as it considers the seasonal terms. However, it is important to note that ARIMA models have limitations in handling 'black swan' events like the COVID-19 pandemic.

2.2.2 RNN-based Models

The statistical techniques mentioned above, such as ARIMA, are designed for univariate time series forecasting. These techniques cannot capture the relationships between multiple time series. In multivariate time series forecasting, the objective of models is to simultaneously predict the future values for each time series based on the historical data of each individual series and the interdependencies among these series. Suppose we have N related time series, each observed at discrete time points $t = 1, 2, \ldots, T$. These time series can be represented as $\{\mathbf{y_1}, \mathbf{y_2}, \ldots, \mathbf{y_N}\}$, where $\mathbf{y_i} \in \mathbb{R}^T$. The aim of multivariate time series forecasting is to predict

$y_{i,t}$, which denotes the value of the i^{th} time series at time t. Mathematically, this problem can be formulated as:

$$\hat{y}_{i,t+1} = f(y_{i,t-k:t}, \mathbf{x}_{i,t-k:t})$$

Here, $\hat{y}_{i,t+1}$ is the predicted value for the i^{th} time series at time $t+1$. The function f represents the forecasting model that maps the historical time series from time $t-k$ to time t, $y_{i,t-k:t}$, and exogenous inputs, $\mathbf{x}_{i,t-k:t}$.

RNNs [74] have gained widespread popularity in time series forecasting due to their inherent 'memory' capabilities, which enable them to connect the current data points with previous ones. Nevertheless, RNNs encounter challenges in learning long-term dependencies. Hochreiter and Schmidhuber [41] introduced Long Short Term Memory networks(LSTM), which are designed to handle this problem. LSTM utilizes an architecture that can selectively add and remove some information.

However, experiments show that RNNs, including LSTM, struggle to capture long-term dependencies and face issues with vanishing gradients in multivariate time series(MTS) forecasting [71]. The attention mechanism, initially used in the encoder-decoder[43], offers a promising approach to address these problems. The typical attention-based RNNs for MTS forecasting assign varying weights to the hidden states of related previous timestamps. However, experiments show that the typical attention-based methods may face challenges in identifying related timestamps as they distribute almost equal attention to each [59]. Li et al. [54] suggested using a competitive random search instead of traditional gradient-based methods to learn the attention weights. In contrast to these typical attention-based RNNs, Shih et al. [78] introduce a novel attention mechanism for MTS forecasting. Instead of focusing on the hidden states of previous timestamps, they apply Convolutional Neural Networks(CNNs) to them, which can capture temporal patterns within a variable. The paper employs several convolution filters to figure out several possible temporal relationships. Subsequently, the attention mechanism determines which variable is responsible for the prediction. Instead of using the traditional softmax function, they opt for the sigmoid function to allocate attention to more possible variables.

2.2.3 Transformer-based Models

The Transformer [90], a potent model widely employed for sequence-to-sequence prediction tasks, particularly in natural language processing, consists of both an encoder and a decoder. The central components that underpin the Transformer's capabilities are the attention mechanisms, which include self-attention and cross-attention. Self-attention involves performing multiplicative operations between queries and keys, both resulting from the linear combination of the input embeddings. The rationale behind this is to assess the similarity between elements in the sequence. The softmax function is applied to check the relative importance of these elements to others. The output is then computed as a combination of the softmax and the value, which is also the linear combination of the input embeddings. The intuition is to establish relationships within the sequences. Like self-attention, cross-attention represents the output embedding as a linear combination of the encoder outputs.

As the Transformer has demonstrated its remarkable abilities in natural language processing, several adapted versions of this model have been employed for Long Sequence Time-Series Forecasting(LSTF). As mentioned earlier in the RNN-based models, the attention mechanism in the Transformer can encounter difficulties in capturing long-range relationships between timestamps in LSTF. Models like Informer [97], Autoformer [93], and FEDformer [98] have been proposed to address these challenges and improve forecasting accuracy. Multiplication in traditional self-attention mechanisms is computationally expensive. To address this problem, Informer presents the ProbSparse self-attention mechanism, wherein each key only attends to u dominant queries. The u dominant queries are determined by a measurement function. Additionally, convolution and max-pooling techniques are employed to refine the model further after self-attention blocks, enhancing the model's ability to capture the temporal dependencies and reducing data dimensionality. The decoder architecture in Informer is similar to the vanilla transformer but incorporates an autoregressive mechanism. Hence, the next prediction is independent of the preceding outputs, minimizing error accumulation. Autoformer is tailored for time series forecasting, emphasizing the temporal characteristics of time series. It employs a seasonal-trend decomposition technique, applying a moving average kernel to the

input sequence. This method can isolate the trend-cyclical components. Moreover, Autoformer replaces the self-attention mechanism with the auto-correlation mechanism, which can discover the period-based dependencies by computing the series autocorrelation. FEDformer, built on top of Autoformer's decomposition scheme, extracts trend-cyclic components using various kernel sizes. Fedformer integrates Fourier-enhanced and wavelet-enhanced blocks in the self-attention mechanism to optimize computational efficiency.

Most time series data are non-stationary, and while standardization methods are commonly used before inputting data into the model, it has been observed that stationarized data may reduce the efficiency of self-attention. Liu et al. [59] proposed the Non-stationary Transformer, which redefines the attention mechanism to account for non-stationarity in data specifically. The transformer is popular in LSTF, but Zeng et al. [96] questioned the efficiency of Transformers in this field. They proposed two non-transformer models, DLinear and NLinear. DLinear incorporates the decomposition scheme from Autoformer. NLinear normalizes the input data by subtracting the input sequence from its last value. Experiments indicated that both DLinear and NLinear outperformed transformer-based models in various datasets.

2.3 Time Series Extrinsic Classification/Regression

Unlike Time Series Forecasting, which predicts continuous sequences, Time Series Extrinsic Regression(TSER) focuses on generating informative outputs from a time series, providing overall data descriptions. Given the similarities between classification and regression tasks, many models designed for Time Series Extrinsic Classification (TSEC) can also be applied to regression. Tan et al. [84] explore this by adapting various TSEC models for TSER. This investigation yielded promising results, with some models demonstrating notable effectiveness. Conceptually, TSEC can be viewed as a process of mapping shapelets, distinct time series subsequences, to some specific values. These models identify and leverage patterns within time series data to generate comprehensive predictions.

TSEC models can be categorized into two categories: those utilizing convolution and those without. Models without convolution typically fall under traditional machine learning, such as Random Forest [12], XGBoost [18], and various shapelet extraction methods used in time series analysis [55][94]. Experiments suggest that incorporating convolution can lead to significant improvements in performance. Apart from deep learning models that employ convolution, one notable model, ROCKET [22], achieves impressive accuracies using a large number of untrained random convolution kernels. The architecture of ROCKET will be discussed in more detail later chapter 4. Deep learning models like InceptionTime [44], Fully Convolution Networks(FCNs), and ResNet demonstrate superior performance in TSEC tasks [92]. IncpetionTime will be examined in detail in subsequent sections chapter 5.

FCNs utilize convolutions over convolutions. These convolution kernels vary in size, allowing the networks to capture temporal patterns from receptive files of different sizes. ResNet, proposed by He et al. [39], enhances the FCNs by integrating residual blocks. These blocks feed the input data back into the output after convolution, which can help preserve information. Additionally, residual blocks effectively address the issue of gradient vanishing.

2.4 Explainable Machine Learning

Machine learning and deep learning are capable of making accurate predictions but often lack explanation. In particular, the complexity of deep learning models tends to make their decision-making process opaque. Yet, in some applications such as healthcare [3][17], risk management[13], and disaster forecasting [31], it is significant to provide trustworthy and comprehensive explanations. Interpretability can be approached in two ways to address this need: intrinsically interpretable models and post-hoc explanations.

2.4.1 Intrinsically Interpretable Models

Linear regression is a widely used interpretable model that predicts outcomes based on the sum of the product of weight and features plus a bias term. The model elucidates the importance of each feature through their corresponding weights. Lasso regression [86], an extension of linear regression, is introduced to increase the sparsity of linear regression by penalizing the magnitude of weights. Logistic regression is a commonly used model for classification tasks that share similarities with linear regression. The importance of the feature in logistic regression can be examined based on weights—however, linear regression and logistic regression struggle to capture non-linear relationships.

The decision tree [11] provides an effective method to tackle this problem. A decision tree begins at a decision node, splits based on a predefined threshold at branch nodes, and ends with leaf nodes representing predictions. Pruning is commonly used to prevent overfitting. Decision trees can present the cutoff thresholds at each decision node, offering clear interpretability by tracing the decision path from the root to the leaf, showing how feature splits lead to a prediction. Historically, decision trees have been computed using heuristics, but recent approaches based on discrete optimization have emerged to compute optimal decision trees globally [9][76].

Typically, there is a trade-off between interpretability and performance. Generalized Additive Model(GAM) [37] can potentially improve the performance by capturing non-linear relationships while retaining interpretability. Mathematically, GAM is expressed as:

$$g(\mathbb{E}(Y)) = \beta_0 + f_1(x_1) + f_2(x_2) + \cdots + f_n(x_n)$$

Here, β_0 is the bias term, and f_i is a spline function. Inspired by GAM, Nori et al. [67] introduced Explainable Boosting Machine(EBM). This model will be discussed in the following chapter. Neural Additive Model(NAM) [1] has been proposed to utilize complex architecture but still maintain interpretability. As aforementioned, GAM employs spline functions to model f_i, whereas, in NAM, the function f_i is constructed by a small neural network.

Decision trees and GAMs can also enhance interpretability in clustering problems. Cohen [20] introduced a soft decision tree approach that supports spectral and kernel-PCA clustering objectives, providing a more interpretable framework for clustering tasks. Additionally, Upadhya and Cohen proposed NeurCAM [89], which leverages GAMs to offer additive explanations for improved interpretability in clustering models.

2.4.2 Post-Hoc Explanation

Post-hoc explanations can typically be categorized into model-agnostic and model-specific. Model-agnostic methods separate the predictive model from the explanation. They provide an explanation only based on inputs and outputs. Conversely, model-specific explanations leverage the architecture of the predictive model itself and are often utilized in neural networks.

Model-agnostic Explanation Methods

Partial Dependence Plots(PDP)[29] and Accumulated Local Effects Plot(ALE)[5] are two feature effect methods in model-agnostic methods that provide global explanations of models. They show the average behavior of models. Conversely, Local Interpretable Model-Agnostic Explanations(LIME) [73] and SHapley Additive exPlanations(SHAP) [61] are techniques for local interpretation.

LIME, proposed by Ribeiro et al., is a technique to explain individual predictions by approximating the complex and black-box model with a simpler and interpretable model. The authors generate perturbations of the original data by sampling from a Gaussian distribution to create a vicinity around the data point. This perturbed dataset is then fed into the explanation model. The objective function of LIME is formulated as:

$$\arg\min_{g} \mathcal{L}(f, g, \pi_x) + \Omega(g),$$

where $\mathcal{L}$ is the fidelity function to quantify how well the explanation model g approximates the

prediction model f on the perturbed data samples, and $\Omega(g)$ penalize the complexity of the explanation model to ensure the interpretability. LIME seeks to balance accurate approximation and the simplicity of the explanation model. LIME is a versatile and accessible method for describing features' contributions to the prediction from machine learning and deep learning. However, its stability and robustness may be challenged in some scenarios due to perturbing data points[4]. Additionally, the method generates the neighborhood of the data point using a Gaussian distribution, which ignores the correlations between features.

Lundberg et al. explain feature importance through Shapley values, introducing two SHAP models: KernelSHAP[61] and TreeSHAP[60]. SHAP leverages the concept of Shapley value from cooperative game theory to assign importance to features. KernelSHAP approximates Shapley values using a weighted linear model. In contrast, TreeSHAp is designed to use tree-based models, which can analyze non-linear relationships.

Model-specific Explanation methods

Model-specific explanation methods, such as gradient-based technique[8] and Layer-wise Relevance Propagation[7], are employed to provide interpretability for neural networks.

Gradient-based Explanation Methods Gradients indicate the direction and rate of a model's output change with respect to its inputs. We can understand how slight input variations can affect a model's output. Gradient-based techniques have been widely used to explain neural networks because of the easy access to gradient information provided by PyTorch and TensorFlow.

Gradient-based methods include Inputs×Gradients, SmoothGrad, and Integrated Gradients. The "Inputs×Gradients" method is proposed by Baehrens et al.[8]. The idea behind this method is that the gradient can show how the output varies in terms of changes in the inputs within a small neighborhood around the input. This is mathematically represented as $x_i \nabla f(x_i)$, where x_i denotes the input and $\nabla f(x_i)$ is the gradient of the model f at x_i.

However, this method has some drawbacks. It is sensitive to the magnitude of the input x_i. Meanwhile, if the model is complicated, the gradient will undergo significant changes, even with slight variations in the input. Addressing the issues, Smilkov et al.[79] introduced SmoothGrad. Instead of calculating the gradient solely at the input x_i, SmoothGrad introduces an improvement by adding normally distributed noise to the input x_i to generate a set of neighbors. The gradient is then computed as the average of the gradients at these points, providing a smoother and more stable representation of the model's sensitivity to input variations. In some cases, the explanations provided by methods like Inputs×Gradients and SmoothGrad can be noisy and unclear. To tackle this problem, Integrated Gradients[46] introduces the reference points. Reference points are inputs that do not carry any information. In many classification tasks, a reference point x^* is defined such that $f(x^*) = 0$. Integrated Gradients enhance interpretability by computing the gradient along the straight line between the input and the reference point. However, it is worth noting that identifying an appropriate reference point is often challenging and computationally expensive.

