mathematics

MDPI

Article

Fuzzy CNN Autoencoder for Unsupervised Anomaly Detection in Log Data

Oleg Gorokhov *, Mikhail Petrovskiy *, Igor Mashechkin and Maria Kazachuk

Computer Science Department, Lomonosov Moscow State University, MSU, Moscow 119234, Russia; mash@cs.msu.su (I.M.)

* Correspondence: gorokhov-oe@cs.msu.ru (O.G.); michael@cs.msu.ru (M.P.)

Abstract: Currently, the task of maintaining cybersecurity and reliability in various computer systems is relevant. This problem can be solved by detecting anomalies in the log data, which are represented as a stream of textual descriptions of events taking place. For these purposes, reduction to a One-class classification problem is used. Standard One-class classification methods do not achieve good results. Deep learning approaches are more effective. However, they are not robust to outliers and require a lot of computational effort. In this paper, we propose a new robust approach based on a convolutional autoencoder using fuzzy clustering. The proposed approach uses a parallel convolution operation to feature extraction, which makes it more efficient than the currently popular Transformer architecture. In the course of the experiments, the proposed approach showed the best results for both the cybersecurity and the reliability problems compared to existing approaches. It was also shown that the proposed approach is robust to outliers in the training set.

Keywords: machine learning; anomaly detection; convolutional neural networks; neural networks; fuzzy logic; deep learning; log mining; computer security; reliability

MSC: 68T07

Citation: Gorokhov, O.; Petrovskiy, M.; Mashechkin, I.; Kazachuk, M. Fuzzy CNN Autoencoder for Unsupervised Anomaly Detection in Log Data. *Mathematics* **2023**, *11*, 3995. https://doi.org/10.3390/math11183995

Academic Editors: Andrey Gorshenin, Mikhail Posypkin and Vladimir Titarev

Received: 8 August 2023
Revised: 7 September 2023
Accepted: 13 September 2023
Published: 20 September 2023

1. Introduction

1.1. Relevance of the Work

Nowadays, computer systems can play an important role in everyday social life. At the same time, the total number of subjects (subsystems, devices, users) interacting with a particular system is highly growing. Therefore, problems that arise during the operation of such systems can lead to significant overhead costs. Thus, the key moments of implementation of such systems are the requirements of cybersecurity and system reliability.

Fulfillment of these requirements can be achieved by anomalous events detection [1–3]. Information about these events is stored as a text stream in the system logs. However, the data volume can be very large, making it impossible to manually analyze events [4]. Also, in most of the existing works, the problem of supervised anomaly detection is solved. However, usually, we do not have access to a complete description of anomalous events, which makes it impossible to solve the problem of anomaly detection in the supervise mode [1].

As a rule, the existing works use classical approaches to unsupervised anomaly detection based on the reduction to a One-class classification problem, such as One-class SVM (Support Vector Machine) [5]. However, this approach allows building only simple data clusters without taking into account complex dependencies between events. These problems are corrected by a fuzzy logic approach using the Mahalanobis distance [6]. However, this approach does not take into account the order of events and their relationship with each other, which is important in the task of detecting anomalies in system logs. Taking into account the sequence of events can be performed by approaches based on

Bidirectional Recurrent Neural Networks [7]. These approaches take into account long-term dependencies between occurring events and build patterns of a system's normal behavior, predicting the next event based on existing patterns. If the sequence of events does not correspond to the pattern of normal behavior, we consider such sequence as anomalous. However, the use of target event prediction alone does not allow to encode general patterns common to all normal sequences, which leads to the fact that the quality of RNN (Recurrent Neural Network)-based algorithms is not high enough. This problem is solved by approaches based on the Transformer architecture [8]. However, they use a computationally complex self-attention mechanism that does not allow such algorithms to be applied in real life.

As a result, this paper proposes an automated unsupervised method for detecting anomalies in the text content of system log data based on the parallel convolutional asymmetric autoencoder and the fuzzy clustering, which allows taking into account the advantages of the existing methods and eliminating their disadvantages.

1.2. Goal of the Work

Based on the above statements, we can formulate the main goal of this work: ***research and development of an automated method for unsupervised anomaly detection in the textual content of log data for computer systems security and reliability tasks.***

1.3. Related Work

An analysis of existing works in the area under consideration [1,4,9,10] allows us the identification of the following steps of anomaly detection process in the log data:

1. **Log collection.** Software systems constantly write information about ongoing events to special logs, which are descriptions stored either in a database or in files. Such information is represented as semi-structured data streams suitable for further analysis.
2. **Log parsing.** After log collection, the data are converted to a text format, which makes them suitable for applying text mining methods.
3. **Feature extraction.** At this stage, text data modeling algorithms are applied.
4. **Anomaly detection.** Once the data model is built, various machine learning models can be trained and applied to solve the considered problem.

A detailed description of each stage based on the analysis of the existing works is presented below.

1.3.1. Nature of the Input Data

Before describing approaches to individual subproblem solutions, it is necessary to pay attention to the nature of the input data that must be taken into account in the existing solutions [4,9,11–13]:

1. The input is a stream of semi-structured descriptions of related (interconnected) events;
2. The volume of the input data can be very large (more than 50 GB per hour);
3. Information about one event can be duplicated several times in a row;
4. The format of event descriptions is highly dependent on the particular system;
5. Often, the event descriptions are almost similar to each other and differ only in some minimal factors (for example, identifiers of related subsystems and processes);
6. A set of patterns can be extracted from similar event descriptions. The number of such patterns is usually small (up to 1000).

1.3.2. Log Collection

In terms of log collection, systems typically record event information in a semi-structured form in files or databases [1]. This information is usually a stream of records containing a timestamp, possible event parameters (type, associated system, etc.), and some

textual description. Due to the nature of the input data described above, it is possible to use data filtering to reduce the data volume.

1.3.3. Log Parsing

As mentioned above, the collected data are streams of semi-structured data that can be represented as text. To take into account the nature of the input data (large data volume, a combination of similar events), it is necessary to parse text data [13,14]. This operation includes both data preprocessing and event pattern creation. For these purposes, text mining methods are used. However, as mentioned above, the format of event descriptions depends on the particular target system. Therefore, to develop a universal approach, an approach based on the use of regular expressions is effective [1,4]. In the parsing process, we obtain a dictionary of event patterns (individual events), and for each event, we specify the corresponding pattern and its parameters (for example, time and used IDs). For example, the event *"2023-02-02 20:55:54 INFO Data read by system No. 123456"* can be represented by the pattern *"!TIME! INFO Data read by system No. !ID!"* with parameter values *!TIME! = "2023-02-02 20:55:54"*, *!ID! = "123456"*.

1.3.4. Feature Extraction

After parsing and receiving individual separate events, it is possible to extract features that describe a data model. To achieve this, machine learning methods are applied [1,4].

First of all, the individual events are encoded based on the dictionary (One-hot encoding; event description models are represented as text models of either a set or a sequence of tokens) [15,16].

Second, the events are grouped into blocks (of a fixed size or by the time of the event occurrence), and a block embedding is built. An embedding is a function (a model) that transforms input objects (in our case, these are event blocks) into some real vectors. An illustration of the object embedding is shown in Figure 1.

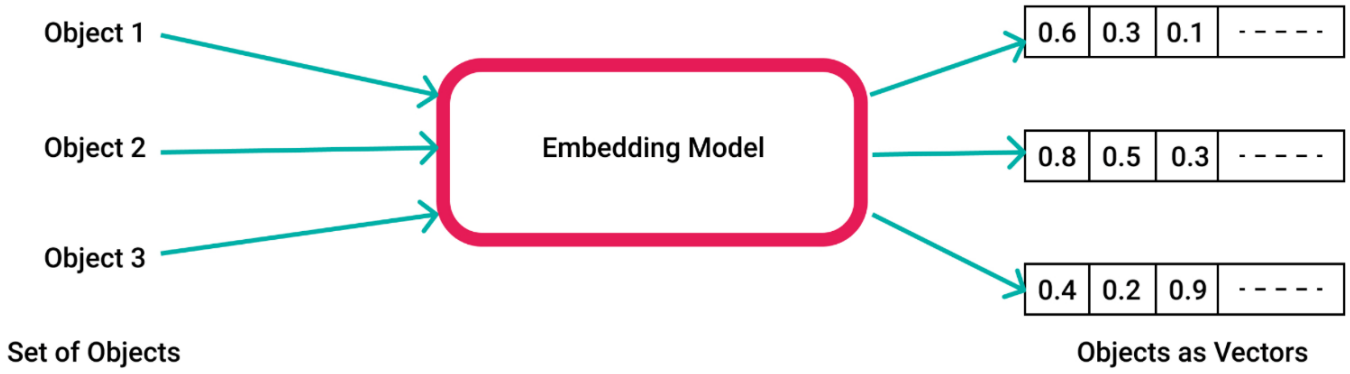

Figure 1. Illustration of the vector embedding. The Figure is taken from [17].

Therefore, this embedding can be considered as a data model. At this stage, approaches based on reducing the dimension of the feature space (NMF—Non-negative Matrix Factorization, LDA—Linear Discriminant Analysis, etc.) are widely used [16]. Recently, approaches based on deep learning (convolutional neural networks, Transformer architectures using the self-attention mechanism) have also been used [11,16,18,19].

1.3.5. Anomaly Detection

After the data model is built, it is necessary to make and train a model that describes the system normal behavior. It should be noted that an anomalous behavior description criterion is needed. In the most practical problems, it is very difficult to correctly describe the full anomalous behavior of systems in general. However, it is possible to collect information about the normal behavior of the system.

Therefore, at this stage, the classical approach of reducing the problem to a One-class classification task is used [9,10].

The most popular method used as a baseline for this purposes is the One-class SVM [5,9]. More efficient is the method based on fuzzy clustering [6,20,21]. Currently, methods based on deep learning are also developing, and they show good results. In particular, Transformer-type neural architectures are widely used [8,16,18].

A more detailed description of these methods is given below.

One-class SVM [5,9,22,23] is a modification of the classical support vector machine method, allowing the building of a separating hyperplane for the training sample objects which are implicitly mapped to a higher-dimensional space H using the kernel function $\Phi : \mathbb{R}^N \to H$. Thus, the optimization problem is solved in this method as can be seen in (1).

$$\min_{w,\rho,\xi_i} \left[\frac{1}{2} \cdot ||w||^2 + \frac{1}{vl} \sum_{i=1}^{l} \xi_i - \rho \right], (w \cdot \Phi(x_i)) \geq \rho - \xi_i; i = \overline{1,l}; \xi_i \geq 0, \tag{1}$$

where each $x_i \in \mathbb{R}$ is a feature vector, $v \in (0,1)$ is the percent of outliers, ρ is the hyperplane shift, l is the size of the training set, ξ_i are penalty variables.

Then, if w and ρ are the solution of the stated optimization problem, $f(x)$ is a decision function as can be seen in (2).

$$f(x) = sign(w \cdot \Phi(x) - \rho). \tag{2}$$

One-class SVM is often used for anomaly detection for various data types (including log data), but uses only a part of the observations (located on the boundaries of the hyperplane), which does not allow taking into account all dependencies in the data. In addition, this approach does not calculate the anomality degree of an individual feature vector. These problems are solved by methods based on fuzzy clustering described below.

Note that the complexity of the classical One-class SVM method grows quadratically with the number of records in data [22]. Therefore, it does not allow efficient processing of large-volume logs. Therefore, iterative SVM [22] and deep learning SVM [23] approaches are proposed.

Fuzzy clustering methods [20,21] build a fuzzy cluster after mapping the features into a higher-dimensional space H with the kernel function $\Phi : \mathbb{R}^N \to H$.

In this case, for each object x_i of the training sample, the degree of typicality u_i is determined according to the following rule [6] described in (3).

$$u_i = \left[1 + \left(\frac{D_i(a)}{\mu} \right)^{\frac{1}{m-1}} \right]^{-1}. \tag{3}$$

Here, μ is the cluster radius where feature vectors have the degree of typicality equal to 0.5. It is calculated based on the distance from each object in the training sample to the center of the cluster. $m > 1$ is a hyperparameter, a degree of fuzziness. $D_i(a)$ is the distance from the feature vector $\Phi(x_i)$ to the constructed center a of the cluster.

Based on the existing works [6], to construct clusters of a more complex shape, it is necessary to use the Mahalanobis distance [24]. $D_i(a)$ is calculated as can be seen in (4).

$$D_i(a) = ||\Phi(x_i) - a||_C^2 = (\Phi(x_i) - a)^T \cdot C^{-1} \cdot (\Phi(x_i) - a), \tag{4}$$

where C is the cluster covariance matrix.

We let $\{x_i \in \mathbb{R}^N | i = 1, \ldots, N\}$ be the final sample of training data. N is the number of samples. Therefore, the task of constructing the fuzzy cluster is reduced to the optimization problem as can be seen in (5).

$$\min_{U,a,\mu} E(U,a,\mu) = \sum_{i=1}^{N} u_i^m \cdot \|\Phi(x_i) - a\|_C^2 + \mu \cdot \sum_{i=1}^{N} (1 - u_i)^m, \tag{5}$$

where U is the set of degrees of typicality of feature vectors.

This approach builds a more accurate cluster for normal data, and also calculates the degree of typicality of each feature vector. However, fuzzy clustering does not take into account the relationship of individual events, which is a key moment in the log mining problems. Therefore, the methods based on the Transformer architecture, which are described below, are currently more efficient.

Deep learning methods [8,16,18,19] are divided into two main groups of approaches: Transformers and convolutional neural networks. While the first approach uses the multi-headed attention mechanism, the second one implements specific feature extraction using convolution with a special filter.

Transformers [8,16,18] are autoencoders that map input features $x = (x_1, \ldots, x_n)$ to a new representation $z = (z_1, \ldots, z_n)$. Then, the decoder performs the inverse transformation of the constructed features z into the output data $y = (y_1, \ldots, y_n)$ from the original space. At every step, the model uses recurrent neural networks [25].

Within this approach, the problem of masked language modeling is solved, which consists of skipping individual events and their predictions based on the constructed embedding for all other events in the block [16,18].

As for the encoder, a multi-head self-attention mechanism is used. Each head h_t is a result of attention A and is described in (6).

$$\begin{aligned} h_t &= A(X^j W_l^Q, X^j W_l^K, X^j W_l^V), \\ A(Q, K, V) &= softmax\left(\frac{QK^T}{\sqrt{d_v}}\right)V, \end{aligned} \tag{6}$$

where $X^j \in \mathbb{R}^{T \cdot d}$ is the embedding for a block of event descriptions with the size $T \cdot d$; W_l^Q, W_l^K and W_l^V are linear mapping weight matrices of dimension $\mathbb{R}^{d \cdot d_v}$ for head number T; d_v is a size of one head. To reduce the complexity of calculations, it is proposed to use the scaled dot-product attention [16,18].

The multi-headed self-attention mechanism performs a concatenation of all heads $(h_1, \ldots, h_H)$ and a linear mapping into some new feature space. Then, to predict the skipped event, a fully connected layer T with an activation function $ReLU$ is used. The classic Transformer architecture consists of several such layers T and builds an embedding for a single event x_t^j.

To solve the problem of anomaly detection in event blocks, two tasks are independently solved: masked modeling of missed events and building a cluster of normal events [8].

However, the considered approaches have a number of disadvantages. First of all, most of the approaches are used within the framework of binary classification [15], which does not allow them correct application within the framework of the considered task. Second, the Transformer architecture uses the recurrent neural networks and the attention mechanism that is computed at every training step to predict the embedding of every skipped event, which leads to significant computational overheads [16]. Therefore, an alternative approach based on building embeddings using convolutional neural networks is more effective.

Convolutional neural networks (CNNs) [16,19,26,27] were originally used for text classification and are based on feature extraction operation during convolution with a given filter. The best results were obtained using a parallel convolution [26]. This approach was used by the authors of this paper and adapted to detect anomalies in texts [19]. The essence of the method is as follows.

We let D be a text document embedded with a model of token (word, letter etc.) sequences. Therefore, $D = x_{1:N} = x_1 \bigoplus \ldots \bigoplus x_N$. Here, N is the amount of terms in the document; $x_i = (x_i^1, x_i^2, \ldots, x_i^M) \in \mathbb{R}^M$ is a token embedding with fixed size $M \in \mathbb{N}$; $\bigoplus$ is a concatenation operator; in general, $x_{i:j} = x_i \bigoplus x_{i+1} \bigoplus \ldots \bigoplus x_j$ is a subsequence of document D limited by the term indexes i and j.

CNNs are based on two main operations: the convolution and the pooling. We let H be a subsequence (window) size, a hyperparameter. Therefore, the convolution applies some linear transformation to each window using a filter $w \in \mathbb{R}^{H,M}$ and a bias $b \in \mathbb{R}$, and then applies some non-linear activation function f to the result as described in (7).

$$z_i = f(w \cdot x_{i:i+H-1} + b). \tag{7}$$

Then, the filter is applied to all existing windows of size H in D. So, the feature map $\hat{z} = (z_1, z_2, \ldots, z_{N-H+1})$ is made.

After convolution, the pooling is performing. Therefore, for one filter, the most important feature $z_{H,1}$ from $\hat{z}$ is selected. By varying the filters randomly, several features for one filter size are formed, $z_H = (z_{H,1}, z_{H,2}, \ldots, z_{H,K}) \in \mathbb{R}^K$, where K is the amount of different filter sizes H. The parallel convolution uses the concatenation of independent such convolutional layers using filters of different sizes. Therefore, after it, the concatenated feature vector $z \in \mathbb{R}^S$ is formed. S is a total size of features from all convolution layers: $S = K_1 + K_2 + \ldots + K_F$, where K_i is a filter amount for one filter size with index i, F is a number of independent convolutional layers.

To apply CNNs to the problem of anomaly detection, the authors propose the use of a Dropout and a radial basis function as the output layer of the network [19] as can be seen in (8).

$$f_{out}(x) = \exp^{-||z'-c||^2}, \tag{8}$$

where z' is the feature vector z after applying the Dropout on the last layer, c is the center of normal data distribution, a fitted parameter; $||z' - c||$ is the Euclidean distance from z' to c.

To avoid overfitting, using $l2$-regularization is proposed.

The loss function is described in (9).

$$\mathcal{L} = \mathcal{L}_{clust} + \mathcal{L}_{l2}, \tag{9}$$

where $\mathcal{L}_{l2}$ is the error for $l2$-regularization, $\mathcal{L}_{clust}$ is the clustering loss function computed by (10).

$$\mathcal{L}_{clust} = -log(f_{out}(x)). \tag{10}$$

This approach builds a distribution center of the convolutional features and helps to achieve good results in text anomaly detection more effectively than the Transformer method. However, it allows the building of a cluster of only a simple form. Also, the minimum cluster size is not limited, which can lead to trivial solutions and bad separations of normal and anomalous data.

In addition, this approach uses *ReLU* as an activation function of the convolutional layer, which leads to the non-normalized features, degrading the quality of the final classification. For modification, the self-normalizing neural networks proposed in [28] are used. But this architecture is considered within the framework of the image classification problem.

For clustering, the authors use an exponential RBF (Radial Basis Function), which also affects the quality of the algorithm. Paper [29] provides an overview of alternative activation functions and methods to implement them in the existing architectures. However, this approach is also used for image classification.

Other approaches to anomaly detection are also considered in the existing works [7,30–35].

In particular, paper [30] describes an approach based on the Principal Component Analysis (PCA). It can be represented as a linear autoencoder that builds a feature description of the event block, and then minimizes information loss during the compression stage using a linear decoder.

Paper [31] introduces the Isolation Forest (iForest) method which is an unsupervised anomaly detection algorithm that represents features as tree structures.

Paper [32] describes a cluster approach for detecting log anomalies.

The DeepLog method is proposed in paper [7]. This method is a recurrent neural network adaptation to detect anomaly patterns in the log data.