Layer-wise Relevance Propagation Explanation Methods Layer-wise Relevance Propagation(LRP) [7], is a technique used to understand the importance of input features in a neural network's output. LRP is a backpropagation process that redistributes relevance from a higher to a lower layer. Consider a neuron A with value a in layer l, which connects to N neurons B_n (where $n = 1, 2, \ldots, N$) in the subsequent layer $l + 1$. Each neuron B_n has a value b_n and is connected to A by a weight w_n. During the forward pass, neuron A contributes a value of $w_n \times a$ to each B_n. Conversely, in the LRP backpropagation process, the proportion of B_n's value attributed to neuron A is $\frac{w_n a}{b_n}$. If neuron B_n is assigned a relevance score R_n, then A receives a fraction of this score, calculated as $\frac{w_n a}{b_n} R_n$ from B_n. Summing up these contributions from all connected neurons in layer $l + 1$, the total relevance score for neuron A is $\sum_n \frac{w_n a}{b_n} R_n$.

The general formulation for LRP is

$$R_i^{(m)} = \sum_j \frac{z_{ij}}{\sum_k z_{kj}} R_j^{(n)}$$

Here, n and m are two consecutive layers. z_{ij} represents the extent for neuron i contributes to neuron j. We can easily get the conclusion that $\sum_j R_j = \sum_i R_i$ and $\sum_i R_i = f(x)$.

LRP begins with the assumption that the relevance score of the output layer, denoted as R, is approximately equal to the function output $f(x)$. This allows for the backpropagation of the relevance score with value $f(x)$ into the input layers. A key assumption in LRP is that the sum of relevance scores in each layer should satisfy the conservation property, ensuring that the total relevance is conserved across layers.

LRP is well-defined for networks with nonlinear layers. Consider a network with three layers: $l-1$, l, and $l+1$. Assume a ReLU activation function $max(0, x)$ is applied between layers $l-1$ and l, and a linear transformation $wx+b$ is applied between layer l and $l+1$. The ReLU function creates a one-to-one correspondence between layers $l-1$ and l. This correspondence implies that the transformation of relevance scores between the two layers is also one-to-one. Therefore, if a neuron in layer l has a value of 0, its contribution to the next layer is also 0 (since any weight multiplied by 0 yields 0). From this, we infer that the relevance score for such a neuron is 0. This approach effectively accounts for the nonlinear transformations within the network.

LRP uses different rules to distribute relevance scores across layers in neural networks. One such rule is the LRP-ϵ, which introduces a small positive epsilon in the denominator to stabilize the relevance. The rule is formulated as

$$R_j = \sum_k \frac{a_j w_{jk}}{\epsilon + \sum_{0,j} a_j w_{jk}} R_k,$$

where R_j denotes the relevance of a neuron, w is a weight, and a is the value of a neuron. The purpose of introducing the epsilon is to manage the distribution of relevance scores for neurons with small products of value and weight. As ϵ grows, the neuron with a small product will gain relevance scores close to zero. This adjustment is beneficial because it can filter out the contribution from neurons that have minimal impact on the output. On the other hand, the

LRP-γ, defined as

$$R_j = \sum_k \frac{a_j \cdot (w_{jk} + \gamma w_{jk}^+)}{\epsilon + \sum_{0,j} a_j \cdot (w_{jk} + \gamma w_{jk}^+)} R_k,$$

incorporates the parameter gamma, which selectively emphasizes positive contribution. By controlling γ, the rule can modulate the extent to which positive contributions are highlighted. Increasing γ gradually diminishes negative contributions, thereby focusing on the positive contribution. When applying LRP on image classification models, LRP-γ is usually preferred. It can help reduce the impact of noise and emphasize the positive contributions that correlate with high probability in the models' outputs.

In practical terms, when elucidating a neural network, we can apply different rules to different layers for a more comprehensible explanation. Specifically, in the context of image classification, it is advantageous to utilize the higher layers in conjunction with the LRP-γ rule. This approach allows for managing negative contributions, thereby emphasizing positive contributions and enhancing the model's interpretability. By selecting appropriate parameters for the LRP rules, we can achieve a clear local explanation of the model's decision-making process. In the realm of image classification or similar tasks where the influential features are well understood, assessing the contributions identified by the LRP and adjusting its parameters becomes straightforward. Conversely, when dealing with unfamiliar data, identifying the contributing features becomes more complex. This complexity makes it challenging to determine appropriate parameters for the LRP, as the lack of familiarity with data characteristics hampers our ability to assess the relevance and influence of specific features on the model's predictions.

LRP is commonly used to interpret image classification models. However, challenges arise when LRP is applied to multiplication operators within LSTM models, where both operands are inputs rather than one being a weight. A widely accepted strategy for redistribution is 'signal takes all' [6]. This approach treats the gate neurons as weights, allowing LRP to allocate all relevance scores to signal neurons.

Employing the formula presented above is not computationally efficient. Therefore, LRP is commonly reformulated into a gradient-based representation [65] to expedite the computation

process.

$$\forall k : z_k = \epsilon + \sum_j a_j \cdot w_{jk}$$
$$\forall k : s_k = \frac{R_k}{z_k}$$
$$\forall j : c_j = \sum_k w_{jk} \cdot s_k$$
$$\forall j : R_j = a_j \cdot c_j$$

The models for our project were developed using PyTorch, which facilitated easy access to these values and expedited the explanation process. Similarly, convolution operation can be reformulated in this context [91].

Chapter 3

Exploratory Data Analysis

The data presented is private and confidential. Due to privacy concerns, explicit details and sensitive information cannot be disclosed.

3.1 Manufacturing Setting

The industrial factory provides three DataFrames related to anomaly prediction: the Sensor DataFrame, the Anomaly DataFrame, and the Stability DataFrame. These DataFrames are recorded at 10-second intervals.

A **Sensor Dataframe** (see illustrative example in Figure 3.1) contains the readings for each sensor at designated times. Each sensor measures different properties and, therefore, uses different units. In our setting, various sensors and inspection points collect data at 10-second intervals, measuring variables like temperature, pressure, and speed.

In our manufacturing setting, products do not have unique IDs for tracking which makes it challenging to link specific sensor readings to corresponding anomalies. Sensors are positioned at different points along the processing line, and the distance between a sensor and the inspection

area is accounted for using time correction. Time correction refers to the expected time gap between when a product passes the sensor and when it reaches the inspection point. However, in practice, this time is not always as expected, and since we lack product IDs for more precise tracking, the corrections may be imprecise. For simplicity, we chose not to explicitly model these time corrections, allowing the model to implicitly learn any necessary adjustments from the data.

	Sensor 1 (°C)	Sensor 2 (m/s)	Sensor 3 (kPa)	...
05-21 15:00	134.4	1.2	789.4	
05-21 16:00	135.7	1.1	798.1	
05-21 17:00	134.8	0.9	800.4	
...				

Figure 3.1: Sensor Dataframe

An **Anomaly DataFrame** (see illustrative example in Figure 3.2) is generated during the inspection phase. It records the number of anomalies for each type at the same intervals as the sensor data. Anomalies detected during the inspection process, which occurs at the end of the pipeline, are categorized into several groups based on size and type. The dataset has two main categories: foreign matter and scratches, with each further divided into smaller subcategories based on size. For our experiments, we focused on the most frequently occurring type of anomaly for two reasons: it provided a sufficient amount of data for prediction, and it was considered the most significant in the production process.

	Type 1	Type 2	Type 3	...
05-21 15:00	1	4	4	
05-21 16:00	0	0	3	
05-21 17:00	3	2	0	
...				

Figure 3.2: Anomaly Dataframe

A **Stability DataFrame** includes two binary indicators: system stability and production stability. These indicators represent whether the system is stable and production is in a good or optimal condition. Stability is defined by the factory based on specific criteria. Unstable periods typically last for several days. In our experiments, we used only the data from periods when both the system and production were stable.

Given the noise in the anomaly counts and sensor data, the data was binned into 1-hour intervals, using the mean to reduce the noise.

3.2 Manufacturing Parameter Selection

In anomaly prediction, a sliding window is applied to the sensor data. For every continuous segment of several hours of sensor data, these data are used to predict future anomalies. Based on the prediction time horizon, anomaly prediction can be classified into short-term 3.3 and long-term predictions 3.4.

Figure 3.3: Short-term Prediction

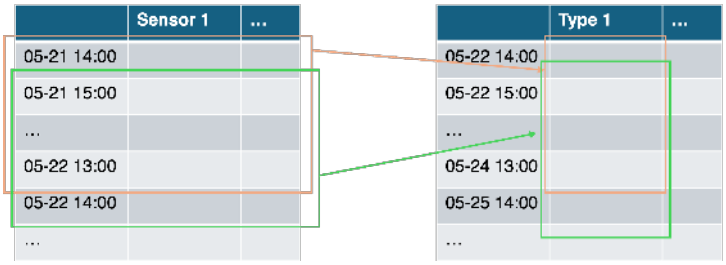


Figure 3.4: Long-term Prediction

In short-term prediction, the task is to predict the count of anomalies over the next several hours. Anomalies in such settings are often correlated with outliers in the sensor data. For example, an extremely high temperature at a certain point may lead to more anomalies. In long-term prediction, the task is to predict the number of anomalies over several days. In such settings, the expectation is that patterns within the sensor data can support long-term predictions. For example, a slow increase in temperature may not lead to immediate changes in anomalies, but its impact may become apparent over a longer period.

In our task, we would like to predict anomalies several days in advance, allowing the factory to prepare and take proactive measures to address potential quality issues before they impact the final product delivered to the customer.

Instead of predicting continuous anomaly counts for the next few hours, we concentrated on forecasting maximum anomaly counts over the next 72 hours (3 days) (refer to the illustration plot 3.5 below). Due to manufacturing requirements, we chose this 72-hour timeframe, allowing sufficient time to adjust settings. The decision to forecast the maximum value was driven by our interest in capturing peak events, which are critical for manufacturing performance. This emphasis on peak events allows us to better detect significant anomalies that could impact overall performance, as unusual sensor readings are often associated with these spikes. Additionally, the maximum anomaly count over 72 hours is likely to differ significantly from the current count, making it a more challenging and meaningful prediction target.

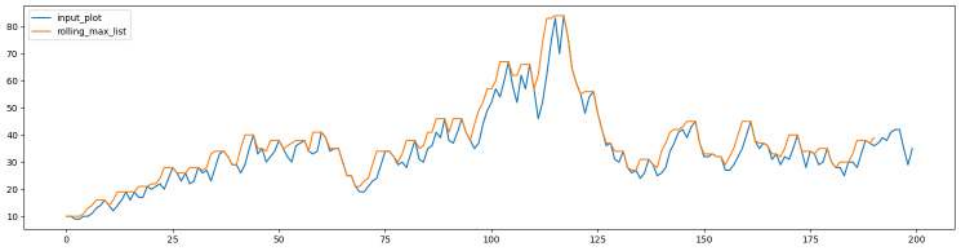


Figure 3.5: Maximum Rolling

Despite this transformation, the graph reveals significant variance in the middle, where some values are exceptionally high compared to the rest. This suggests the presence of outliers or non-standard operational periods that differ from typical inspection results and may warrant further investigation.

We categorized the anomalies into six levels: ultra-low, low, medium, high, very high, and ultra-high, thereby converting the problem into a classification task. The classification thresholds were determined using quantiles:

- Class 0 for anomaly counts below the 0.1 quantiles of all samples,
- Class 1 for counts between the 0.1 and 0.3 quantiles,

- Class 2 for counts between the 0.3 and 0.7 quantiles,
- Class 3 for counts between the 0.7 and 0.8 quantiles,
- Class 4 for counts between the 0.8 and 0.9 quantiles,
- Class 5 for counts above the 0.9 quantiles.

We opted for a 24-hour lookback window for input data for two main reasons. First, a product typically takes around 12 hours in the factory to travel from the first sensor in the pipeline to the inspection phase. Therefore, setting the lookback window to 24 hours ensures that the model captures relevant data closely tied to potential future anomalies. Second, this lookback window provided better accuracy and facilitated more effective explanatory analysis. The dataset fed into the model consisted of continuous 24-hour sensor data records used to predict anomaly categories.

3.3 Data Preprocessing

3.3.1 Sensor Selection

Sensors lacking variation in the data were considered unnecessary for predictive analysis and thus removed to streamline the dataset. We normalized the sensor data using the StandardScaler to filter out static data. Furthermore, we discarded any sensor data with variation below a predefined threshold of 1e−3. This procedure resulted in the elimination of 7 sensors from the dataset. Even with the dataset reduced, more than 300 sensors remained.

To perform feature engineering, we employed Pearson Correlation Graph Filtering. Pearson correlation [68] provides a measure of the linear correlation between two variables. We defined a threshold value, α, and connected sensors within a network if their Pearson correlation exceeded this threshold. Once the network was established, we divided the sensors into several subgraphs. Each subgraph was a distinct cluster of sensors that were not connected to sensors

in other subgraphs, meaning there were no edges (connections) between the subgraphs. This network then formed several subgraphs of highly correlated sensors. Within each subgraph, we randomly selected a representative sensor. By setting α at 0.85, we identified 52 groups and, therefore, selected 52 representative sensors. These sensors were then employed in the subsequent modeling phase.

3.3.2 Train-Test Split

The sample was generated using a rolling step of one. We divided the dataset into 21 periods, corresponding to the number of months, to account for seasonal or yearly changes. Within each period, we designated the initial 80% of the data for training and validation, and the subsequent 20% for testing. We removed the last few samples from each training segment corresponding to data in the testing period to prevent overlap between training and testing sets.

Refer to the illustration plot 3.6, where the orange blocks represent the training and validation phases, and the blue blocks represent the testing phase.

Figure 3.6: Train-Test Split Illustration

3.3.3 Data Augmentation

To prevent the model from being biased in the distribution of classes in the data, upsampling of the minority classes in the data is necessitated. In this project, the SMOTE(Synthetic Minority Oversampling Technique) [16] was utilized to upsample and generate new data through the interpolation between two points within the class.