Paper [33] describes LogAnomaly, a unified log anomaly detection system that can detect sequential and quantitative log anomalies.

However, article [8] provides a detailed assessment of the considered approaches, as a result of which the authors conclude that these methods do not allow the achievement of high classification quality compared to the Transformer-based LogBERT (Bidirectional Encoder Representations from Transformers) approach proposed in the article.

Also, paper [34] describes the approach based on GAN (Generative Adversarial Networks) architecture. And the VAE (Variational Autoencoder) is considered in [35]. However, based on the performed local experiments, it can be concluded that they do not allow the achievement of good results in the considered task.

2. Materials and Methods

In this paper, we propose **FuzzyCNN**, a new approach to log anomaly detection, which combines the advantages of convolutional neural networks and fuzzy clustering methods described above and eliminates all the disadvantages of the methods described above.

The proposed approach is an autoencoder method that uses parallel convolution as an encoder and a simple decoder to minimize information loss during encoding. For anomaly detection, a modified layer with a radial basis function is used; it is based on fuzzy clustering using the Mahalanobis distance.

An asymmetric autoencoder with parallel convolution and a fully connected decoding layer generate features taking into account the order of events while minimizing encoding losses. At the same time, the calculation is performed more efficiently than in the case of the self-attention mechanism in Transformers.

The fuzzy clustering layer limits the minimum radius of the normal data cluster, which makes it robust (resistant to outliers in the training sample). Also, the Mahalanobis distance is used, which builds elliptical clusters and tracks more complex interconnections in the data.

The proposed approach uses self-normalization methods [28] to make the solution more robust.

The full architecture of the proposed approach is shown in Figure 2.

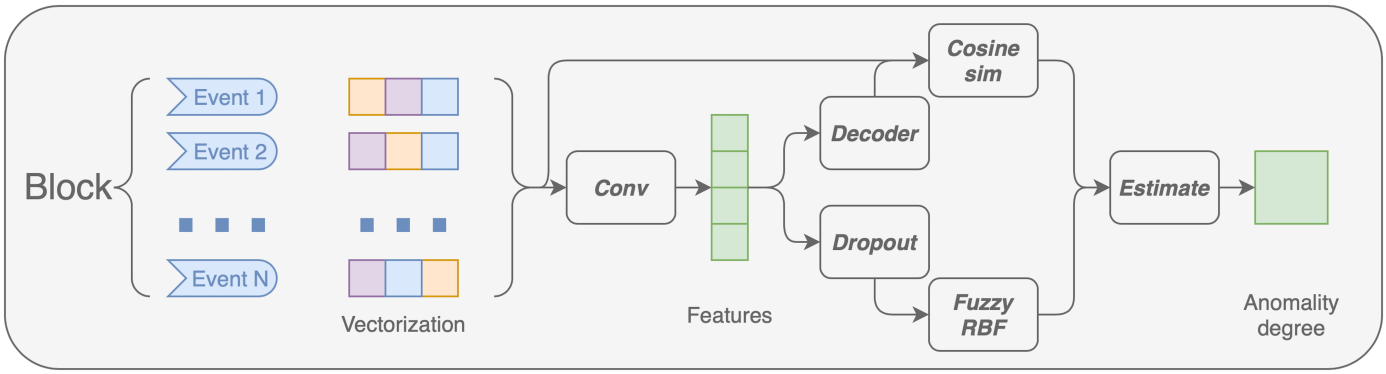

Figure 2. Architecture of the proposed approach. Each event in the block is encoded using One-hot encoding and event pattern dictionary. Thus, each block is One-hot representation of the event. Then, the parallel convolutional block generates a feature map, which is transferred separately to the decoder and fuzzy RBF layers. At the decoder stage, the cosine similarity is computed to minimize the information loss during the parallel convolutional encoding. In the Fuzzy RBF, we cluster data to a fuzzy cluster. At the final stage, we accumulate the decomposition error of the decoder and normality degree from the fuzzy RBF and obtain the final anomality score of the event block.

2.1. Log Collection

As for data collection, we use files generated by various systems. These files contain text descriptions of events as a sequence of lines. Each line has the following format: "!TIME! !ATTR! !TEXT DESCRIPTION!". !TIME! is a timestamp for an individual event;

!ATTR! is some structural event information (event type, connected system etc.), and it can be used for further analysis; !TEXT DESCRIPTION! is some unstructured event description represented as text.

2.2. Log Parsing

At the log parsing stage, a regular expression is used. It highlights the text description and attributes. To implement a universal solution, the authors single out only the identifiers of related subsystems as the attributes, while the rest of the information refers to text description.

For example, let us look at the event record: "2023.02.02 16:01:11 INFO dfs.DataNode $ PacketResponder: PacketResponder 2 for block blk_123 terminating". Here, the !TIME! is a string "2023.02.02 16:01:11"; !ATTRIBUTES! are "2" and "blk_123"; !TEXT DESCRIPTION! is a pattern: "!TIME! INFO dfs.DataNode$PacketResponder: PacketResponder !ID1! for block !ID2! terminating" with !TIME!, !ID1! and !ID2! from !ATTR!.

2.3. Data Vectorization and Grouping

At this stage, textual information from !TEXT DESCRIPTION! is converted into a structured form. For this, approaches based on the text mining are used [16,19]. The analysis of the existing works and datasets showed that a simple One-hot encoding can be used for these purposes, because the volume of the dictionary of unique text descriptions is small (up to 1000 events) [16,19,30,36].

Also, as a vectorization algorithm, we can use random encoding as well as embedding models (Doc2Vec, GloVe) [16,26]. They are considered at the stage of experimental evaluation of the algorithm.

Thus, after vectorization, a dictionary of unique event descriptions is formed. For each event, a structured description is built, including a One-hot embedding in accordance with the index in the dictionary, to which a list of attributes obtained as a result of parsing is added. Also, it is possible to use random event encoding [16,19,26], Doc2Vec and GloVe models [16].

Therefore, each event is represented as a vector $x_i = (x_i^1, x_i^2, \ldots, x_i^M) \in \mathbb{R}^M$ with an additional attribute vector $\hat{x}_i = (\hat{x}_i{}^1, \hat{x}_i{}^2, \ldots, \hat{x}_i{}^{M_1}) \in \hat{\mathbb{X}}^{M_1}$ that is used for grouping. Here, M is the dictionary size; M_1 is the attributes amount; $\hat{\mathbb{X}}$ is the attribute value set. The authors use the union as a set of attributes: $\hat{\mathbb{X}} = \mathbb{R} \cup \mathbb{T}$. $\mathbb{R}$ is used for ID numeric description while $\mathbb{T}$ is the set of strings describing the timestamp in the given format.

Further, for each event, a set of related attributes is considered, and the events are combined into blocks by grouping based on this set. The authors use either grouping by the number of events, or grouping by identifiers, followed by filtering by the number of events to obtain blocks of a fixed size.

After vectorization and grouping, all attributes are discarded and the input data are represented as a sequence of blocks $D = (D_1, D_2, \ldots, D_K)$ containing descriptions of the grouped events. Here, K is the size of D; each $D_i = (x_1, x_2, \ldots, x_N) \in \mathbb{R}^{N,M}$ is the block embedding matrix; $x_i \in \mathbb{R}^M$ is the event embedding; N is a fixed event amount for each block; M is a fixed size of unique event dictionary.

2.4. Convolution Neural Networks for Feature Extraction

At this stage, it is proposed to use the approach based on the operation of parallel convolution [19]. The classic solution described above applies to text data. This paper proposes its modification for the log mining.

Separate event blocks are used as text documents of event descriptions that can be considered as tokens. For convolution, three parallel convolution layers with different filter sizes are used. The architecture of the convolution block is shown in Figure 3.

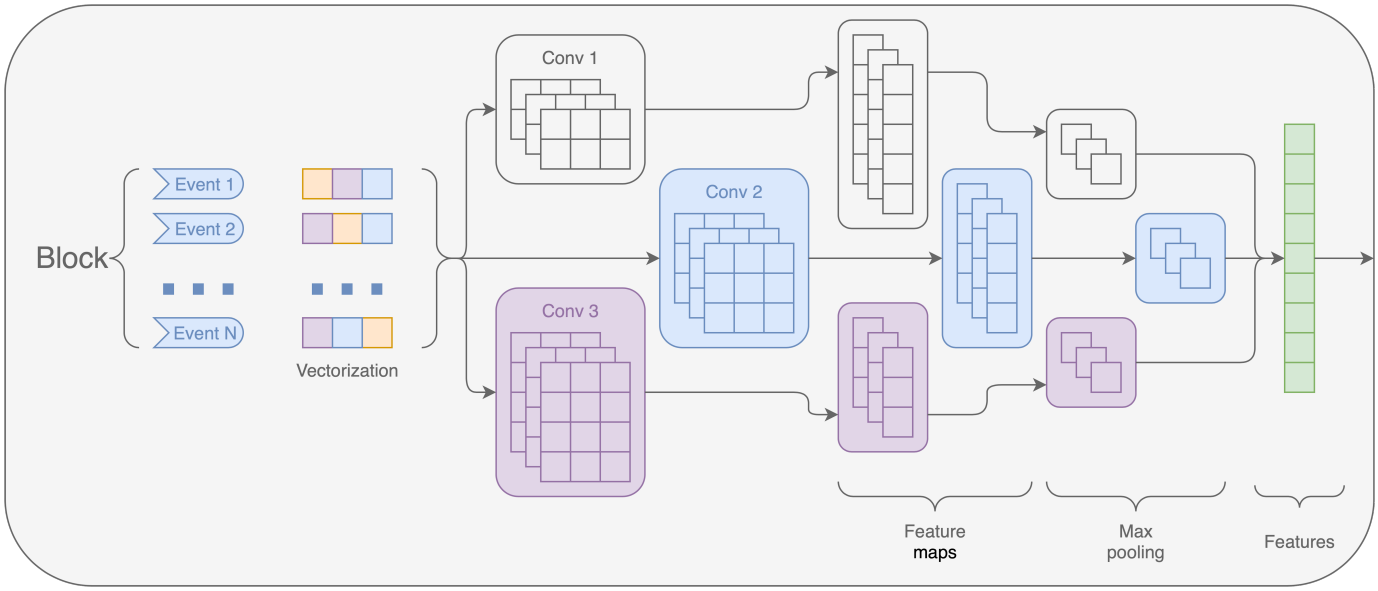

Figure 3. Architecture of the convolution block. The block embedding matrix is transferred to each convolutional layer separately. Max pooling layer also follows each convolutional layer. After that, the outputs of the each pair of layers (convolution, max-pooling) are concatenated to a feature map. This feature map can be considered as a target event block embedding vector.

As in the base approach, for each convolutional layer, we apply the transformation described in (7). To normalize the features, the authors propose to use *SELU* [28] as an activation function that is a modification of the *ReLU* described in (11).

$$SELU(x) = \lambda \cdot \begin{cases} x, & \text{if } x > 0, \\ \alpha \cdot e^x - \alpha, & \text{if } x \leq 0. \end{cases} \tag{11}$$

For the correct work of the convolution, the authors propose to use the LeCun initializer, which is also a part of the self-normalizing networks [28].

Thus, during the transformation by each filter w with the size H for each convolutional layer, a single feature $\hat{z}_i$ for an event window $x_{i:i+H-1}$ is formed as can be seen in (12).

$$\hat{z}_i = SELU(w \cdot x_{i:i+H-1} + b), \tag{12}$$

where $w \in \mathbb{R}^{H,M}$ is a filter and $b \in \mathbb{R}$ is a bias for the current filter. Thus, in the process of applying a filter of a certain size to one subsequence of events of length H, we actually combine information about events and their relationships within the subsequence, generating a single value for each such subsequence—an informative description (embedding) of the subsequence. Next, we apply the filter to all possible nearby subsequences of length H, resulting in an entire feature vector. Further, in the process of pulling, we extract the most significant information, leaving only the maximum feature.

Theoretically, the proposed function *SELU* provides more stable features. Also, experimental estimation of these function is performed. We compare *SELU* and the *ReLU* (it is widely used at convolutional layers in the existing works [19,26]) as an activation function of each convolutional layer. These experiments are described below.

By setting hyperparameters $F \in \mathbb{N}$ (number of independent convolution layers), $K \in \mathbb{N}$ (number of filters per a layer), and $H = (H_1, H_2, \ldots, H_F)$ (a filter size array, H_i is a filter size for layer numbered i), we can build the final parallel convolution architecture. By combining all the features from all layers, filters and windows, we obtain the final feature map z for a block of events, as can be seen in (13).

$$z = (z_1, z_2, \ldots, z_S) \in \mathbb{R}^S. \tag{13}$$

Here, $S \in \mathbb{N}$ is the total feature amount and it is described by (14).

$$S = K \cdot \sum_{i=1}^{F} (N - H_i + 1). \tag{14}$$

Taking into account the requirement that the convolution is performed correctly (the size of each filter should not exceed the total size of the event block), this expression can be rewritten as can be seen in (15) and (16).

$$S = K \cdot \left(N \cdot F + F - \left(\sum_{i=1}^{F} H_i \right) \right), \tag{15}$$

$$\forall i = 1, \ldots, F \implies H_i \leq N. \tag{16}$$

2.5. Asymmetric Decoder to Minimize Information Loss When Extracting Features

To minimize the information loss during convolution, the authors propose to use a decoder that restores the original block event description from convolution features while calculating the reconstruction error. For these purposes, a cosine similarity is used because it shows good results in the complex structured data analysis [16].

The performed experiments show that an ordinary fully connected layer demonstrates good results. Thus, as a result of the work of the decoder for each jth block, an estimate q_j of the compression quality is formed as can be seen in (17).

$$\begin{aligned} q_j &= \cos(D_i^f, \hat{D}_i), \\ D_i^f &= x_1^i \oplus x_2^i \oplus \ldots \oplus x_N^i, \\ \hat{D}_i &= ReLU(z \cdot w_d + b_d). \end{aligned} \tag{17}$$

Here, $D_i^f \in \mathbb{R}^{M \cdot N}$ is a flattened representation of block $D_i = (x_1^i, x_2^i, \ldots, x_N^i)$; $\hat{D}_i \in \mathbb{R}^{M \cdot N}$ is the flattened block D_i representation reconstructed by the decoder; z is the feature map described in (13); $w_d \in \mathbb{R}^{S, M \cdot N}$ is the weight matrix and b_d is the bias for the linear mapping in the decoder; *ReLU* is the decoder activation function. The choice of one layer of the decoder and the use of *ReLU* as the activation function is based on the experiments carried out, which are described below.

Also, $\oplus$ is a concatenation operator and *cos* is a cosine such as $\forall X, Y \in \mathbb{R}^M : X = (X_1, \ldots, X_M), Y = (Y_1, \ldots, Y_M) \implies$ (18).

$$\begin{aligned} \cos(X, Y) &= \frac{X \cdot Y}{\|X\| \|Y\|} = \frac{\sum_{i=1}^{M} X_i \cdot Y_i}{\sum_{i=1}^{M} X_i^2 \cdot \sum_{i=1}^{M} Y_i^2}, \\ X \oplus Y &= (X_1, \ldots, X_M, Y_1, \ldots, Y_M) \in \mathbb{R}^{2 \cdot M}. \end{aligned} \tag{18}$$

At the same time, the closer the value of the cosine similarity is to 1, the greater the accuracy of the performed reconstruction. Thus, q_j needs to be maximized for each event block during the training process. This approach based on the cosine measure of similarity is widely used in the problems of constructing an informative vector representation of text data [16]. It is based on a vector data model and the assumption that similar texts will be described by similar vectors. Vectors are considered similar if the angle between them is sufficiently small, i.e., the cosine is close to 1. The lengths of vectors are not taken into account since, as a rule, models for constructing an informative vector representation use normalized vectors. In our task, each block is represented by a sequence of text descriptions of events, the vector representation of which is also normalized. Therefore, this approach can be used in our work. An illustration of this method is shown in Figure 4.

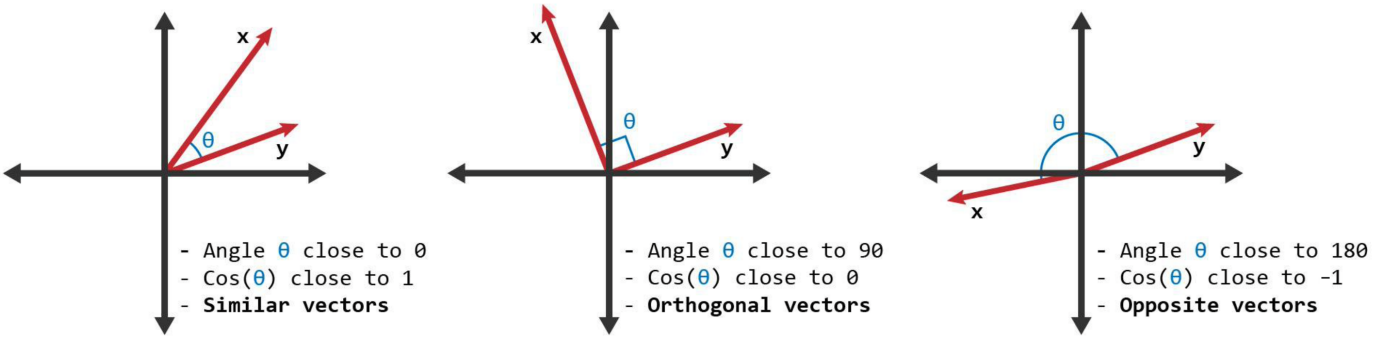

Figure 4. Illustration of the cosine similarity. The Figure is taken from [37].

When training in a One-class mode, the autoencoder builds embeddings for normal data more correctly, while for anomalous data, the cosine value is close to 0.

However, this solution is not stable against outliers and requires an additional robust layer to solve anomaly detection problems.

2.6. Fuzzy Clustering and Anomaly Detection

To solve the problem of anomaly detection, the authors propose to consider a modification of the fuzzy clustering method for the constructed convolutional features.

Based on the existing work [6], we propose our own network layer that determines the distribution of normal data by introducing a distribution center as a trainable parameter, as well as a positive definite covariance matrix to calculate the Mahalanobis distance. Also, additionally trainable is the μ parameter that controls the degree of fuzziness. Thus, in this layer, typicality degree u_j is calculated for a block numbered j with feature map z as can be seen in (19) and (20).

$$u_j = \left[1 + \left(\frac{d_j(a)}{\mu}\right)^{\frac{1}{m-1}}\right]^{-1}, \tag{19}$$

$$d_j(a) = \|\Phi(z^{(j)}) - a\|_C^2 = (\Phi(z^{(j)}) - a)^T \cdot C^{-1} \cdot (\Phi(z^{(j)}) - a). \tag{20}$$

Here, d_j is the Mahalanobis distance; u_j is the degree of typicality; Φ is a kernel function for transformation of the feature space into a higher-dimensional space H, $\Phi : \mathbb{R}^S \to H$; $a \in H$ is the center of a normal data cluster; C is a covariance matrix; μ is the cluster radius mentioned in (3); a, $C > 0$ and $\mu > 0$ are trainable parameters (to achieve $\mu > 0$, we can assume $\mu = \hat{\mu}^2 + \epsilon$ where $\hat{\mu} \in \mathbb{R}$ is a simple trainable parameter without any constraints, $\epsilon \in \mathbb{R}$ is a small constant; for example, we can use $\epsilon = 10^{-3}$; as for $C > 0$—a covariance matrix—we can use a diagonal matrix with non-negative values, so we can fit vector $c \in H$ of positive real values, each of which can also be represented as the sum of the square of the actually trained parameter and some small number ϵ). In the course of the experiments carried out below, it is necessary to consider various kernel functions Φ considered in papers [6,29]. Also, it is necessary to experimentally compare the Mahalanobis distance and other metrics. And, finally, we need to define the initial values for trainable parameters a, $C > 0$ and $\mu > 0$.