Chapter 4

Explainable Prediction with Random Convolutional Kernels

Random convolution models have the potential to capture temporal relationships within data and enhance robustness. ROCKET is a model that utilizes random convolution in combination with linear regression. The ROCKET model has shown remarkable performance in TSEC and TSER tasks. However, linear regression often struggles to capture nonlinear relationships. In manufacturing, interpretability is crucial for companies to understand and trust the model's predictions. While convolution itself lacks interpretability, post-hoc techniques are necessary to explain the model's behavior and provide insights into its predictions.

In this chapter, we employed the ROCKET-EBM method to forecast the maximum anomaly count over the next 72 hours, using 24-hour historical sensor data. Drawing inspiration from ROCKET, we leveraged random convolution kernels to process our data. However, instead of solely relying on linear regression after the convolution process, we experimented with more sophisticated, non-linear machine learning algorithms. We utilized Explainable Boosting Machines (EBM) to enable the application of Layer-wise Relevance Propagation (LRP), thereby providing local explanations. To further elucidate the model's predictions and highlight feature importance, we utilized LRP and designed a custom explanation algorithm for the ROCKET-

EBM model.

In this chapter, we make the following contributions:

- We presented ROCKET-EBM, an extension of the ROCKET model that included an EBM classifier and a feature selection mechanism. The integration of EBM enabled the computation of local feature importance over the random convolutions to support our LRP-based explanation technique.
- We designed an LRP-based explanation algorithm to compute and visualize local explanations for the sensor inputs to the ROCKET-EBM model, thereby enhancing the interpretability of the model's predictions.
- We conducted the experimental analysis to evaluate the performance of the ROCKET-EBM model, demonstrating its effectiveness in forecasting maximum anomaly counts.
- We conducted experiments to demonstrate the faithfulness of the explanations provided by our LRP-based explanation algorithm.

4.1 Background

4.1.1 Random Convolution Kernels

RandOm Convolutional KErnel Transform(ROCKET), proposed by Dempster et al.[22], utilizes a large number of convolution filters in lieu of pre-trained ones. Unlike other random convolution kernel methods, ROCKET employs varying kernel lengths, padding sizes, and dilation. After each convolution operation, pooling techniques are applied. The authors use global max pooling and proportion of positive value(PPV), where PPV is the ratio of positive values after the convolution process. ROCKET stands out for its computational efficiency and relatively short training time compared to the conventional convolution layers. When dealing with small datasets, ROCKET surpasses conventional convolution methods, which makes it easy to

achieve accurate convolution kernels. ROCKET can employ exponential convolution kernels to get high accuracy. The ROCKET model was initially designed for TSEC. However, Tan et al. [82] also demonstrated its effectiveness in TSER.

Owing to its straightforward implementation and high accuracy, MINImally RandOm Convolutional KErnel Transform(MiniRocket)[23] introduces optimizations to the computation process of ROCKET. Unlike ROCKET, which uses various kernel lengths of $\{7, 9, 11\}$, MiniRocket employs a fixed kernel length of 9, along with a fixed padding and a fixed dilation. While the ROCKET samples kernel weights from a standard normal distribution, MiniRocket selects weights from two predetermined values, α and β. The choice of the two values is arbitrary, but α and β are set to be -1 and 2 respectively. For a kernel length of 9, 6 weights are assigned to be α, and the remaining 3 weights should be β. This configuration can significantly reduce the number of possible kernels. Meanwhile, a bias term in ROCKET is sampled from a standard normal distribution; however, MiniRocket derives a value from previous convolution outputs. Instead of applying two pooling techniques, MiniRocket only uses PPV to extract features. Despite inputting fewer features, experiments show that MiniRocket achieves similar accuracy in time series classification tasks. Building upon MiniRocket, Tan et al.[83] developed MultiRocket, which enhances the model by utilizing additional pooling techniques. Beyond the global max and PPV, MultiRocket introduces the Mean of Positive Values(MPV) and the Mean of Indices of Positive Values(MIPV), which computes the average index position within the sequence. It also considers the Longest Stretch of Positive Values(LSPV), finding the longest sequence of uninterrupted positive values.

4.1.2 LightGBM

Decision trees are a versatile and powerful machine learning model for classification and regression tasks. They predict outputs based on input features passed through each node's conditional statements. The tree uses a threshold at each node to split the data, directing it toward the appropriate leaf. The training process is to learn each node's optimal threshold and features. However, beginning with a complex decision can be computationally intensive and inefficient.

Boosting methods address this by starting with a simple, weak model and successively building up a series of simple models that correct the residuals of the ensemble. Two main techniques in boosting are AdaBoost[28] and Gradient Boosting[30]. Xgboost[18] and LightGBM[48], two powerful tree-based models, are in the family of Gradient Boosting Decision Tree(GBDT).

LightGBM introduced two novel algorithms, Gradient-based One-Side Sampling(GOSS) and Exclusive Feature Bundling(EFB), dramatically reducing training time without losing accuracy. GOSS is an algorithm that samples data for gradient-boosting trees, categorizing instances based on the magnitude of their gradients. Large gradients mean that the instance carries more information and can lead to greater information gain. GOSS selects the top k instance with large gradients for training. To maintain data balance, it also samples a proportion of instances with small gradients, adjusting the size to preserve a ratio similar to the original dataset distribution. Additionally, considering that most datasets are sparse with lots of zero values, EFB is employed to bundle some features, which reduces the dimensionality and accelerates the training process.

4.1.3 Explainable Boosting Machine

Explainable Boosting Machine (EBM)[67] is an interpretable machine learning model that extends the framework of Generalized Additive Models(GAM)[37] by integrating the gradient boosting method. This model breaks the conventional balance between interpretability and performance, making it a compelling choice for the application. The model not only delivers accurate predictions but also explains those predictions and allows for easy debugging.

EBM consists of ensembles of simple tree-based models, where each tree captures the influence of either a single feature or interactions between pairs of features. This methodology diverges from traditional GAMs, which use smooth functions to represent the effect of individual features. Instead, EBM utilizes trees to identify complex patterns and interactions.

A significant advantage of EBM is its ability to offer clear, local explanations for predic-

tions. The model can demonstrate how specific feature values or combinations contribute to a particular outcome.

4.2 Methodology

In our project, we aimed to provide an accurate classification of the maximum anomaly count for the next 72 hours. We developed a machine learning model inspired by the ROCKET model, employing random convolution kernels. This design enabled the extraction of temporal and spatial patterns from the data through random convolution and pooling techniques. Subsequently, a predictive model integrated these insights to generate accurate predictions. The architecture was designed to support explainability through LRP.

4.2.1 ROCKET-EBM Architecture

The first step was to generate random convolutions. Padding and dilation values were selected randomly, and the kernel size was randomly selected from 7, 9, and 11. The convolutional kernels were drawn from a standard normal distribution. To ensure that each kernel operated at a similar scale, the drawn weights were adjusted by subtracting the mean of the weights within each convolution. The computation of the random convolution followed the standard rules of the convolution operation.

Our study utilized a model that integrated random convolution layers followed by four pooling techniques: global maximum pooling, global mean pooling, proportion of positive values (PPV), and mean of positive values (MPV).

- **Global Max Pooling:** Selects the maximum value from all input elements.
- **Global Mean Pooling:** Computes the average of all input values.
- **Proportion of Positive Values:** Calculates the ratio of positive values to the total

number of input values.

- **Mean of Positive Values:** Computes the average of all positive input values.

We aimed to develop a highly expressive model that could not only accurately predict outcomes but also provide local feature importance. LRP has the ability to provide local explanations on the convolutional layers, but it requires that the predictive model itself can also generate local feature importance. To meet this requirement, we chose Explainable Boosting Machines (EBM) due to their efficient performance and ability to provide interpretable local explanations. However, due to a large number of extracted features, EBM struggles to handle high-dimensional data, resulting in reduced transparency and Interpretability [33]. To address this issue, we implemented a feature selection mechanism that utilized LightGBM to refine and reduce the feature set. LightGBM was selected for its rapid processing capabilities and its promising accuracy. It provides global feature importance, estimated by the gain in accuracy when each feature is used for decision splits in the decision tree. Based on these feature importances, we identified the top k features.

The selected features were then fed into an Explainable Boosting Machine (EBM) as it can provide local explanations. 4.1 illustrates the structure of the model.

4.2.2 Local Explanation via LRP

Although EBM is interpretable and can provide local explanations, introducing random convolution into the model means that EBM can only provide feature importance for the extracted random convolution rather than the original inputs. To obtain local explanations for the inputs, we utilized Layer-wise Relevance Propagation (LRP) and backpropagated the feature importance given by the EBM model.

However, the significance of feature importance for pair combinations in the EBM can be challenging to explain and visualize. Therefore, We focused on using EBM with main effects

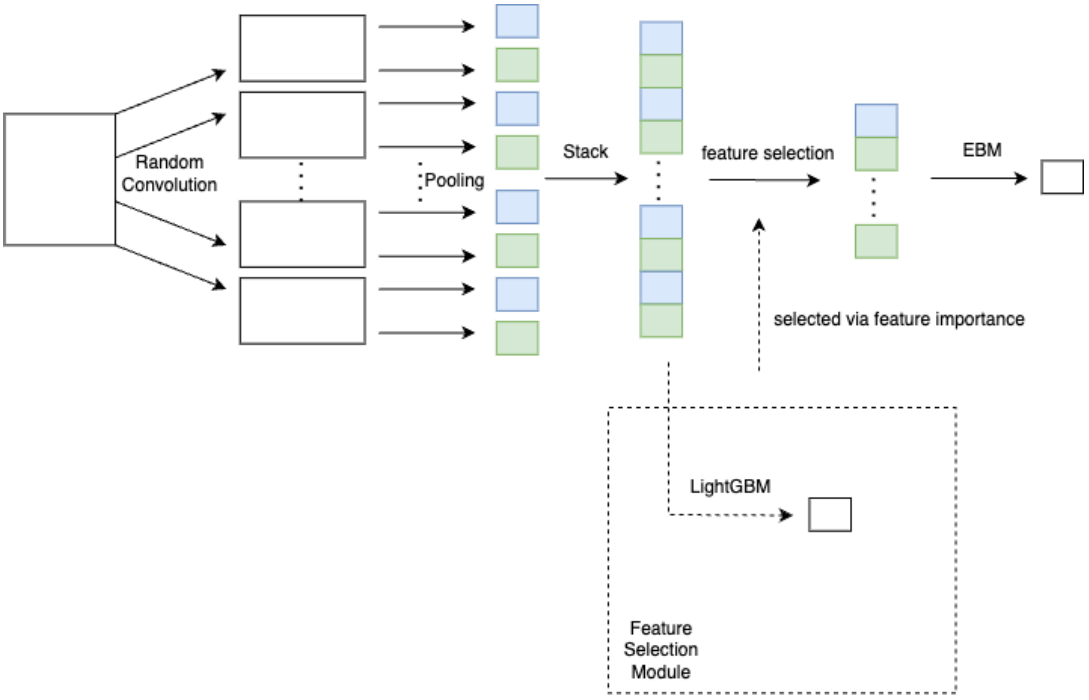


Figure 4.1: Random Convolution Model Structure

only, leaving the investigation of LRP rules for interaction effects for future work. The provided algorithm is outlined below 1.

Note: Abbreviations used in the algorithm:

- **GMXImp:** GlobalMaxPoolingFeatureImportance
- **GMNImp:** GlobalMeanPoolingFeatureImportance
- **PPVImp:** ProportionPositiveValuesFeatureImportance
- **MPVImp:** MeanPositiveValuesFeatureImportance
- **featImp:** featureImportances

The importance of features after random convolution was determined using pooling techniques. In the model, four pooling techniques were utilized, each with distinct characteristics that influence how feature importance was calculated:

Algorithm 1 Feature Importance Backpropagation Algorithm

Input: k ▷ Number of top features to select
Output: Feature importance for inputs
procedure FeatureSelectionPooling
    $importances \leftarrow$ GetFeatureImportance(EBMModel)
    $topKIndexes \leftarrow$ SelectTopKIndexes($importances, k$)
    for $i \leftarrow 0$ **to** length($importances$) - 1 **do**
        if $i \notin topKIndexes$ **then**
            $importances[i] \leftarrow 0$
        end if
    end for
    $IndexBeforeFeatureSelection \leftarrow$ TraceOriginalIndex($EBMModel$)
    $featImp \leftarrow$ array of zeros with shape of input data
    for $i \leftarrow 0$ **to** length($IndexBeforeFeatureSelection$) - 1 **do**
        if $IndexBeforeFeatureSelection[i]\%1 = 0$ **then**
            $convImportance \leftarrow$ GMXImportance($importances[i]$)
        else if $IndexBeforeFeatureSelection[i]\%2 = 0$ **then**
            $convImportance \leftarrow$ GMNImportance($importances[i]$)
        else if $IndexBeforeFeatureSelection[i]\%3 = 0$ **then**
            $convImportance \leftarrow$ PPVImportance($importances[i]$)
        else if $IndexBeforeFeatureSelection[i]\%4 = 0$ **then**
            $convImportance \leftarrow$ MPVImportance($importances[i]$)
        end if
        $featImp \leftarrow featImp +$ ApplyLRP($inputData, convImportance$)
    end for
end procedure

- **Global Max Pooling:** In the context of backpropagating feature importance, all the feature importance is attributed to the feature with the highest value. This means that only the input with the maximum value is considered influential in the prediction.
- **Global Mean Pooling:** The relevance scores assigned back to each input are proportional to its value relative to the sum of all inputs. This approach distributes the importance across all inputs based on their individual contributions to the overall average.
- **Proportion of Positive Values:** In this method, all positive values are considered equally in their contribution to the final output. The feature importance is evenly distributed among all positive inputs, regardless of their individual magnitudes.
- **Mean of Positive Values:** This distribution only focuses on positive values. The contribution of each positive number is based on its value. This method assigns more importance to higher positive values while still considering the impact of smaller positive numbers on the model's output.