As mentioned above, the Mahalanobis distance allows a more accurate determination of the distribution of normal data, determining not only the center of the distribution, but also the covariance matrix. Due to this, it is possible to build elliptical clusters. An example of clustering using Euclidean distance and Mahalanobis distance is shown in Figure 5.

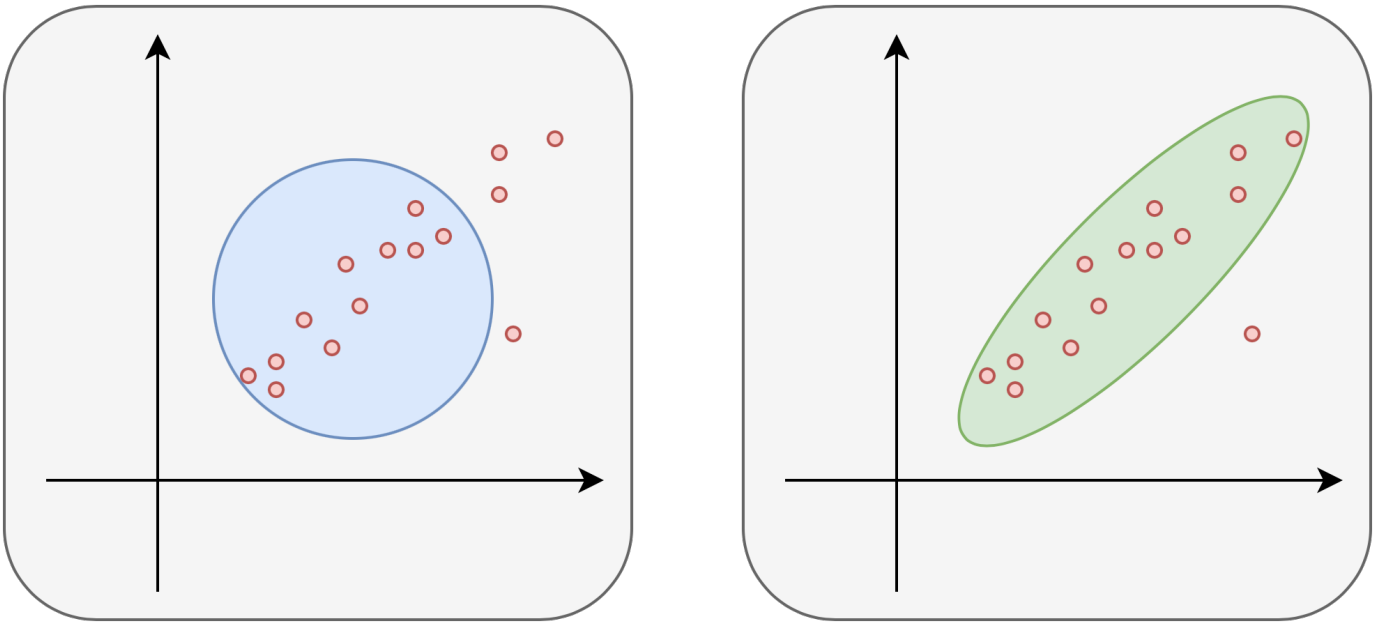

Figure 5. Illustration of the Euclidean-distance- (on the **left**) and the Mahalanobis-distance (on the **right**)-based clustering. The Euclidean distance allows to build only a round cluster (marked in blue) while the Mahalanobis distance allows building a more complex elliptical cluster (marked in green).

After this stage, $u_j \in (0,1)$ is formed. It is the degree of typicality for the block numbered j. Therefore, the larger this value, the more typical the object for the training set.

Thus, we can combine this value with the value of the cosine measure of similarity, obtaining the final estimate of the normality of the data y_j as can be seen in (21).

$$y_j = q_j \cdot u_j. \tag{21}$$

This value reaches its maximum at $q_j = 1, u_j = 1$ when the data are normal and correctly transformed by the convolution. Therefore, it is necessary to maximize y_j in the network training process.

Thus, when training the proposed solution in the One-class classification mode, we build a model of the system normal behavior that describes the patterns of normal behavior based on the training data. At the same time, we try to bring the normality estimate y_j for each block of events from the training sample closer to the value equal to 1. When applying the model to new data, we obtain a single number at the output of the neural network—the normality score of this block. This score shows the ways in which the given block of events corresponds to the normal system behavior model. The higher these values, the more typical the block.

In order to accurately determine whether a block is normal or abnormal, it is necessary to compare this estimate with some threshold, which is determined by the specific dataset being used. The problem of choosing the threshold is a separate problem of the proposed approach evaluation and is not considered in the current paper.

2.7. Regularization

To avoid overfitting, it is proposed to use n-regularization, as well as Dropout in the input of the fuzzy layer. At the same time, to improve the stability of the self-normalizing part, the authors use the AlphaDropout [28].

2.8. Training

At the network training stage, the minimization problem is solved that is described in (22)–(24).

$$\min_{w,b,a,C,\mu} \mathcal{L}(w,b,a,C,\mu), \tag{22}$$

$$\mathcal{L}(w,b,a,C,\mu) = \sum_{i=1}^{F} \left(\mathcal{L}_{l2}^{i}\right) + \mathcal{L}_{fuzzy} + \mathcal{L}_{out}, \tag{23}$$

$$C > 0 \in \mathbb{R}^{S,S}, \mu > 0 \in \mathbb{R}. \tag{24}$$

Here, w is a matrix of all weights from the network; b is a vector of all matrix biases; a is a normal data cluster center; C is a *positive definite* covariance matrix; μ is a *positive* parameter of fuzzy clustering. $\mathcal{L}_{l2}^{i}$ is a $l2$-regularization loss function with convolution layer numbered i. $\mathcal{L}_{fuzzy}$ is the loss function from the fuzzy layer. $\mathcal{L}_{out}$ is the loss function for the network output.

To solve this problem, the authors use the RMSProp (Root Mean Square Propagation) algorithm [38] with an exponential decay.

Therefore, the network training takes place in several epochs. At the same time, at each epoch, the initial training sample D is randomly divided into a finite set of batches as can be seen in (25).

$$\begin{aligned} D &= B_1 \cup B_2 \cup \ldots \cup B_R, \\ \forall i,j = 1,\ldots,R &\implies B_i \cap B_j = \{\varnothing\}, \\ \forall i = 1,\ldots,R &\implies B_i \subseteq (P(D) \backslash \{\varnothing\}), \\ P(D) &= \{X | X \subseteq D\}. \end{aligned} \tag{25}$$

For each batch $B_j = (D_1, D_2, \ldots, D_V)$, the problem described in (22) is solved. In this case, the functions $\mathcal{L}_{fuzzy}$ and $\mathcal{L}_{out}$ from (23) are calculated as described in (26).

$$\begin{aligned} \mathcal{L}_{fuzzy} &= \sum_{i=1}^{V} (u_i \cdot d_i(a)), \\ \mathcal{L}_{out} &= \frac{1}{V} \sum_{i=1}^{V} (y_i - 1)^2. \end{aligned} \tag{26}$$

Here, V is the batch size; u_i is the typicality degree of the ith block D_i from the batch described in (19); y_i is the network output for block D_i as can be seen in (21).

Thus, in the process of the neural network training, the authors build the distribution of normal data while trying to reach the final estimate of the normality of each block from the training sample closer to the maximum value (it is equal to 1).

2.9. Evaluation Metrics

In some existing works [1,4,8], it is proposed to use as evaluation metrics such functions as accuracy, precision, recall, F1-score, etc. However, these papers propose algorithms that classify data as normal or abnormal. The decision about anomalous data is based on comparing the obtained anomality score with a certain threshold, the value of which can be varied depending on the task. In our paper, we propose a generalized approach that builds only an anomality score. Thus, non-threshold methods are used to evaluate it.

To evaluate our approach, it is proposed to use the integral quality metrics ROC AUC (Area Under Receiver-Operating-Characteristic Curve) and PR AUC (Area Under Precision–Recall Curve), which are widely used in the existing works [6,9,19]. At the same time, in order to obtain a reliable estimation, it is proposed to evaluate the distribution of metrics on various subtasks.

2.10. Materials

During the study of the existing works, the Loghub resource was found [39]. It contains various datasets for solving a large number of tasks related to the log data analysis. The existing works related to the detection of system log anomalies were also analyzed [1,4,6,11,40–42]. Based on the analysis, the following datasets were selected, which contain a sufficient number of anomalies for training the proposed solution and are used in most of the considered works:

1. **HDFS1 (Hadoop Distributed File System).** This dataset is a file containing textual descriptions of events occurring on more than 200 Amazon EC2 nodes. In total, it contains about 11 million events. Each event description contains a block ID—an associated subsystem identifier that can be used to group events. The dataset contains the marking of blocks into anomalous and normal [30].
2. **BGL (Blue Gene/L).** This dataset is a file containing information about events occurring in the BlueGene/L system. In total, it contains about 5 million events. At the same time, the dataset contains information about marking each event as normal or abnormal [36].
3. **Villani.** This dataset is a file containing textual descriptions of keystroke events by individual users collected during their computer work. In total, this dataset contains information about the order of 2 million events, which are key presses and releases for 144 different users [40–42]. For this dataset, it is necessary to build a separate model for each user, while events from a current user are considered as normal, and events from all other users are anomalies [6].

3. Results

To evaluate the proposed solution on the selected datasets, we need to select the hyperparameter values of the approach and also compare the proposed solution with the considered existing approaches based on the constructed evaluation criteria. It is also necessary to evaluate the proposed solution in case of adding random outliers (anomalous blocks of events) to the training sample to confirm the robustness of the solution.

3.1. Evaluation Metrics

As mentioned above, we use ROC AUC and PR AUC as evaluation metrics. At the same time, the distribution of metrics is built dividing the task into separate subtasks. This split is dataset dependent, as described below.

For HDFS1 and BGL datasets, it is proposed to use bootstrapping of the test sample into separate subsamples and to calculate metrics for them.

In the Villani dataset, the anomaly criterion corresponds to a specific user. Therefore, in this task, a separate model for each user is built. At the same time, both the data of the current user (they are marked as normal) and the data of other users (they are marked as anomalous) are used in validation and test samples. For each model, the considered metrics are calculated.

3.2. Experimental Setup

This section describes the specificity of applying individual algorithms to the considered datasets.

3.2.1. Log Collection

For each dataset, the log is a text file consisting of event records sequence. Each entry contains a timestamp and some textual description. For example:

1. **HDFS1**: "2023-02-02 20:55:54 INFO dfs.DataNode$DataXceiver: Receiving block blk_5792489080791696128 src: /10.251.30.6:33145 dest: /10.251.30.6:50010";
2. **BGL**: "2005-06-03-15.42.50.363779 R02-M1-N0-C:J12-U11 RAS KERNEL INFO instruction cache parity error corrected";

3. **Villani**. This dataset is a .csv-file consisting of information about user, system, a key code and timestamp of the key pressed and released.

3.2.2. Log Parsing

For the HDFS1 and BGL datasets, the authors use parsing by the regular expression highlighting timestamp and text description. As for HDFS1, each event corresponds to the block number, which is also allocated. As for BGL, some events are duplicated several times (for reasons of reliability), so it is proposed to filter them.

In the Villani dataset, each record is divided into two events: pressing and releasing a key, described by the key code, the event type (pressing/release), timestamp of the corresponding event and the username. It is proposed to construct an event name using the key code and the type of event (pressing/releasing). Thus, the final preprocessed file contains a set of records including a timestamp, event name (made on the key code and pressing/releasing type) and username.

3.2.3. Feature Extraction

As for feature extraction, the authors use One-hot encoding that builds a dictionary of unique events. Next, the events are grouped as shown below.

1. **HDFS1**.
   (a) Dictionary size: 75.
   (b) As for grouping, the authors use the number of an event block. To build fixed size groups, the first 20 events in each block are selected. For blocks smaller than 20, a special event *!EMPTY EVENT!* is added the required number of times. This event is also added to the dictionary.
2. **BGL**.
   (a) Dictionary size: 728.
   (b) Since the dataset contains information about the anomaly of individual events, it is proposed to solve the problem of predicting the onset of an anomalous event. Data are combined into groups of five events using a sliding window. Then, the result of the work of the solution is the anomality degree of the next event.
3. **Villani**.
   (a) Dictionary size: 446 (223 separate keys with two event types for each key).
   (b) Events for each username are grouped into groups of size 100 using a sliding window. For the correct work of the network, 14 users are selected with at least 312 blocks (312 is the 90% percentile in terms of data volume for all users).

3.2.4. Anomaly Detection

As for anomaly detection, it is necessary to build an anomaly criterion for each dataset as shown below.

1. **HDFS1:** *blocks are considered anomalous if they are marked as anomalous in the original dataset.*
2. **BGL:** *anomalous blocks are those that immediately precede the event marked as anomalous in the original dataset.*
3. **Villani:** *blocks associated with users not included in the training sample are considered abnormal.*

3.3. Technical Characteristics

The authors run all experiments on a Linux server with AMD EPYC 7532 32-CPU and 256GB DDR4 RAM, on which Ubuntu 20.04.4 with Linux kernel 5.4.0-153-generic is running. For greater efficiency, the authors use NVIDIA RTX A6000 GPU with 48 GB RAM.

3.4. Evaluation

To evaluate the proposed solution, it is necessary to divide the initial datasets into training, validation and test sets. The network is trained on the training set. At validation, the ROC AUC and PR AUC are estimated for the selection of hyperparameter values. As for final estimation, the specified metrics are calculated on the test set.

For the HDFS1 and BGL datasets, it is necessary to specify the number of subsamples into which the test sample is divided to estimate the metrics distribution on a test sample. The volumes of the indicated samples for datasets and subsample amount are indicated in Table 1.

Table 1. Sample Size Description for HDFS1 and BGL Datasets.

	Value	HDFS1	BGL
Train		601,576	541,385
Validation	Normal blocks	3367	4263
	Anomalous blocks	3367	4263
Test	Normal blocks	13,471	17,053
	Anomalous blocks	13,471	17,053
	Subsample amount	10	10

As for the Villani dataset, we build an each-against-each model. For each user, N_{train} is considered equals to 80% of the user data as a training sample, N_{val} is considered equal to 4% as a validation sample, and N_{test}—16% of the all user data as a test sample. Next, for all other users, N_{train}, N_{val}, and N_{test} data samples are selected for the training, validation and test sets. Then, for each user, hyperparameter values are tuned on the corresponding validation set and target metrics are calculated. Thus, the final distribution of metrics for all pairs of users is formed.

3.4.1. Evaluation of the Encoder

At the validation stage, the following hyperparameter values are tuned:

1. F—number of independent convolution layers. For all datasets, $F = 3$;
2. $H = (H_1, H_2, \ldots, H_F) \in \mathbb{H}$—filter sizes for each convolution layer (H_i is a filter size for layer numbered i). This parameter must fulfill the constraint described in (16);
3. $K \in \mathbb{K}$—number of filters per layer. This value greatly affects the complexity of the network, so it should not be very large. In addition, to optimize the running time of the proposed solution on a GPU with the mixed precision, it is proposed to consider only values that are multiples of eight, as shown in [43];
4. $\epsilon_{L_2} \in \mathbb{E}$—is the $l2$-regularization constant to compute $\mathcal{L}_{l2}^1$, $\mathcal{L}_{l2}^2$, and $\mathcal{L}_{l2}^3$ in (23).

Also, within the framework of this evaluation, an experimental study of the existing approaches to the vectorization of individual events is carried out. One-hot encoding, random encoding and approaches to constructing informative embeddings (Word2Vec, GloVe) are considered.

For each dataset, we consider the next values of hyperparameters: $F = 3$; $\mathbb{K} = \{8, 16, 32, 64, 128\}$; $\mathbb{E} = \{0.001, 0.01, 0.1, 0.2, 0.5, 0.9, 0.99\}$. The N (an event block size) and $\mathbb{H}$ values are described in Table 2.

Table 2. The N and $\mathbb{H}$ Hyperparameter Values.

Dataset	N	$\mathbb{H}$
HDFS1	20	$\{(3,4,5), (5,10,15), (10,11,12), (12,15,17)\}$
BGL	5	$\{(2,3,4)\}$
Villani	100	$\{(2,3,4), (10,20,30), (20,30,50), (30,50,70)\}$

Also, during the experimental evaluation, as mentioned above, different activation functions of convolutional layers are evaluated. The function $ReLU$ that is used in most works describing similar solutions [19,26], as well as a new function $SELU$ [28] proposed in this paper, are considered. The experiments are carried out for the network architecture from which the Fuzzy layer is removed. Only one layer is used as a decoder. On the validation sample, the number of training epochs and the batch size are selected. Further, on test subsamples, the median values of the target metrics (ROC AUC and PR AUC) are estimated. As a result of the experiments carried out on all datasets, the best results are obtained using the proposed function $SELU$.

Based on the results of the experiments, the following hyperparameter values are chosen for the convolutional part:

1. **HDFS1**. $F = 3$, $K = 16$, $H = (10, 11, 12)$, $\epsilon_{L_2} = 0.001$;
2. **BGL**. $F = 3$, $K = 64$, $H = (2, 3, 4)$, $\epsilon_{L_2} = 0.99$;
3. **Villani**. $F = 3$, $K = 64$, $H = (20, 50, 70)$, $\epsilon_{L_2} = 0.99$.

We also find out that for each dataset, a small increase is offered by the GloVe-based approach. However, this approach offers a slight increase in the value of the metrics compared to One-hot encoding while being computationally complex (a separate model needs to be built and applied). Therefore, as an approach to vectorization, we stopped at the use of One-hot encoding.

3.4.2. Evaluation of the Decoder

Further, experiments to consider different types of decoders are carried out. Various architectures used in existing works (fully connected layers, recurrent layers, convolutional networks) are considered. Experiments are also carried out on an architecture without a fuzzy layer; parallel convolution with the described selected parameters is used as an encoder. Further, on test subsamples, the median values of the target metrics are estimated. According to the results of the experiments, the best result is obtained on a conventional fully connected decoder.

Next, an evaluation of the number of layers in the decoder is conducted. The results of the carried out experiments are shown in Table 3.

Table 3. Comparison of the different layers amount of the fully connected decoder. The comparison is carried out on an architecture without a fuzzy layer.

Dataset	Layers Amount	ROC AUC			PR AUC		
		Median	Q1	Q3	Median	Q1	Q3
HDFS1	1	0.73	0.711	0.734	0.752	0.744	0.754
	2	0.734	0.713	0.735	0.753	0.749	0.755
	3	0.729	0.727	0.731	0.751	0.749	0.753
BGL	1	0.684	0.681	0.687	0.697	0.695	0.713
	2	0.685	0.682	0.686	0.698	0.696	0.711
	3	0.683	0.682	0.685	0.695	0.694	0.696
Villani	1	0.631	0.59	0.645	0.657	0.643	0.667
	2	0.655	0.631	0.671	0.678	0.666	0.682
	3	0.641	0.637	0.648	0.662	0.66	0.664

As we can see, increasing the number of layers does not offer a significant increase in accuracy on all considered datasets. Therefore, in the final architecture, it is proposed to use only one fully connected layer.

In addition, we see that the metric values obtained only with the help of the autoencoder are quite low, which does not allow the use of this approach to solve the problem

of detecting the pre-failure state of the system. Therefore, it is necessary to improve the proposed approach with the addition of a Fuzzy layer.