After the relevance scores are determined through the pooling technique, the next step involves backpropagating these scores to the input in the convolution layer. This process can be guided by the principles of Layer-wise Relevance Propagation(LRP). Consider a convolution kernel operating on a window of length N, with input value a_i and corresponding weight w_i. Let the output of this window be c. The relevance score for each a_i is then calculated using the formula $\frac{a_i w_i}{c}$ multiplied by the relevance score in the output. Sometimes, the bias term b significantly contributed to the output. In such cases, the formula is adjusted to $\frac{a_i w_i + \frac{b}{N}}{c}$ multiplied by the relevance score in the output, preserving the relevance score for each layer.

To enhance the clarity and precision of our explanation, we introduced a small constant, epsilon (ϵ), set at 0.0001, in the denominator of our relevance score formula. This adjustment can make values contributing minimally diminished, reducing to zero. Furthermore, in the EBM feature importance evaluation, we focused on the top 10% features based on their relevance scores. This approach serves to clarify the explanation by highlighting the most powerful features, reducing the influence of noise, and decreasing the computational complexity.

4.2.3 Evaluation

We executed each model ten times with different seeds and reported the methods' performance using accuracy, F1 score, and MAE. Accuracy measures the fraction of outputs predicted correctly. The F1 score is the harmonic mean of recall and precision. MAE stands for mean absolute error. The reason for using the MAE metric in the classification problem is that the classes are categorized based on quantiles; nearby classes are similar, so we aim to penalize confusion between farther classes more severely. We presented both the mean and standard deviation(std) of the ten runs. The inclusion of standard deviation was essential due to the stochastic nature of the random convolution component. All numeric values were maintained with two decimal places.

4.3 Experiments

4.3.1 Experiment Setting

To evaluate the effectiveness of the proposed random convolutional model(ROCKET-EBM), we utilized three types of baseline models to set a benchmark.

1. **Non-Convolutional Machine Learning Models**: These include Ridge Classification (RC), Support Vector Classification (SVC), K-Nearest Neighbors (KNN), Gaussian Naive Bayes (GaussianNB), Decision Tree (DT), Random Forest (RF), XGBoost, LightGBM, and EBM. Ridge Classification offers significant interpretability but may lack accuracy. These models were fed processed sensor data to discern relationships between inputs and predict outcomes. We aimed to see if these models performed better without using random convolution kernels to identify temporal and spatial relationships within the inputs. If performance improved, it would suggest that random convolution kernels were not essential for analyzing this dataset.

2. **Basic Convolutional Models**: These models are the same as the non-convolutional ones but with the application of random convolution. This approach checks if the feature selection mechanism can identify crucial features.

None of the models discussed in this section involved neural networks; neural networks will be covered in the next chapter. The random convolution layer in these models was implemented using the FastMath package.

In our proposed ROCKET-EBM and basic convolutional models, we utilized $10,000$ random convolution kernels with sizes of 7, 9, and 11. The outputs from these convolutions were subjected to four pooling techniques: global max pooling, global mean pooling, proportion of positive values (PPV), and mean of positive values (MPV). Following this, the feature extraction phase produced $40,000$ features for each sample.

The feature selection mechanism trained the LightGBM model and selected the top 1000 features based on their feature importance.

4.3.2 Experimental Results: Predictive Performance

From the performance metrics evaluated on the testing set of non-convolutional machine learning models in table 4.1, the Gaussian Naive Bayes model exhibited the weakest performance. This was anticipated, as Gaussian Naive Bayes assumes independence between input variables, which is often untrue. Ridge Classification, Support Vector Classification (SVC), and K-Nearest Neighbors (KNN) achieved comparable results, yet they significantly underperformed compared to the tree-based models. Among these, the Decision Tree (DT) is the simplest and understandably did not surpass the more complex models such as Random Forest, XGBoost, and LightGBM. Random Forest achieved the best performance; however, it is a complex model that requires more training time. XGBoost and LightGBM attained comparable performances. Most of the non-convolutional models presented in the table showed stable performance, with standard deviations close to 0. Only the Decision Tree and Random Forest exhibited variability

in their performance, which could be attributed to the challenge of managing a large number of input variables. However, EBM did not achieve satisfactory performance because the features within EBM were not entangled together, making it difficult to capture the temporal relationships.

	Accuracy (mean)	Accuracy (std)	F1 score (mean)	F1 score (std)	MAE (mean)	MAE (std)
RC	0.50	0	0.51	0	0.67	0
SVC	0.53	0	0.54	0	0.61	0
KNN	0.51	0	0.52	0	0.65	0
GaussianNB	0.45	0	0.45	0	0.67	0
DT	0.54	0.02	0.56	0.02	0.65	0.02
RF	0.64	0.01	0.64	0.01	0.48	0.02
XGBoost	0.61	0	0.62	0	0.52	0
LightGBM	0.61	0	0.61	0	0.52	0
EBM	0.56	0	0.57	0	0.58	0

Table 4.1: Performance Metrics of Non-convolutional Machine Learning Models

Models employing untrained random convolution kernels generally showed improvements in accuracy, F1 score, and MAE, as indicated in 4.2. The addition of random convolution kernels notably enhanced the performance of the Ridge Classification and Support Vector Classification. However, specific models, such as the Gaussian Naive Bayes Model and the Decision Tree, exhibited significantly worse performance. The random convolutions increased the likelihood of correlations between input variables, which adversely affected the Gaussian Naive Bayes model. The Decision Tree model became overfitted and unstable. Random Forest, XGBoost, and LightGBM continued to outperform other models. The introduction of random convolution improved the performance of XGBoost and LightGBM by approximately 5 percent in both accuracy and F1 score. EBM also improved significantly by incorporating random convolution. The MAE of EBM performance was significantly lower than that of other models. Meanwhile, EBM's performance reached a level comparable to LightGBM and XGBoost. Models employing random convolution generally achieved better performance than those without, suggesting that convolution can assist in pattern recognition within the data. However, the performance varied considerably due to the inherent randomness and potentially insufficient convolutions.

As demonstrated by the above experiments, the employment of random convolution kernels

	Accuracy (mean)	Accuracy (std)	F1 score (mean)	F1 score (std)	MAE (mean)	MAE (std)
RC	0.56	0.02	0.57	0.02	0.57	0.03
SVC	0.59	0.01	0.60	0.01	0.53	0.02
KNN	0.56	0	0.58	0	0.61	0.01
GaussianNB	0.35	0.03	0.37	0.03	0.80	0.05
DT	0.50	0.03	0.50	0.03	0.72	0.07
RF	0.64	0.01	0.63	0.01	0.52	0.02
XGBoost	0.65	0.02	0.65	0.02	0.47	0.02
LightGBM	0.65	0.01	0.65	0.01	0.48	0.03
EBM	0.63	0	0.63	0	0.47	0.01

Table 4.2: Performance Metrics of Convolutional Non-filter Models

proved advantageous for the dataset. However, EBM suffered from less interpretability and transparency when dealing with large dimensions. To meet the interpretability requirements, we constrained the number of features input into the predictive model, EBM. By feeding fewer features into the model, we reduced the risk of overfitting and simplified the explanation of the model's decisions. We integrated LightGBM after convolutions and poolings to reduce the feature dimension to $1,000$. With the use of the feature selection mechanism, only 2.5% of the extracted features after pooling were fed into the EBM. Despite this reduction, the performance did not decrease significantly 4.3. The selection not only maintained performance but also brought sparsity, which was helpful for the subsequent explanation.

	Accuracy (mean)	Accuracy (std)	F1 score (mean)	F1 score (std)	MAE (mean)	MAE (std)
ROCKET-EBM	0.64	0.01	0.64	0.01	0.48	0.02

Table 4.3: Performance Metrics of ROCKET-EBM

The provided Table 4.4 illustrates the confusion matrices for ROCKET-EBM. It is observed that samples in classes 0, 1, and 2 are commonly misclassified, likely due to similarities in sensor data that these classes share. However, in general, the misclassified predictions are not drastically incorrect but relatively close to the actual class values, indicating that while the models may confuse classes, they do so within a close range. Class 5, which has a high count of anomalies, demonstrates a satisfactory performance. This suggests that the model is particularly adept at discerning and relating the abnormal sensor data to this class with higher

anomaly counts.

Actual \ Predicted	**Class 0**	**Class 1**	**Class 2**	**Class 3**	**Class 4**	**Class 5**
Class 0	8	29	20	0	0	0
Class 1	4	65	23	1	0	0
Class 2	0	57	517	35	0	12
Class 3	0	0	50	46	28	1
Class 4	0	0	35	28	28	96
Class 5	0	0	0	0	45	174

Table 4.4: Confusion Matrix for ROCKET-EBM

4.3.3 Experimental Results: Explainability

Toy Experiments

We conducted experiments using specially designed toy datasets to evaluate the faithfulness of the feature importance backpropagation method in our ROCKET-EBM model. We started the experiments focusing on a classification task: determining whether a given time series contains any outliers.

The toy data for this experiment was created as follows:

1. **Generation of Sine Waves:** We generated sine waves of varying frequencies and magnitudes. Each sine wave had a length of 100 data points. This step created baseline time series data for the experiment.

2. **Introduction of Outliers:** For each time series, we introduced between 0 and 10 outlier points. These outliers were defined as points where the value was either more than 1.2 times or less than 0.8 times the original value of the sine wave at that point. The selections of these outliers were random.

3. **Label Criteria:** Time series with 0 outliers are labelled as non-anomalous, assigned to class 0. Time series with 1 to 10 outliers are labeled as anomalous and assigned to class 1. To balance the datasets, 50% of time series are generated without any outliers (class 0).

The remaining 50% of the time series are generated with a random number of outliers, ranging from 1 to 10 (class 1).

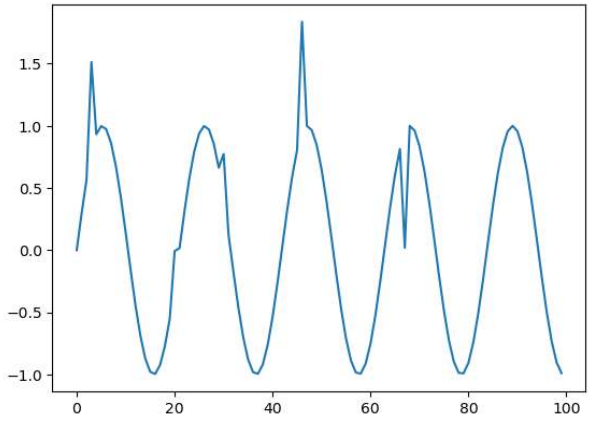


Figure 4.2: Toy Data Sample

In Figure 4.2, we selected one testing sample to illustrate the explanation through the proposed algorithm. The testing sample contained five outliers at indices 3, 20, 30, 46, and 67, and the EBM correctly predicted the presence of outliers. From the visualization, the data point at index 46 was easily identified as an outlier. Feature importance analysis revealed that the top five features with the highest importance scores were at indices 46, 27, 69, 4, and 9. Notably, the algorithm successfully detected and assigned high feature importance to index 46. While the outliers at indices 3, 30, and 67 were not precisely detected, the neighboring indices 3, 27, and 69 were identified as anomalies, which can still be regarded as a successful outcome. This result demonstrated the algorithm's effectiveness in identifying key features that significantly influence the model's predictions. It is important to note that the model might not necessarily identify all outliers explicitly due to the binary classification task. Additionally, points surrounding the outliers could also be classified as outliers.

We then generated 500 testing samples, all with 6 outliers, which the model correctly predicted. Next, we applied the explanation algorithm to these samples. Table 4.5 shows whether the post-hoc explanation correctly indicated the outliers. According to the feature importance of the inputs, we calculated the percentage of the 500 samples where the top k features in the explanation contained an outlier. Considering that neighboring points of outliers

can also be classified as outliers, we allowed a tolerance of 2, meaning within a range of 2 of this point.

	Percentage (%)
Top 1	68.2
Top 4	93.8
Top 7	98.4
Top 10	99.6

Table 4.5: Percentage of samples where the top k features in the explanation contain any outlier.

The table shows that almost all explanations correctly indicated outliers from the top 10 feature importance scores, demonstrating the algorithm's high accuracy. Additionally, 61% of the outliers among the 6 outliers in each sample were correctly identified through the top 10 feature importance explanation.

Explanation Visualization

Our toy experiments demonstrate that the algorithm effectively identifies inputs with high feature importance. When we applied this algorithm to our factory data model, we analyzed several specific data samples that were accurately predicted. To visualize the feature importance, we used a heatmap and scaled the feature importance values between -1 and 1. Features with dark red indicate a positive contribution to the final output, while features with dark blue indicate a negative contribution. Blank spaces mean the features are not related to the prediction.

Figure 4.3 shows the explanation of two samples in class 5. The x-axis represents the sensors, and the y-axis represents the 24-hour lookback window. Although they belong to the same class, the local explanations can be significantly different. However, they still share some similar sensor features.

To validate the explanations provided by the post-hoc technique and to further decrease the number of inputs, we used the testing set that was correctly predicted by the model. By aggregating the local explanations of the testing set together (see Figure 4.4), we retained the

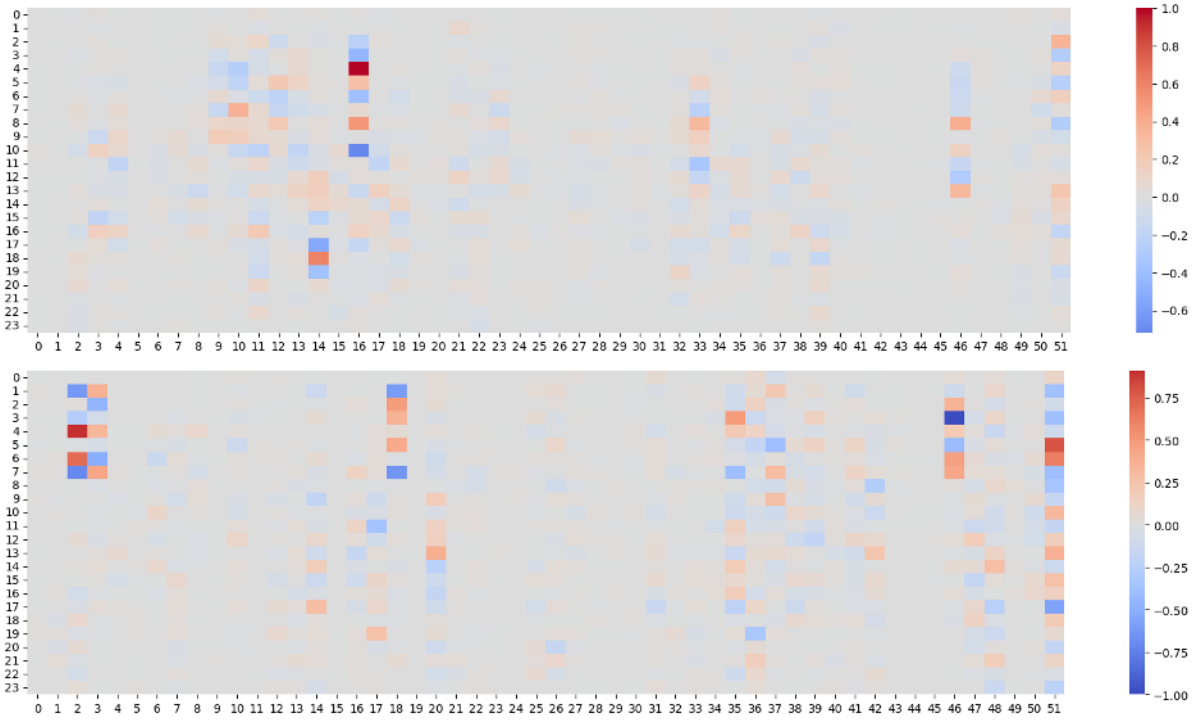


Figure 4.3: Feature Importance Heatmap for Testing Samples in Class 5 for ROCKET-EBM

top 30 features with the highest absolute values, setting all remaining features to zero. The shape of the data was preserved to allow for convolution, and the model was rerun with these selected features.

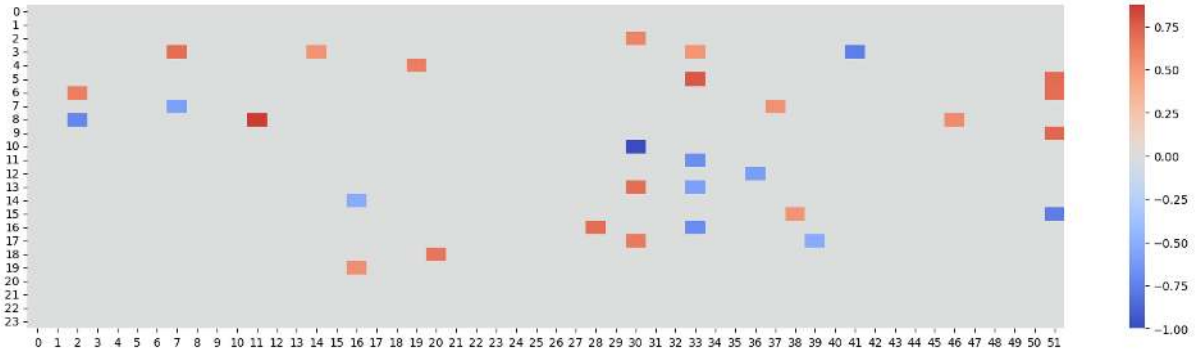


Figure 4.4: Aggregated Feature Importance Heatmap for ROCKET-EBM

The performance of the sparse ROCKET-EBM model significantly improved, particularly in terms of MAE, which observed a substantial decrease 4.6 4.7. This improvement was achieved using just 30 features, which greatly increased the sparsity of the input. As a result, the model

became more robust and less prone to overfitting. This outcome suggests that the explanations provided by the post-hoc technique play a crucial role in enhancing the model's performance.

	Accuracy	F1 score	MAE
Spare ROCKET-EBM	0.68	0.67	0.36

Table 4.6: Performance Metrics of Sparse ROCKET-EBM

Actual \ Predicted	**Class 0**	**Class 1**	**Class 2**	**Class 3**	**Class 4**	**Class 5**
Class 0	35	1	21	0	0	0
Class 1	6	85	2	0	0	0
Class 2	0	58	503	59	0	1
Class 3	0	0	50	34	41	0
Class 4	0	0	31	35	41	80
Class 5	0	0	2	0	32	185

Table 4.7: Confusion Matrix for Sparse ROCKET-EBM

Due to nonlinearity, it's challenging to predict whether increasing the value of a significant feature will increase the output for these complex models. The local explanation is intended to describe the feature's importance for the particular output.

4.4 Conclusion

In this section, we utilized ROCKET-EBM to predict the class of maximum future 72-hour anomaly counts based on historical sensor data. ROCKET-EBM combines random convolution with EBM, where random convolution enables the model to capture temporal relationships and robustness, and EBM facilitates local explanations through a post-hoc explanation technique. To enhance the interpretability of EBM, we employed LightGBM to filter the extracted inputs before feeding them into the EBM. We then applied the post-hoc technique to ROCKET-EBM to provide local explanations, with EBM allowing us to determine the local feature importance of the extracted features. However, extending these local explanations to the input sensors posed a challenge due to the convolutional layer. To address this, we employed Layer-wise Relevance Propagation (LRP), which considers both the weights and input values to backpropagate feature importance to the original input layers. We designed an LRP-based algorithm specifically to

explain the feature importance in ROCKET-EBM.

From the experiments, ROCKET-EBM significantly outperformed non-convolution classical models. By using EBM, the model achieved a lower MAE than other random convolutional models.

We validated the algorithm on a toy model, which was a binary classification task in anomaly detection, indicating the effectiveness of our approach. We then implemented the algorithm on our actual prediction task. By aggregating several local explanations, we identified and selected the top 30 features. The sparse ROCKET-EBM utilized only these 30 features and still achieved considerable performance. The sparse ROCKET-EBM outperformed the original model because it was less prone to overfitting, indicating that the identified 30 features were pivotal for our predictions.

Chapter 5

Explainable Prediction with Trained Convolutional Kernels

In the previous chapter, we demonstrated that models incorporating random convolution outperformed those that did not, highlighting the effectiveness of convolution for this particular task. Convolutions are adept at extracting temporal patterns from data to generate valuable features. Utilizing untrained random convolutions provides robust performance, which is attributed to the large number of random kernels. Although random convolutions do not require training, applying a large number of convolutions at once to make predictions is time-consuming despite the use of optimized computation. Additionally, implementing them sequentially, as in stacked convolution, is challenging given the extensive number of convolutions.

Zeiler and Fergus [95] employed a DeConvNet architecture to visualize the functionality of each convolutional layer. Their insights revealed that lower layers tend to capture detailed features and noise, whereas higher layers discern more significant object components. Stacked convolutions tend to filter out more noise due to this hierarchical processing. Hence, in this chapter, we developed prediction models for manufacturing anomalies based on trained convolutions. Concretely, we implemented stacked convolutions to refine the feature extraction process.

The InceptionTime model [44] is a unique adaptation of the Inception model [80] specifically designed for time series analysis. Unlike conventional neural networks, the InceptionTime model operates with parallel convolutions with varying receptive fields. This innovative approach allows it to simultaneously capture patterns at multiple temporal scales, setting it apart from its counterparts.

In this chapter, we make the following contributions:

- We developed explainable prediction models for manufacturing anomaly prediction based on the InceptionTime [44] architecture and the LRP rules for deep neural network [7].
- We conducted extensive experiments, comparing our model to Multilayer Perceptrons [38], Convolutional Neural Networks [75], and ResNet [39]. We observed that our model significantly outperformed these baseline models.
- We identified a potential issue with layer-wise relevance score propagation. To address this, we applied LRP on the InceptionTime model with intermediate relevance score filtering. By selecting only the top 30 features based on their relevance scores, the sparse model was able to reproduce the original model's performance. This explanation proved useful for our model.

5.1 Background

5.1.1 Convolution

Convolution is a mathematical operation defined as the integral of the product of two functions after one is reversed and shifted,

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau$$

This technique is widely used across various fields, such as image processing, acoustics, and electrical engineering.

The convolution neural network(CNN), proposed by Yann LeCun et al., comprises three integral components: convolution, pooling, and non-linear layers. The convolution operation can be visualized as a sliding window that captures the local relationships within the data.

For example, consider an image with dimensions of height(H) and width(W) containing C channels, leading to a data shape denoted as (C, H, W). In the case of RGB images, the channel count is typically 3. If we apply a convolution kernel of size $m \times n$, it is essential to note that each channel has distinct kernel weights. Besides kernel size, convolution has parameters like padding, stride, and dilation. Padding involves adding extra 'blank' data around the input's edges, preventing the reduction of feature map dimensions after successive convolutions and ensuring that edge information receives attention. Stride determines the step size the convolution filter moves across the input; it defaults to 1 but can be adjusted according to specific requirements. Dilation refers to the spacing between the kernel elements; it allows the filter to cover a larger input area and capture more global features without increasing the number of parameters. The formulas for the output height (H_{out}) and width (W_{out}) of a convolution layer are given by:

$$H_{\text{out}} = \left\lfloor \frac{H + 2 \times \text{padding} - \text{dilation} \times (m-1) - 1}{\text{stride}} + 1 \right\rfloor$$
$$W_{\text{out}} = \left\lfloor \frac{W + 2 \times \text{padding} - \text{dilation} \times (n-1) - 1}{\text{stride}} + 1 \right\rfloor$$

Padding, stride, and dilation parameters in convolution neural networks can be specified in two dimensions. However, using a single value for both dimensions is common practice for simplicity. After convolution filters are applied to each channel, the resulting features from each channel are aggregated through a weighted sum, resulting in the output channel (C_{out}).

Subsampling in convolution neural networks refers to pooling techniques that reduce the dimensions of the feature maps. The most commonly used pooling methods are max pooling

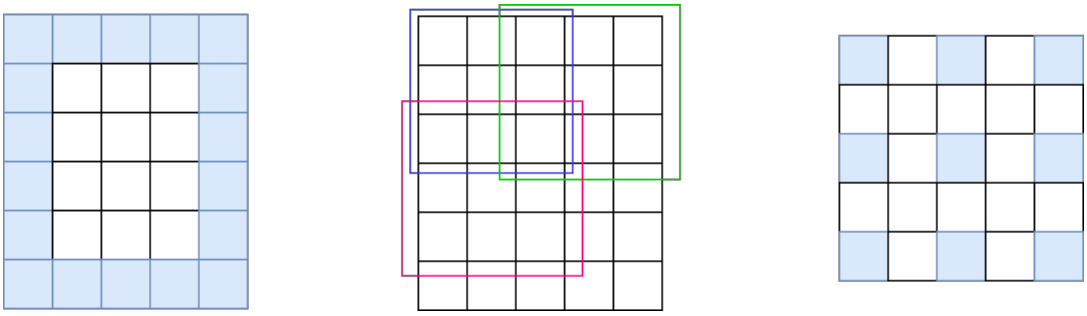


Figure 5.1: The left image demonstrates padding with a value of 1, the center image depicts a stride of 2 for convolution, and the right image shows a convolution with a dilation rate of 2.

and average pooling.

AlexNet [51], building upon LeNet-5, pioneered in applying convolution neural networks to outperform other image classification methods. Additionally, it was the first to employ GPUs to accelerate computation.

5.1.2 ResNet

When addressing complex problems, it is often necessary for neural networks to be profound, involving a significant number of layers. However, several challenges can emerge as the number of stacked layers increases. Performance may begin to degrade, and issues such as gradient vanishing and explosion can occur. To restain these problems, He et al. [39] introduced a technique known as residual connection, which leverages identity mapping, represented as $F(x)+x$. This approach has been shown through experiments to enable deep neural networks to converge more rapidly and achieve substantial performance improvements. Despite the effectiveness of residual connection in mitigating deep learning challenges, their theoretical underpinnings remain elusive. One plausible explanation for their success is that they build shallow networks within the architecture, alleviating gradient vanishing and explosion issues.

5.1.3 InceptionTime

The InceptionTime model [44] is an adapted version of Inception architecture, initially developed for image classification. The original Inception model[80], known for its efficacy in image recognition tasks, leverages the power of convolution layers. However, traditional approaches often involve stacking multiple convolution layers to enhance performance, which can lead to issues such as overfitting and gradient vanishing, as well as increased computational demands. In contrast, the Inception architecture addresses these challenges by performing parallel convolutions of varying kernel size on the input, allowing it to capture complex features at different scales effectively. Specifically, the Inception model incorporates three types of convolutions with kernel sizes of $1 \times 1, 3 \times 3$, and 5×5, alongside a max pooling operation. The outputs of these operations are then concatenated. A key efficiency feature of the Inception architecture is using 1×1 convolutions, which reduce dimensionality. This approach not only helps control the computational cost but also aids in mitigating the risk of overfitting.

Inception v4[81] marks a significant evolution in Inception by integrating the concept of residual blocks. Initially popularized by ResNet, these residual blocks facilitate deeper networks by enabling more efficient training and mitigating the vanishing gradient problem. The InceptionTime model, drawing inspiration from Inception v4, incorporates these advancements but with a tailored approach for time series classification. The InceptionTime architecture is an ensemble of 5 Inception blocks. An essential adaptation in InceptionTime is the modification of kernel sizes in convolution layers. Unlike Inception v4, which is optimized for image data, InceptionTime adjusts its kernel size to better suit the characteristics of time series data.

5.2 Methodology

5.2.1 InceptionTime Architecture

The InceptionTime architecture [44] is designed to harness a range of receptive field sizes in its convolutions and max pooling to extract pertinent information from the data. The structure of an InceptionTime block is depicted in 5.2. The model comprises multiple such blocks to refine the information from the dataset. Within an InceptionTime block, the input data is initially processed through a bottleneck convolution. This type of convolution, specifically a 1×1 1D convolution, reduces dimension by decreasing the number of input channels, thereby making computation more efficient. Following this, the bottleneck convolution's output is processed by three separate convolutions with varying receptive field sizes - small, medium, and large - to capture diverse scales of information. Simultaneously, max pooling is applied to the input to abstract information directly, preserving vital data from the original input. A subsequent bottleneck convolution then reduces the dimension of this pooled output. Finally, the outputs from the convolution and max pooling pathways are concatenated to form a unified feature representation.