3.4.3. Evaluation of the Fuzzy Layer

To evaluate the fuzzy layer, the parameters of the fuzzy clustering layer are selected. In particular, the following parameter values are considered:

1. Fuzziness degree m in (19): $m \in \{X | X = 1 + 0.1 \cdot i, i = 1, 2, \ldots, 10\}$;
2. Kernel function Φ in (20). Here, various types of radial basis functions are considered based on the review carried out in work [29]. Sigmoidal, linear and quadratic functions are considered;
3. Distance function $d_j(a)$ in (20). The classical Euclidean distance and the Mahalanobis distance are considered;
4. Algorithms for initializing trainable parameters μ, a and C (for Mahalanobis distance). Initializers based on normal distribution and constant initializers are considered. They are used in the existing works on similar topics [19,26].

The selection of parameters is based on the validation dataset. As the parameters of the convolutional asymmetric decoder, the parameters that offer the best result based on the experimental study carried out in the previous chapter and described earlier are used.

Based on the experiments, it is found that the following parameters are the best for all datasets:

1. $m = 2$;
2. $\Phi(x) = x$;
3. For the initial value of parameter a, the best initializer is based on a continuous uniform distribution with mean equal to zero and variance equal to one; for the initial value of parameter C, a constant identity diagonal matrix is used (stored as a vector of diagonal values); for the initial value of the parameter μ, the best as the initial value is the constant zero;
4. As a distance metric, the Mahalanobis distance shows the best results, which allows building an elliptical-shaped normal data cluster.

3.5. Comparison with Existing Approaches

To compare the proposed solution, the authors decided to use three existing approaches: One-class SVM, Fuzzy and LogBERT (a Transformer).

We also considered architectures based on GAN and VAE, but they did not show good results in preliminary experiments.

During the experiments, the authors found out that SVM and fuzzy methods showed the best results when encoding blocks of events in the form of an event set model: for the One-class SVM, the best block embedding was the sum of the One-hot representations of individual events; for the fuzzy method, the TF-IDF (Term Frequency-Inverse Document Frequency) representation became the best. The LogBERT approach is based on the Transformer architecture described above and uses the sequence model as an embedding for the event block. In this case, an event index vectorized using a sinusoid was added to each One-hot representation of a single event. For all algorithms, hyperparameter values were tuned on a validation sample, as described above. The considered hyperparameter values are presented below.

1. One-class SVM:
   (a) Kernel activation function $F \in \{RBF, polynomial, linear\}$;
   (b) Parameter γ of the RBF kernel $\gamma \in \{X | X = 0.1 \cdot i, i = 1, 2, \ldots, 9\}$.
2. Fuzzy:
   (a) Degree of fuzziness $m \in \{X | X = 1 + 0.1 \cdot i, i = 1, \ldots, 10\}$;
   (b) Outliers percent $n \in \{X | X = 0.05 \cdot i, i = 1, 2, \ldots, 10\}$;
   (c) Parameter γ of the RBF kernel $\gamma \in \{X | X = 0.1 \cdot i, i = 1, 2, \ldots, 9\}$.

3. LogBERT:
   (a) The amount of Transformer layers $N_T \in \{2, 3, 4\}$;
   (b) Token embedding size $M \in \{X | X = 50 \cdot i, i = 1, 2, \ldots, 5\}$;
   (c) Hidden state embedding size $H \in \{16, 32, 64, 128, 256\}$;
   (d) Masked event percent p_m for each block $p_m \in \{X = 0.05 \cdot i, i = 1, 2, \ldots, 15\}$;
   (e) The number of candidates g for determining the anomality degree of the predicted event $g \in \{X = 5, 6, \ldots, 17\}$.

According to the experimental comparison of the considered existing approaches, the best result was shown by the LogBERT algorithm. Its comparison with the proposed approach is shown in Table 4.

Table 4. Comparison of the Proposed Approach with the LogBERT algorithm.

Dataset	Approach	ROC AUC			PR AUC		
		Median	Q1	Q3	Median	Q1	Q3
HDFS1	LogBERT	0.9	0.894	0.907	0.901	0.898	0.911
	FuzzyCNN	***0.973***	***0.971***	***0.974***	***0.97***	***0.969***	***0.972***
BGL	LogBERT	0.908	0.901	0.912	0.898	0.893	0.901
	FuzzyCNN	***0.939***	***0.934***	***0.94***	***0.921***	***0.916***	***0.926***
Villani	LogBERT	0.8	0.781	0.81	0.821	0.819	0.824
	FuzzyCNN	***0.856***	***0.843***	***0.863***	***0.867***	***0.855***	***0.87***

As can be seen, the median values of all metrics on all datasets as well as the values of Q1 and Q3 determining the range of metrics distribution are higher for the proposed method than for the Transformer algorithm, which showed the best result among the existing solutions.

In addition, we see that the range between Q1 and Q3 in all experiments is small, which indicates that the solution allows the obtention of almost the same result on all considered subsamples.

3.6. Robustness Estimation

To explore the robustness of the outlier approach, the authors added anomalous data to the training set, labeling them as normal. The anomalous data were taken from the original test set and were not used during the testing phase. As outliers, the authors used a certain percentage (20% and 50%) of all anomalous events that were added to the training set. Further, all experiments were carried out in the same way as described above. The results of experiments are shown in Table 5.

Table 5. Robustness Estimation of the Proposed Approach.

Dataset	Percent	ROC AUC			PR AUC		
		Median	Q1	Q3	Median	Q1	Q3
HDFS1	0.2	0.959	0.955	0.963	0.958	0.953	0.963
	0.5	0.937	0.933	0.94	0.944	0.94	0.949
BGL	0.2	0.846	0.844	0.85	0.872	0.871	0.874
	0.5	0.805	0.8	0.81	0.845	0.843	0.848
Villani	0.2	0.823	0.81	0.825	0.831	0.829	0.841
	0.5	0.78	0.767	0.792	0.793	0.787	0.799

As can be seen from the experiments, the quality of the algorithm decreases with an increase in the amount of outliers, which is logical (because they shift the distribution of training data). However, even with a relatively large number of outliers, the quality of

the algorithms on the average does not significantly decrease on some considered datasets (on the HDFS1 dataset, values of the PR AUC and ROC AUC metrics drop by 3–4%; for Villany—by 9%). As for the BGL dataset, ROC AUC and PR AUC values drop by 15% for 50% outliers and drop by 10% for 20% outliers. Thus, the proposed solution can be considered as robust.

4. Discussion

This section presents an analysis of the results obtained in comparison with the existing approaches, considering their accordance to the following criteria of scientific importance: difference from known results and scientific novelty.

The existing classical approaches to unsupervised anomaly detection do not allow achieving high accuracy [6,8,21]. In addition, most approaches based on building a cluster of normal data [5] build only a simple cluster, which does not allow taking into account all the necessary information about the collected data. In this paper, we propose to introduce our own fuzzy layer based on [6] using the RBF kernel and the Mahalanobis distance, which makes it possible to build elliptical clusters.

In addition, the running time of the existing approaches has a quadratic dependence on the amount of data [22,23], which makes them inapplicable to log data, which can be very large. More efficient are approaches based on the architecture of the Transformer [8]. However, they use the self-attention operation [16,18], which also significantly increases the time complexity of the algorithm. The proposed solution is based on a parallel convolution operation and an asymmetric autoencoder with a simple fully connected decoder, which can significantly reduce the overheads [16].

Based on the results of the experimental studies, we can conclude that the proposed solution to the unsupervised anomaly detection problem in logs for computer security and reliability tasks allows us the obtention of better results compared to the existing approaches [5,6,8] for all considered datasets [30,36,40–42].

The scientific novelty of the work consists in the proposed new approach for constructing features for descriptions of event blocks based on an asymmetric self-normalized autoencoder with parallel convolution in the encoder and a fully connected decoder and a proposed new approach to unsupervised anomaly detection in log data based on fuzzy elliptic clustering of convolutional features.

Further Research

In this paper, we consider the approach that only allows estimation of the typicality score of event blocks. As a result of the work, a certain value is obtained—the typicality score. To make a final decision about the presence of anomalies in the considered block, it is necessary to compare the score with a certain threshold. This threshold is highly dependent on the specific systems under consideration and may vary depending on the task. The study of the threshold is beyond the scope of this paper and will be considered in future research.

In addition, this paper proposes the new approach to determining the typicality score of system log data. Its theoretical and experimental evaluation is given. A more detailed study of the approach, including the study of the limitations (computational complexity, number of parameters and running time for various data volumes) and practical implementation of the anomaly detection system, will be carried out in future research. Also, we are planning to adapt the solution for analyzing a wider class of data, non-structured data (texts in particular), to solve the anomaly detection problem.

5. Conclusions

In this paper, the authors propose a new approach to unsupervised anomaly detection in system logs based on the use of the Fuzzy CNN asymmetric autoencoder. A detailed description of the proposed approach is given, as well as its qualitative evaluation and

experimental comparison with the existing solutions. As a result of the experiments, the authors obtain the following values of classification quality metrics:

1. **HDFS1 dataset**. Median ROC AUC = 0.973, median PR AUC = 0.97;
2. **BGL dataset**. Median ROC AUC = 0.939, median PR AUC = 0.921;
3. **Villani dataset**. Median ROC AUC = 0.856, median PR AUC = 0.867.

Stability evaluation of the approach to outliers is also presented. Based on the experimental results, we can conclude that even when outliers are added to the training sample, the proposed approach also shows good results: with the addition of 20% outliers to the training sample, the quality of the final algorithm drops by no more than 10%.

According to the results of the experiments, it can be concluded that the proposed solution is resistant to outliers.

Therefore, the fast parallel convolution operation and the fully connected decoder used make it possible to obtain a more efficient solution compared to the existing Transformer architecture and classical approaches.

Author Contributions: Conceptualization, M.P. and I.M.; methodology, O.G., M.P. and M.K.; software, O.G. and M.K.; validation, O.G., I.M. and M.K.; formal analysis, O.G., M.P. and M.K.; investigation, M.P. and I.M.; resources, M.P. and I.M.; data curation, O.G. and M.K.; writing—original draft preparation, O.G. and M.K.; writing—review and editing, M.P. and I.M.; visualization, O.G.; supervision, M.P.; project administration, I.M. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by Non-commercial Foundation for the Advancement of Science and Education "INTELLECT".