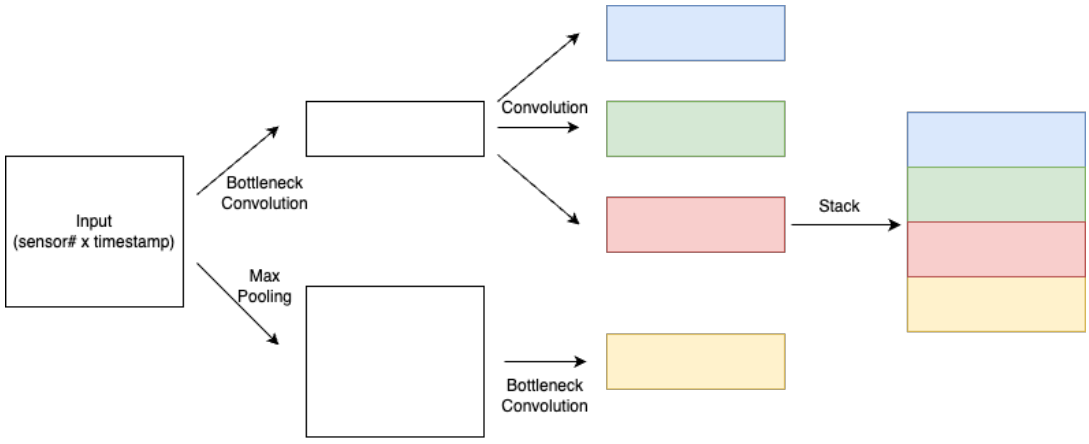


Figure 5.2: InceptionTime Block Structure

The provided figure 5.3 illustrates the concept of stacked convolutions employed within the InceptionTime model to enhance data filtering for improved performance. In this model,

InceptionTime blocks are stacked and interconnected with skip connections. In the diagram, blue blocks denote InceptionTime blocks, the green block represents the activation function (ReLU in this case), yellow indicates average pooling layers used to reduce dimensionality, orange signifies a flattened operation to reshape the data, and red depicts a linear layer and a softmax function which aggregates the information into a specific number of class values. The normalized data passes through three InceptionTime blocks, with the original data added back before the InceptionTime block via skip connections, followed by the application of the ReLU activation function. This procedure is repeated, and the data is subjected to average pooling.

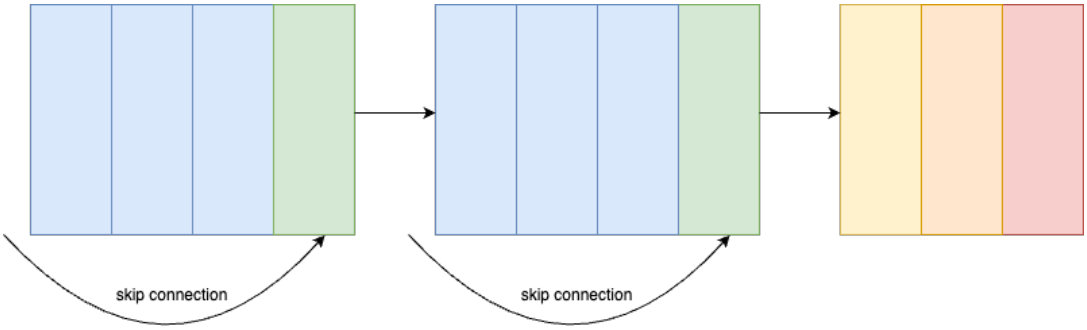


Figure 5.3: InceptionTime Model Structure

5.2.2 Local Explanation via LRP

A post-hoc technique was then employed on the InceptionTime model. The InceptionTime model is a neural network where Layer-wise Relevance Propagation (LRP) can be effectively applied to each layer, allowing relevance scores to be backpropagated to the input. LRP is well-suited for linear and convolution layers. In practice, the neural network outputs a probability for each class at the output layer. To apply LRP, the highest probability—corresponding to the predicted class—is retained, while the probabilities for all other classes are set to zero. This selection criterion focuses the backpropagation on the most relevant output. By following the rules of LRP, we can compute and assign relevance scores to the inputs, thereby identifying the most influential factors contributing to the prediction.

5.3 Experiments

5.3.1 Experiment Setting

Figure 5.4 illustrates the architecture of MLP, FCN, and ResNet, which served as baseline models in this section. MLP, or Multilayer Perceptron, employs linear layers and ReLU activations; the dashed lines indicate dropout layers. FCN stands for Fully Convolution Networks, which utilizes sequences of convolution layers followed by batch normalization and ReLU activations. ResNet, on the other hand, incorporates numerous convolution layers and employs skip connections to facilitate deeper network training.

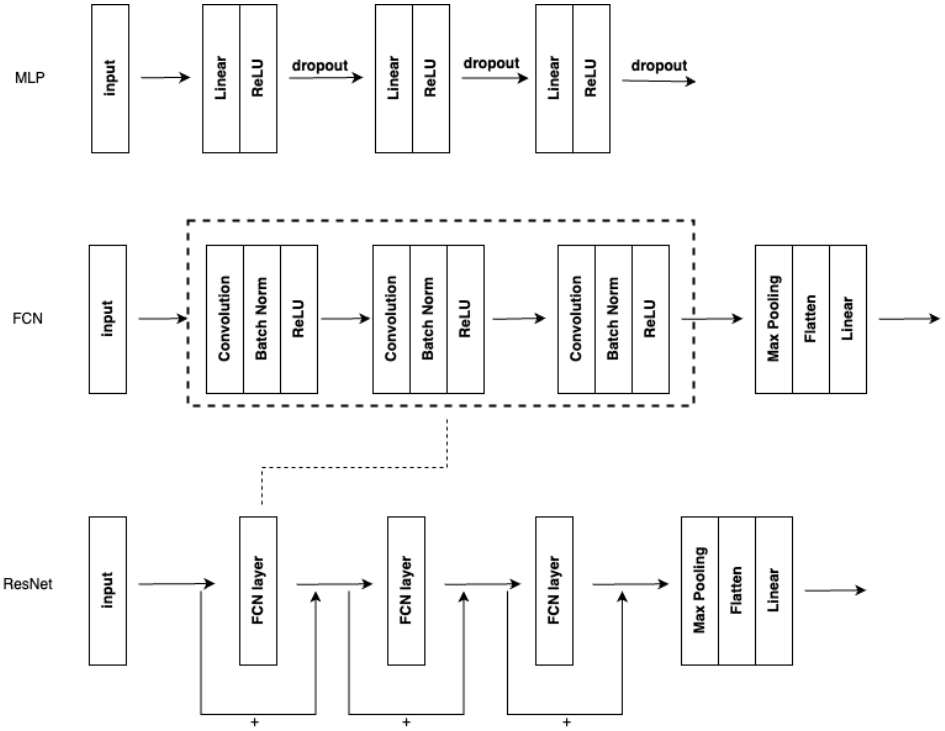


Figure 5.4: Baseline Models in [92]

The models MLP, FCN, ResNet, and InceptionTime represent a progression of increasing complexity. MLP is the simplest, while InceptionTime is the most complex. The MLP used in this experiment is a linear model, which does not involve ReLU activations. While simple, it has more parameters than linear regression and serves as a baseline for deep learning models in this experiment. By tracking the weights, we can interpret this model; however, due to its simplicity,

it struggles to capture complex data patterns. The FCN model incorporates convolution layers to improve pattern recognition. ResNet further enhances this by adding additional stacked convolution layers and skip connections, facilitating deeper network training and potentially improved performance. InceptionTime, the most complex model, uses parallel convolutions, utilizing three convolutions of varying receptive field sizes and stacked convolutions to capture intricate patterns in the data.

We conducted experiments with MLPs using three different hidden layer dimensionalities: 128, 256, and 512. These configurations were also tested on an MLP with dropout layers(MLP*) set at a rate of 0.5.

For the FCN model, we included three convolution layers, each with 128 dimensions. In the FCN variant with small kernels (FCN_s), the kernel sizes for the first, second, and third layers were set to 8, 5, and 3, respectively. Conversely, in the FCN variant with large kernels (FCN_l), the kernel sizes for the three layers were 15, 11, and 7.

The ResNet model was configured similarly, with kernel sizes of 8, 5, and 3 for ResNet_s, and 15, 11, and 7 for ResNet_l. Meanwhile, the InceptionTime model utilized parallel convolutions with kernel sizes of 3, 7, and 11.

The batch size for training was set to 512. For this task, we selected cross-entropy loss as the loss function and employed the Adam optimizer. Since cross-entropy loss inherently combines the log-softmax function, the softmax layers were removed from the models.

5.3.2 Experimental Results: Predictive Performance

According to the results presented in Table 5.1, introducing the dropout mechanism helps improve accuracy and robustness. Models with dropout exhibit smaller variance compared to those without. Although ridge classification and MLP are both linear models, the inherent complexity of the MLP is attributed to its superior performance compared to ridge classification.

By integrating convolution layers into our architecture, the FCN generally outperformed the MLP 5.2. The convolution layers proved effective at capturing critical patterns for accurate predictions. In comparing FCN configurations, the model employing smaller kernels slightly outperformed the version with larger kernels. This observation suggests that sudden changes in sensor readings are closely related to future scratches. Using a smaller kernel size can help the model focus more on these abrupt changes in the data.

ResNet with smaller kernel sizes performed better than those with larger ones. In deep models, small kernels are generally more effective at capturing information. Compared to the FCN, ResNet did not significantly improve accuracy and F1 score. However, ResNet achieved a much lower MAE. Since neighboring classes had similar characteristics, ResNet performed better than FCN in this context.

InceptionTime significantly outperformed ResNet and FCN, primarily due to its use of parallel convolutions, enabling it to identify patterns from different receptive fields. Regarding accuracy, F1 score, and MAE, InceptionTime demonstrated superior performance compared to the other models.

	Accuracy (mean)	Accuracy (std)	F1 score (mean)	F1 score (std)	MAE (mean)	MAE (std)
MLP_128	0.59	0.03	0.58	0.03	0.63	0.05
MLP_256	0.58	0.04	0.57	0.05	0.64	0.11
MLP_512	0.58	0.03	0.57	0.03	0.61	0.08
MLP*_128	0.60	0.01	0.58	0.02	0.57	0.04
MLP*_256	0.59	0.01	0.58	0.01	0.58	0.03
MLP*_512	0.60	0.02	0.59	0.02	0.57	0.05

Table 5.1: Performance Metrics of Multilayer Perceptron

	Accuracy (mean)	Accuracy (std)	F1 score (mean)	F1 score (std)	MAE (mean)	MAE (std)
FCN_s	0.63	0.01	0.61	0.01	0.52	0.03
FCN_l	0.61	0.04	0.58	0.04	0.55	0.05
ResNet_s	0.61	0.03	0.60	0.03	0.60	0.08
ResNet_l	0.60	0.02	0.57	0.02	0.57	0.05
Inception	0.66	0.02	0.64	0.02	0.45	0.05

Table 5.2: Performance Metrics of Convolution Neural Networks

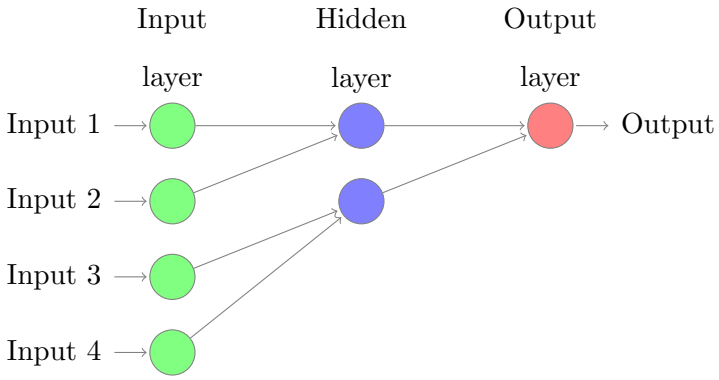
Compared to the ROCKET-EBM, InceptionTime performed slightly better. The confusion matrix (see Table 5.3) for InceptionTime shows similarities to that of the ROCKET-EBM, indicating that certain classes are frequently misclassified as their neighboring classes.

Actual \ Predicted	**Class 0**	**Class 1**	**Class 2**	**Class 3**	**Class 4**	**Class 5**
Class 0	0	40	17	0	0	0
Class 1	0	86	7	0	0	0
Class 2	0	54	476	47	44	0
Class 3	0	0	67	37	0	21
Class 4	0	0	36	35	56	60
Class 5	0	0	39	0	21	159

Table 5.3: Confusion Matrix for InceptionTime

5.3.3 Experimental Results: Explainability

LRP is a powerful tool for explaining neural networks. However, due to their complexity, LRP can sometimes be noisy and lead to potential issues. LRP struggles to effectively manage positive and negative contributions within the network, indicating a need to reassess the LRP formula. To illustrate this issue, we will analyze a simple neural network as an example.



The neural network is structured as follows: the first layer consists of four neurons, the second layer has two neurons, and the third layer contains a single neuron. In the first layer, two neurons are connected to a single neuron in the second layer, while the remaining two neurons from the first layer connect to the other neuron in the second layer. Subsequently, both neurons in the second layer connect to the neuron in the third layer.