Data Availability Statement: Datasets which were used in this paper: HDFS1 and BGL can be downloaded from the links https://github.com/logpai/loghub/tree/master/HDFS (accessed on 1 August 2023) and https://github.com/logpai/loghub/tree/master/BGL (accessed on 1 August 2023). The Villani dataset and the research computer code are available on request.

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

References

1. Chen, Z.; Liu, J.; Gu, W.; Su, Y.; Lyu, M.R. Experience report: Deep learning-based system log analysis for anomaly detection. *arXiv* **2021**, arXiv:2107.05908.
2. Wang, B.; Hua, Q.; Zhang, H.; Tan, X.; Nan, Y.; Chen, R.; Shu, X. Research on anomaly detection and real-time reliability evaluation with the log of cloud platform. *Alex. Eng. J.* **2022**, *61*, 7183–7193. [CrossRef]
3. Landauer, M.; Skopik, F.; Wurzenberger, M.; Rauber, A. System log clustering approaches for cyber security applications: A survey. *Comput. Secur.* **2020**, *92*, 101739. [CrossRef]
4. He, S.; Zhu, J.; He, P.; Lyu, M.R. Experience report: System log analysis for anomaly detection. In Proceedings of the 2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE), Ottawa, ON, Canada, 23–27 October 2016; pp. 207–218.
5. Manevitz, L.M.; Yousef, M. One-class SVMs for document classification. *J. Mach. Learn. Res.* **2001**, *2*, 139–154.
6. Kazachuk, M.; Petrovskiy, M.; Mashechkin, I.; Gorohov, O. Novelty Detection Using Elliptical Fuzzy Clustering in a Reproducing Kernel Hilbert Space. In Proceedings of the Intelligent Data Engineering and Automated Learning–IDEAL 2018: 19th International Conference, Madrid, Spain, 21–23 November 2018; Proceedings, Part II 19; Springer: Berlin/Heidelberg, Germany, 2018; pp. 221–232.
7. Du, M.; Li, F.; Zheng, G.; Srikumar, V. Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, Dallas, TX, USA, 30 October–3 November 2017; pp. 1285–1298.
8. Guo, H.; Yuan, S.; Wu, X. Logbert: Log anomaly detection via bert. In Proceedings of the 2021 International Joint Conference on Neural Networks (IJCNN), Shenzhen, China, 18–22 July 2021; pp. 1–8.
9. Chandola, V.; Banerjee, A.; Kumar, V. Anomaly detection: A survey. *ACM Comput. Surv. (CSUR)* **2009**, *41*, 1–58. [CrossRef]
10. Chalapathy, R.; Chawla, S. Deep learning for anomaly detection: A survey. *arXiv* **2019**, arXiv:1901.03407.

11. Yadav, R.B.; Kumar, P.S.; Dhavale, S.V. A survey on log anomaly detection using deep learning. In Proceedings of the 2020 8th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO), Noida, India, 4–5 June 2020; pp. 1215–1220.
12. Mi, H.; Wang, H.; Zhou, Y.; Lyu, M.R.T.; Cai, H. Toward fine-grained, unsupervised, scalable performance diagnosis for production cloud computing systems. *IEEE Trans. Parallel Distrib. Syst.* **2013**, *24*, 1245–1255. [CrossRef]
13. Liu, Y.; Zhang, X.; He, S.; Zhang, H.; Li, L.; Kang, Y.; Xu, Y.; Ma, M.; Lin, Q.; Dang, Y.; et al. Uniparser: A unified log parser for heterogeneous log data. In Proceedings of the ACM Web Conference 2022, Lyon, France, 25–29 April 2022; pp. 1893–1901.
14. Zhu, J.; He, S.; Liu, J.; He, P.; Xie, Q.; Zheng, Z.; Lyu, M.R. Tools and benchmarks for automated log parsing. In Proceedings of the 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP), Montreal, QC, Canada, 25–31 May 2019; pp. 121–130.
15. Le, V.H.; Zhang, H. Log-based anomaly detection with deep learning: How far are we? In Proceedings of the 44th International Conference on Software Engineering, Pittsburgh, PA, USA, 25–27 May 2022; pp. 1356–1367.
16. Chollet, F. *Deep Learning with Python*; Simon and Schuster: New York, NY, USA, 2021.
17. What Are Vector Embeddings. Available online: https://www.pinecone.io/learn/vector-embeddings/ (accessed on 1 August 2023).
18. Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A.N.; Kaiser, Ł.; Polosukhin, I. Attention is all you need. *Adv. Neural Inf. Process. Syst.* **2017**, *30*, 6000–6010.
19. Gorokhov, O.; Petrovskiy, M.; Mashechkin, I. Convolutional neural networks for unsupervised anomaly detection in text data. In Proceedings of the International Conference on Intelligent Data Engineering and Automated Learning, Guilin, China, 30 October–1 November 2017; Springer: Berlin/Heidelberg, Germany, 2017; pp. 500–507.
20. Girolami, M. Mercer kernel-based clustering in feature space. *IEEE Trans. Neural Netw.* **2002**, *13*, 780–784. [CrossRef]
21. Petrovskiy, M. Outlier detection algorithms in data mining systems. *Program. Comput. Softw.* **2003**, *29*, 228–237. [CrossRef]
22. Liu, D.; Qian, H.; Dai, G.; Zhang, Z. An iterative SVM approach to feature selection and classification in high-dimensional datasets. *Pattern Recognit.* **2013**, *46*, 2531–2537. [CrossRef]
23. Erfani, S.M.; Rajasegarar, S.; Karunasekera, S.; Leckie, C. High-dimensional and large-scale anomaly detection using a linear one-class SVM with deep learning. *Pattern Recognit.* **2016**, *58*, 121–134. [CrossRef]
24. Mahalanobis, P.C. On the generalized distance in statistics. *Sankhyā: Indian J. Stat. Ser. A* **2018**, *80*, S1–S7.
25. Graves, A. Generating sequences with recurrent neural networks. *arXiv* **2013**, arXiv:1308.0850.
26. Kim, Y. Convolutional neural networks for sentence classification. *arXiv* **2014**, arXiv:1408.5882.
27. LeCun, Y.; Bottou, L.; Bengio, Y.; Haffner, P. Gradient-based learning applied to document recognition. *Proc. IEEE* **1998**, *86*, 2278–2324. [CrossRef]
28. Klambauer, G.; Unterthiner, T.; Mayr, A.; Hochreiter, S. Self-normalizing neural networks. *Adv. Neural Inf. Process. Syst.* **2017**, *30*, 972–981.
29. Amirian, M.; Schwenker, F. Radial basis function networks for convolutional neural networks to learn similarity distance metric and improve interpretability. *IEEE Access* **2020**, *8*, 123087–123097. [CrossRef]
30. Xu, W.; Huang, L.; Fox, A.; Patterson, D.; Jordan, M.I. Detecting large-scale system problems by mining console logs. In Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles, Big Sky, MT, USA, 11–14 October 2009; pp. 117–132.
31. Liu, F.T.; Ting, K.M.; Zhou, Z.H. Isolation forest. In Proceedings of the 2008 Eighth IEEE International Conference on Data Mining, Pisa, Italy, 15–19 December 2008; pp. 413–422.
32. Lin, Q.; Zhang, H.; Lou, J.G.; Zhang, Y.; Chen, X. Log clustering based problem identification for online service systems. In Proceedings of the 38th International Conference on Software Engineering Companion, Austin, TX, USA, 14–22 May 2016; pp. 102–111.
33. Meng, W.; Liu, Y.; Zhu, Y.; Zhang, S.; Pei, D.; Liu, Y.; Chen, Y.; Zhang, R.; Tao, S.; Sun, P.; et al. Loganomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs. In Proceedings of the IJCAI, Macao, China, 10–16 August 2019; Volume 19, pp. 4739–4745.
34. Duan, X.; Ying, S.; Yuan, W.; Cheng, H.; Yin, X. A Generative Adversarial Networks for Log Anomaly Detection. *Comput. Syst. Sci. Eng.* **2021**, *37*, 135–148. [CrossRef]
35. Zhou, Y.; Liang, X.; Zhang, W.; Zhang, L.; Song, X. VAE-based deep SVDD for anomaly detection. *Neurocomputing* **2021**, *453*, 131–140. [CrossRef]
36. Oliner, A.; Stearley, J. What supercomputers say: A study of five system logs. In Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07), Edinburgh, UK, 25–28 June 2007; pp. 575–584.
37. Cosine Similarity. Available online: https://www.learndatasci.com/glossary/cosine-similarity/ (accessed on 1 August 2023).
38. Hinton, G.; Srivastava, N.; Swersky, K. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. *Cited* **2012**, *14*, 2.
39. He, S.; Zhu, J.; He, P.; Lyu, M.R. Loghub: A large collection of system log datasets towards automated log analytics. *arXiv* **2020**, arXiv:2008.06448.
40. Tappert, C.C.; Villani, M.; Cha, S.H. Keystroke biometric identification and authentication on long-text input. In *Behavioral Biometrics for Human Identification: Intelligent Applications*; IGI Global: Hershey, PA, USA, 2010; pp. 342–367.

41. Monaco, J.V.; Bakelman, N.; Cha, S.H.; Tappert, C.C. Developing a keystroke biometric system for continual authentication of computer users. In Proceedings of the 2012 European Intelligence and Security Informatics Conference, Odense, Denmark, 22–24 August 2012; pp. 210–216.
42. Villani, M.; Tappert, C.; Ngo, G.; Simone, J.; Fort, H.S.; Cha, S.H. Keystroke biometric recognition studies on long-text input under ideal and application-oriented conditions. In Proceedings of the 2006 Conference on Computer Vision and Pattern Recognition Workshop (CVPRW'06), New York, NY, USA, 17–22 June 2006; p. 39.
43. Micikevicius, P.; Narang, S.; Alben, J.; Diamos, G.; Elsen, E.; Garcia, D.; Ginsburg, B.; Houston, M.; Kuchaiev, O.; Venkatesh, G.; et al. Mixed precision training. *arXiv* **2017**, arXiv:1710.03740.