Assuming all weights in the neural network are 1, and there is no bias term, with the output being 1.001. The neurons in the second layer have values of 1(top) and 0.001(bottom), respectively. Let's assume the neuron in the third layer carries a relevance score of 1.001. Consequently, the relevance scores sent to the second layer are 1 for the top and 0.001 for the bottom neuron, indicating that the top neuron in the second layer is more significant than the lower one. In the first layer, the values from top to bottom are $0.5, 0.5, 2.001$, and -2, respectively. Thus, the relevance scores for the first and second neurons in the first layer are both 0.5. The third and fourth neurons carry relevance scores of 2.001 and -2, respectively, suggesting that they convey more information than the first and second neurons. However, revisiting the second neuron in the second layer, which is nearly non-informative with a relevance score of 0.001, we encounter a contradiction. This contradiction suggests an inconsistency in the distribution of relevance scores when both positive and negative values are connected to a neuron.

InceptionTime contains several layers, increasing the likelihood of this issue occurring in our explanations. To address this, we applied intermediate relevance score filtering, meaning that we only backpropagate the top k relevance scores for each layer. This approach can cause relevance scores with small absolute values to disappear.

For the LRP algorithm, we applied top 80 percent relevance score filtering specifically to the flattened operation layer 5.3. This means that only 80 percent of the relevance scores are retained in this layer. In other words, we preserved 80 percent of the relevance score output from the linear layer. We tuned the relevance score filtering for different layers, but applying filtering to the convolutional layers may decrease the explanations' quality.

Figures 5.5 show the feature importance of two test samples. We observed that the explanations for both samples are similar.

We selected the top 30 features with the highest absolute values by aggregating feature importance across the correctly predicted test set. Figure 5.6 shows the heatmap of the aggregated feature importance, with values outside of these 30 features masked.

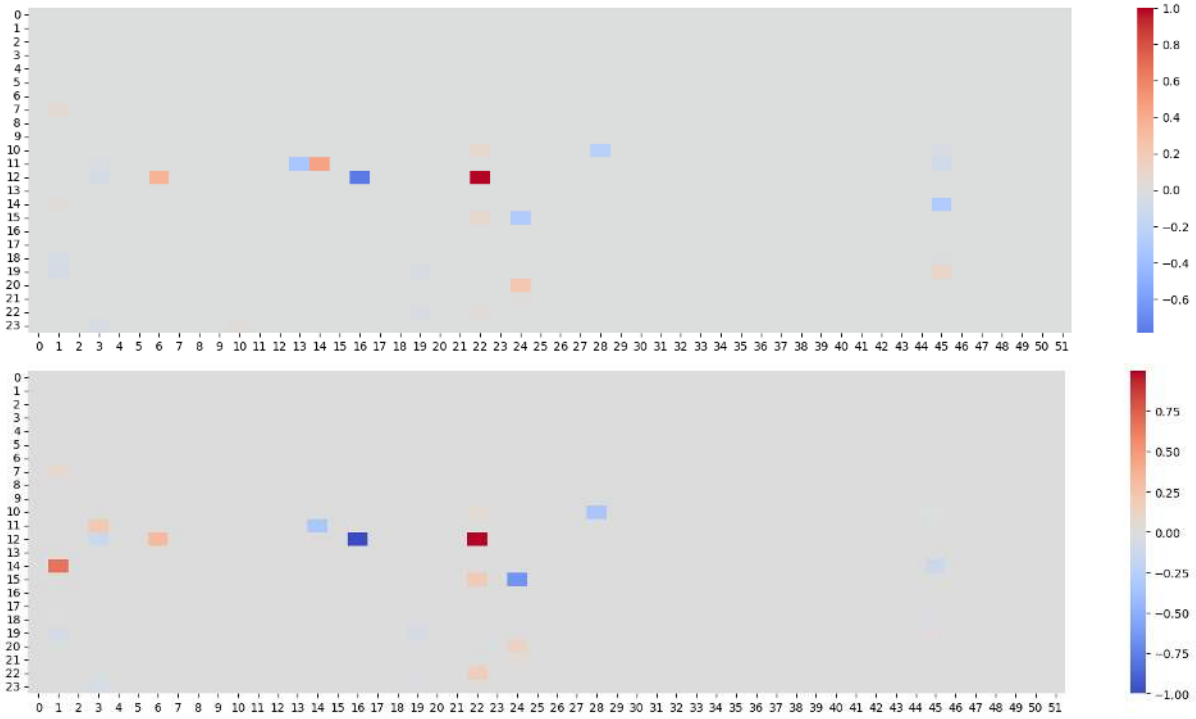


Figure 5.5: Feature Importance Heatmap for Testing Samples in Class 5 for InceptionTime

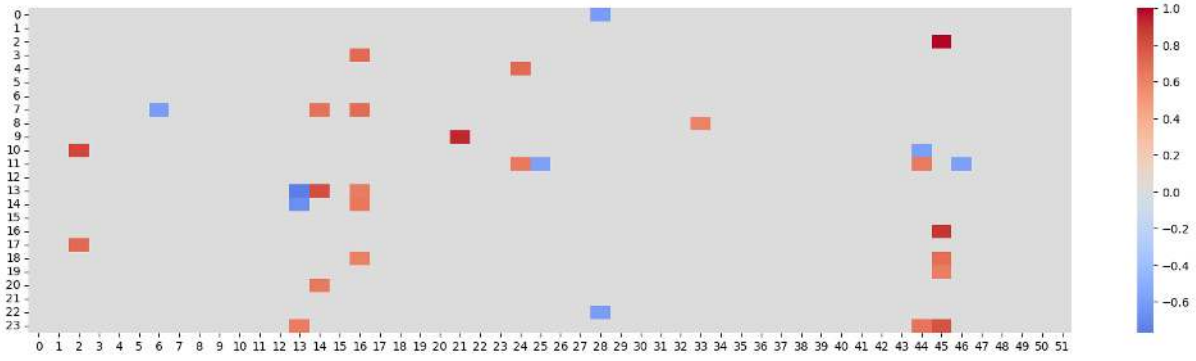


Figure 5.6: Aggregated Feature Importance Heatmap for InceptionTime

We retrained the model using only the top 30 features, masking the other inputs by setting them to zero. The performance of the sparse InceptionTime model was able to replicate the original model's performance with just these 30 features 5.4. This indicates that the features identified by LRP are critical for the InceptionTime model.

	Accuracy	F1 score	MAE
Spare InceptionTime	0.63	0.62	0.47

Table 5.4: Performance Metrics of Sparse InceptionTime

5.4 Conclusion

In this chapter, we tackled the same task as in Chapter 4, but this time employing a neural network to classify the maximum anomaly count over a forthcoming 72-hour period based on 24-hour historical data from 52 sensors.

Drawing inspiration from the simultaneous use of various convolution sizes in the random convolution approach, we applied a time series adaptation of the Inception model named InceptionTime. This model leverages both parallel and stacked convolutions. The parallel convolutions, varying in size, are adept at detecting information across different receptive fields. Meanwhile, the stacked convolutions delve deeper, filtering out the noise to capture more refined information. We used MLP, FCN, and ResNet as baselines. InceptionTime significantly outperformed these models and performed slightly better than the ROCKET-EBM.

We applied Layer-wise Relevance Propagation (LRP) to the InceptionTime model to identify key features in the input data and explain the model's predictions. Given the complexity of InceptionTime, we recognized potential issues with LRP. To address these, we implemented intermediate relevance score filtering. The model, using only the 30 features identified by LRP, could reproduce the model's performance.

Chapter 6

Conclusion

6.1 Summary

In our project, the goal was to design a model that could accurately predict manufacturing anomalies while also being fully explainable. To achieve this, we leveraged sensor data to forecast future anomaly counts. Due to the inherent noise in anomaly count data, we chose to estimate the maximum count over the upcoming 72-hour period, categorizing these counts into six quantile-based classes. Additionally, we aimed to utilize post-hoc techniques to provide local feature importance.

Convolutions enable models to capture temporal relationships in time series data effectively. We employed both random and trained convolutional models: ROCKET-EBM and InceptionTime. ROCKET-EBM is inspired by the ROCKET model, utilizing a large number of random convolutions. These random convolutions add robustness to the model, leading to strong performance. By integrating the Explainable Boosting Machine (EBM), ROCKET-EBM can provide local explanations through Layer-wise Relevance Propagation (LRP). InceptionTime, on the other hand, is a neural network that uses parallel and stacked convolutions to capture complex patterns. Both models demonstrated superior prediction performance, with InceptionTime

slightly outperforming ROCKET-EBM.

To explain the two models, we applied Layer-wise Relevance Propagation to provide local explanations. We designed a LRP-based algorithm specifically for ROCKET-EBM. In the case of InceptionTime, LRP is well-defined for convolutional and linear layers, allowing it to be directly applied to InceptionTime.

The top features selected in ROCKET-EBM using the LRP-based algorithm were validated as crucial to the model's performance. By utilizing these identified features, ROCKET-EBM can enhance its testing performance. Given the complexity of InceptionTime and the potential issues and noise associated with LRP, we applied LRP with relevance score filtering to eliminate small intermediate relevance scores. The 30 features selected through this process were sufficient to reproduce the model's performance.

When comparing the performance of ROCKET-EBM and InceptionTime with all features included, both models achieved similarly strong results, with InceptionTime slightly outperforming ROCKET-EBM. InceptionTime was able to identify more accurate patterns within the data. However, the explanations provided by ROCKET-EBM were much more interpretable than those from InceptionTime. This is supported by the fact that a sparse version of ROCKET-EBM was able to achieve better performance than the original version, whereas InceptionTime did not outperform its original variant. This is mainly because, when applying LRP to the model, noise and errors are more likely to occur, particularly in models with more layers. InceptionTime, with its deeper architecture, is more prone to these issues compared to ROCKET-EBM, which has fewer layers. As a result, LRP can provide a clearer and more reliable explanation for ROCKET-EBM.

In practice, the factory can identify potential sensor issues by analyzing the features associated with low and high anomalies, as explained by the LRP. By examining the threshold of sensor readings that differentiate low and high anomalies, it becomes possible to define the sensor's safe operating range [32][40].

6.2 Future work

- **Quantile**: The categories in this task are classified based on quantiles. However, this classification technique may lead to incorrect boundaries, causing predictions to fall into neighboring classes. Therefore, the quantiles should be designed based on the distribution of the data. Additionally, we will impose heavier penalties on predictions that deviate significantly from the actual class.

- **Interaction**: In Chapter 4, we focused solely on EBM without considering interactions. However, we will explore how to apply LRP in cases where interactions are present.

- **Model**: Post-hoc explanation techniques suffer from certain limitations [4][42]. Although LRP can explain the models' predictions, it becomes inaccurate when relevance scores include both positive and negative values. While EBM is inherently interpretable, it does not deliver satisfactory performance in our task. Additionally, introducing convolution to EBM, as seen in ROCKET-EBM, compromises its inherent interpretability. To address these challenges, we will explore the use of more powerful inherently interpretable models, such as Neural Additive Models [1], to assess their effectiveness for this task. NAM is a generalized additive model similar to the EBM but leverages neural networks to construct additive functions. The complexity of neural networks potentially offers more powerful capabilities.

Bibliography

[1] Rishabh Agarwal et al. "Neural additive models: Interpretable machine learning with neural nets". In: *Advances in neural information processing systems* 34 (2021), pp. 4699–4711.

[2] Abhishek Agrawal et al. "An application of time series analysis for weather forecasting". In: *International Journal of Engineering Research and Applications* 2.2 (2012), pp. 974–980.

[3] Muhammad Aurangzeb Ahmad, Carly Eckert, and Ankur Teredesai. "Interpretable machine learning in healthcare". In: *Proceedings of the 2018 ACM international conference on bioinformatics, computational biology, and health informatics*. 2018, pp. 559–560.

[4] David Alvarez-Melis and Tommi S Jaakkola. "On the robustness of interpretability methods". In: *arXiv preprint arXiv:1806.08049* (2018).

[5] Daniel W Apley and Jingyu Zhu. "Visualizing the effects of predictor variables in black box supervised learning models". In: *Journal of the Royal Statistical Society Series B: Statistical Methodology* 82.4 (2020), pp. 1059–1086.

[6] Leila Arras et al. "Explaining recurrent neural network predictions in sentiment analysis". In: *arXiv preprint arXiv:1706.07206* (2017).

[7] Sebastian Bach et al. "On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation". In: *PloS one* 10.7 (2015), e0130140.

[8] David Baehrens et al. "How to explain individual classification decisions". In: *The Journal of Machine Learning Research* 11 (2010), pp. 1803–1831.

[9] Dimitris Bertsimas and Jack Dunn. "Optimal classification trees". In: *Machine Learning* 106 (2017), pp. 1039–1082.

[10] George EP Box et al. *Time series analysis: forecasting and control*. John Wiley & Sons, 2015.

[11] Leo Breiman. *Classification and regression trees*. Routledge, 2017.

[12] Leo Breiman. "Random forests". In: *Machine learning* 45 (2001), pp. 5–32.

[13] Niklas Bussmann et al. "Explainable machine learning in credit risk management". In: *Computational Economics* 57 (2021), pp. 203–216.

[14] Jian Cao, Zhi Li, and Jian Li. "Financial time series forecasting model based on CEEMDAN and LSTM". In: *Physica A: Statistical mechanics and its applications* 519 (2019), pp. 127–139.

[15] Varun Chandola, Arindam Banerjee, and Vipin Kumar. "Anomaly detection: A survey". In: *ACM computing surveys (CSUR)* 41.3 (2009), pp. 1–58.

[16] Nitesh V Chawla et al. "SMOTE: synthetic minority over-sampling technique". In: *Journal of artificial intelligence research* 16 (2002), pp. 321–357.

[17] Hongbo Chen et al. "A Machine Learning Approach with Human-AI Collaboration for Automated Classification of Patient Safety Event Reports: Algorithm Development and Validation Study". In: *JMIR Human Factors* 11.1 (2024), e53378.

[18] Tianqi Chen and Carlos Guestrin. "Xgboost: A scalable tree boosting system". In: *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. 2016, pp. 785–794.

[19] Vinay Kumar Reddy Chimmula and Lei Zhang. "Time series forecasting of COVID-19 transmission in Canada using LSTM networks". In: *Chaos, solitons & fractals* 135 (2020), p. 109864.

[20] Eldan Cohen. "Interpretable Clustering via Soft Clustering Trees". In: *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*. Springer. 2023, pp. 281–298.

[21] Robert Dawson. “How significant is a boxplot outlier?” In: *Journal of Statistics Education* 19.2 (2011).

[22] Angus Dempster, François Petitjean, and Geoffrey I Webb. “ROCKET: exceptionally fast and accurate time series classification using random convolutional kernels”. In: *Data Mining and Knowledge Discovery* 34.5 (2020), pp. 1454–1495.

[23] Angus Dempster, Daniel F Schmidt, and Geoffrey I Webb. “Minirocket: A very fast (almost) deterministic transform for time series classification”. In: *Proceedings of the 27th ACM SIGKDD conference on knowledge discovery & data mining*. 2021, pp. 248–257.

[24] Alexander Dokumentov, Rob J Hyndman, et al. “STR: A seasonal-trend decomposition procedure based on regression”. In: *Monash econometrics and business statistics working papers* 13.15 (2015), pp. 2015–13.

[25] Claude Duchon and Robert Hale. *Time series analysis in meteorology and climatology: an introduction*. John Wiley & Sons, 2012.

[26] Shereen Elsayed et al. “Do we really need deep learning models for time series forecasting?” In: *arXiv preprint arXiv:2101.02118* (2021).

[27] Tolga Ergen and Suleyman Serdar Kozat. “Unsupervised anomaly detection with LSTM neural networks”. In: *IEEE transactions on neural networks and learning systems* 31.8 (2019), pp. 3127–3141.

[28] Yoav Freund and Robert E Schapire. “A desicion-theoretic generalization of on-line learning and an application to boosting”. In: *European conference on computational learning theory*. Springer. 1995, pp. 23–37.

[29] Jerome H Friedman. “Greedy function approximation: a gradient boosting machine”. In: *Annals of statistics* (2001), pp. 1189–1232.

[30] Jerome H Friedman and Jacqueline J Meulman. “Multiple additive regression trees with application in epidemiology”. In: *Statistics in medicine* 22.9 (2003), pp. 1365–1381.

[31] David John Gagne II et al. “Interpretable deep learning for spatial analysis of severe hailstorms”. In: *Monthly Weather Review* 147.8 (2019), pp. 2827–2845.

[32] João Gama et al. “From fault detection to anomaly explanation: A case study on predictive maintenance”. In: *Journal of Web Semantics* 81 (2024), p. 100821.

[33] Brandon M Greenwell, Annika Dahlmann, and Saurabh Dhoble. “Explainable Boosting Machines with Sparsity–Maintaining Explainability in High-Dimensional Settings”. In: *arXiv preprint arXiv:2311.07452* (2023).

[34] Shalmoli Gupta et al. “Local search methods for k-means with outliers”. In: *Proceedings of the VLDB Endowment* 10.7 (2017), pp. 757–768.

[35] James D Hamilton. “A new approach to the economic analysis of nonstationary time series and the business cycle”. In: *Econometrica: Journal of the econometric society* (1989), pp. 357–384.

[36] Muhammad Hafiz Hassan, AR Othman, and Shahrul Kamaruddin. “A review on the manufacturing defects of complex-shaped laminate in aircraft composite structures”. In: *The International Journal of Advanced Manufacturing Technology* 91 (2017), pp. 4081–4094.

[37] Trevor J Hastie. “Generalized additive models”. In: *Statistical models in S*. Routledge, 2017, pp. 249–307.

[38] Simon Haykin. *Neural networks: a comprehensive foundation*. Prentice Hall PTR, 1994.

[39] Kaiming He et al. “Deep residual learning for image recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[40] Nguyen Xuan Hoang et al. “Explainable anomaly detection for industrial control system cybersecurity”. In: *IFAC-PapersOnLine* 55.10 (2022), pp. 1183–1188.

[41] Sepp Hochreiter and Jürgen Schmidhuber. “Long short-term memory”. In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[42] Xuanxiang Huang and Joao Marques-Silva. “On the failings of Shapley values for explainability”. In: *International Journal of Approximate Reasoning* (2024), p. 109112.

[43] O I Vinyals and QV Sutskever Le. “Sequence to sequence learning with neural networks”. In: *Advances in neural information processing systems* (2014).

[44] Hassan Ismail Fawaz et al. “Inceptiontime: Finding alexnet for time series classification”. In: *Data Mining and Knowledge Discovery* 34.6 (2020), pp. 1936–1962.

[45] Klaus Kammerer et al. “Anomaly detections for manufacturing systems based on sensor data—insights into two challenging real-world production settings”. In: *Sensors* 19.24 (2019), p. 5370.

[46] Andrei Kapishnikov et al. “Guided integrated gradients: An adaptive path method for removing noise”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2021, pp. 5050–5058.

[47] Zahra Karevan and Johan AK Suykens. “Transductive LSTM for time-series prediction: An application to weather forecasting”. In: *Neural Networks* 125 (2020), pp. 1–9.

[48] Guolin Ke et al. “Lightgbm: A highly efficient gradient boosting decision tree”. In: *Advances in neural information processing systems* 30 (2017).

[49] Young Shin Kim et al. “Time series analysis for financial market meltdowns”. In: *Journal of Banking & Finance* 35.8 (2011), pp. 1879–1891.

[50] Gebhard Kirchgässner, Jürgen Wolters, and Uwe Hassler. *Introduction to modern time series analysis*. Springer Science & Business Media, 2012.

[51] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “Imagenet classification with deep convolutional neural networks”. In: *Advances in neural information processing systems* 25 (2012).

[52] Kwei-Herng Lai et al. “Revisiting time series outlier detection: Definitions and benchmarks”. In: *Thirty-fifth conference on neural information processing systems datasets and benchmarks track (round 1)*. 2021.

[53] Kenan Li et al. “W-TSS: A Wavelet-Based Algorithm for Discovering Time Series Shapelets”. In: *Sensors* 21.17 (2021), p. 5801.

[54] Youru Li et al. “EA-LSTM: Evolutionary attention-based LSTM for time series prediction”. In: *Knowledge-Based Systems* 181 (2019), p. 104785.

[55] Jason Lines et al. “A shapelet transform for time series classification”. In: *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2012, pp. 289–297.

[56] Datong Liu et al. “Fragment anomaly detection with prediction and statistical analysis for satellite telemetry”. In: *IEEE Access* 5 (2017), pp. 19269–19281.

[57] Jie Liu et al. “Semi-supervised anomaly detection with dual prototypes autoencoder for industrial surface inspection”. In: *Optics and Lasers in Engineering* 136 (2021), p. 106324.

[58] Qian Chu Liu et al. “The effect of manufacturing defects on the fatigue behaviour of Ti-6Al-4V specimens fabricated using selective laser melting”. In: *Advanced Materials Research* 891 (2014), pp. 1519–1524.

[59] Yong Liu et al. “Non-stationary transformers: Exploring the stationarity in time series forecasting”. In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 9881–9893.

[60] Scott M Lundberg, Gabriel G Erion, and Su-In Lee. “Consistent individualized feature attribution for tree ensembles”. In: *arXiv preprint arXiv:1802.03888* (2018).

[61] Scott M Lundberg and Su-In Lee. “A unified approach to interpreting model predictions”. In: *Advances in neural information processing systems* 30 (2017).

[62] Jiangang Ma et al. “Supervised anomaly detection in uncertain pseudoperiodic data streams”. In: *ACM Transactions on Internet Technology (TOIT)* 16.1 (2016), pp. 1–20.

[63] Manpreet Singh Minhas and John Zelek. “Semi-supervised anomaly detection using autoencoders”. In: *arXiv preprint arXiv:2001.03674* (2020).

[64] Prapanna Mondal, Labani Shit, and Saptarsi Goswami. “Study of effectiveness of time series modeling (ARIMA) in forecasting stock prices”. In: *International Journal of Computer Science, Engineering and Applications* 4.2 (2014), p. 13.

[65] Grégoire Montavon et al. “Layer-wise relevance propagation: an overview”. In: *Explainable AI: interpreting, explaining and visualizing deep learning* (2019), pp. 193–209.

[66] Meinard Müller. “Dynamic time warping”. In: *Information retrieval for music and motion* (2007), pp. 69–84.

[67] Harsha Nori et al. “Interpretml: A unified framework for machine learning interpretability”. In: *arXiv preprint arXiv:1909.09223* (2019).

[68] Karl Pearson. “Note on regression and inheritance in the case of two parents”. In: *Proceedings of the Royal Society of London* 58 (1895), pp. 240–242.

[69] Francesco Piccialli et al. “Artificial intelligence and healthcare: Forecasting of medical bookings through multi-source time-series fusion”. In: *Information Fusion* 74 (2021), pp. 1–16.

[70] Thanawin Rakthanmanon and Eamonn Keogh. “Fast shapelets: A scalable algorithm for discovering time series shapelets”. In: *proceedings of the 2013 SIAM International Conference on Data Mining*. SIAM. 2013, pp. 668–676.

[71] Roop Ranjan and AK Daniel. “CoBiAt: A Sentiment Classification Model Using Hybrid ConvNet-Dual-LSTM with Attention Techniques.” In: *Informatica (03505596)* 47.4 (2023).

[72] Robert W Resek. “Investment by manufacturing firms: A quarterly time series analysis of industry data”. In: *The Review of Economics and Statistics* (1966), pp. 322–333.

[73] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. “” Why should i trust you?” Explaining the predictions of any classifier”. In: *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. 2016, pp. 1135–1144.

[74] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. “Learning representations by back-propagating errors”. In: *nature* 323.6088 (1986), pp. 533–536.

[75] Jürgen Schmidhuber. “Deep learning in neural networks: An overview”. In: *Neural networks* 61 (2015), pp. 85–117.

[76] Pouya Shati, Eldan Cohen, and Sheila A McIlraith. “SAT-based optimal classification trees for non-binary data”. In: *Constraints* 28.2 (2023), pp. 166–202.

[77] Haoqiang Shi, Shaolin Hu, and Jiaxu Zhang. “LSTM based prediction algorithm and abnormal change detection for temperature in aerospace gyroscope shell”. In: *International Journal of Intelligent Computing and Cybernetics* 12.2 (2019), pp. 274–291.

[78] Shih Shun-Yao, Sun Fan-Keng, and Lee Hung-yi. "Temporal pattern attention for multivariate time series forecasting [J]". In: *Machine Learning* 108.8-9 (2019).

[79] Daniel Smilkov et al. "Smoothgrad: removing noise by adding noise". In: *arXiv preprint arXiv:1706.03825* (2017).

[80] Christian Szegedy et al. "Going deeper with convolutions". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 1–9.

[81] Christian Szegedy et al. "Inception-v4, inception-resnet and the impact of residual connections on learning". In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 31. 1. 2017.

[82] Chang Wei Tan et al. "Monash University, UEA, UCR time series extrinsic regression archive". In: *arXiv preprint arXiv:2006.10996* (2020).

[83] Chang Wei Tan et al. "MultiRocket: multiple pooling operators and transformations for fast and effective time series classification". In: *Data Mining and Knowledge Discovery* 36.5 (2022), pp. 1623–1646.

[84] Chang Wei Tan et al. "Time series extrinsic regression: Predicting numeric values from time series data". In: *Data Mining and Knowledge Discovery* 35 (2021), pp. 1032–1060.

[85] TT Teoh et al. "Anomaly detection in cyber security attacks on networks using MLP deep learning". In: *2018 International Conference on Smart Computing and Electronic Enterprise (ICSCEE)*. IEEE. 2018, pp. 1–5.

[86] Robert Tibshirani. "Regression shrinkage and selection via the lasso". In: *Journal of the Royal Statistical Society Series B: Statistical Methodology* 58.1 (1996), pp. 267–288.

[87] Anamika Tiwari, Vikrant Bansode, and Anurag S Rathore. "Application of advanced machine learning algorithms for anomaly detection and quantitative prediction in protein A chromatography". In: *Journal of Chromatography A* 1682 (2022), p. 463486.

[88] Ramazan Ünlü. "A comparative study of machine learning and deep learning for time series forecasting: a case study of choosing the best prediction model for Turkey electricity production". In: *Süleyman Demirel Üniversitesi Fen Bilimleri Enstitüsü Dergisi* 23.2 (2019), pp. 635–646.

[89] Nakul Upadhya and Eldan Cohen. "NeurCAM: Interpretable Neural Clustering via Additive Models". In: *arXiv preprint arXiv:2408.13361* (2024).

[90] Ashish Vaswani et al. "Attention is all you need". In: *Advances in neural information processing systems* 30 (2017).

[91] Frederik H. Vilshøj. *TorchLRP*. https://github.com/fhvilshoj/TorchLRP. GitHub repository. 2023.

[92] Zhiguang Wang, Weizhong Yan, and Tim Oates. "Time series classification from scratch with deep neural networks: A strong baseline". In: *2017 International joint conference on neural networks (IJCNN)*. IEEE. 2017, pp. 1578–1585.

[93] Haixu Wu et al. "Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting". In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 22419–22430.

[94] Lexiang Ye and Eamonn Keogh. "Time series shapelets: a novel technique that allows accurate, interpretable and fast classification". In: *Data mining and knowledge discovery* 22 (2011), pp. 149–182.

[95] Matthew D Zeiler and Rob Fergus. "Visualizing and understanding convolutional networks". In: *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part I 13*. Springer. 2014, pp. 818–833.

[96] Ailing Zeng et al. "Are transformers effective for time series forecasting?" In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 37. 9. 2023, pp. 11121–11128.

[97] Haoyi Zhou et al. "Informer: Beyond efficient transformer for long sequence time-series forecasting". In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 35. 12. 2021, pp. 11106–11115.

[98] Tian Zhou et al. "Fedformer: Frequency enhanced decomposed transformer for long-term series forecasting". In: *International Conference on Machine Learning*. PMLR. 2022, pp. 27268–27286.