electronics

Article

Scalable Inline Network-Intrusion Detection System with Minimized Memory Requirement

Taehoon Kim and Wooguil Pak *

Department of Information and Communication Engineering, Yeungnam University, Gyeongsan 38541, Republic of Korea
* Correspondence: wooguilpak@yu.ac.kr; Tel.: +82-53-810-3092

Abstract: Currently used network-intrusion detection systems (NIDSs) using deep learning have limitations in processing large amounts of data in real time. This is because collecting flow information and creating features are time consuming and require considerable memory. To solve this problem, a novel NIDS with θ(1) memory complexity for processing a flow is proposed in this study. Owing to its small memory requirement, the proposed model can handle numerous concurrent flows. In addition, it uses raw packet data as input features for the deep learning models, resulting in a lightweight feature-creation process. For fast detection, the proposed NIDS classifies a flow using a received packet, though it is prone to false detection. This weakness is solved through the validation model proposed in this research, resulting in high detection accuracy. Furthermore, real-time detection is possible since intrusion detection can be performed for every received packet using the Inception model. A performance comparison with existing methods confirmed an effectively improved detection time and lower memory requirement by 73% and 77% on average while maintaining high detection accuracy. Thus, the proposed model can effectively overcome the problems with modern deep-learning-based NIDSs.

Keywords: machine learning; network intrusion detection system; Inception-ResNet; low memory requirement; packet feature

Citation: Kim, T.; Pak, W. Scalable Inline Network-Intrusion Detection System with Minimized Memory Requirement. *Electronics* **2023**, *12*, 2061. https://doi.org/10.3390/electronics12092061

Academic Editor: Cheng-Chi Lee

Received: 24 February 2023
Revised: 19 April 2023
Accepted: 28 April 2023
Published: 29 April 2023

1. Introduction

Network intrusion detection systems (NIDSs) are crucial for protecting networks and users from security threats [1,2]. Therefore, NIDSs have been recently developed using the latest technologies such as machine learning. Accurately detecting an intrusion is a challenging task because with the exponential increase in the amount of data traffic, network intrusion attempts increase, and the types of intrusions become diverse. Therefore, a NIDS that uses deep learning (DL) was proposed [3–7].

However, deep-learning-based NIDSs cannot satisfactorily perform real-time processing of large-scale received network traffic [2–7] due to two reasons. First, a NIDS with DL generally requires feature creation and classification of the received features, which are time-consuming processes and require considerable memory. Second, collecting sufficient packets belonging to a flow or waiting until the flow ends is a prerequisite for applying DL.

When a NIDS cannot process the incoming packets in real time, detecting network intrusions without delay becomes challenging. Consequently, a novel approach that is different from the existing DL-based NIDS is required to solve this problem. The novel approach proposed in this study has the following characteristics. The proposed method first converts the received packets directly into features using light processing. Although an accurate feature can be created through a sophisticated algorithm and statistical analysis of the received packet, this process inevitably causes delay. Therefore, in the proposed method, the packet is used as a feature value to remove the delay.

In addition, the proposed method does not require the collection of packets belonging to an entire flow; instead, intrusion detection is performed using each received packet.

A flow comprises several packets; therefore, it becomes difficult to determine whether a flow is used for intrusion with only a single packet. However, despite this risk, detecting an attack using a single packet significantly reduces the amount of memory required in comparison with the existing methods. Using a single packet to detect intrusions has been used in existing NIDSs, including SNORT; however, in these NIDSs, a pattern-based approach is typically used for each packet to detect malicious traffic [1,2]. Recently, single-packet-based NIDSs that do not use pattern-based matching have been proposed, though most of them produce classification models trained biasedly on training datasets [8,9]. Such models have a limitation in that the detection accuracy is greatly reduced in a real network where hosts not included in the training dataset exist [10,11].

The proposed method differs from the existing single-packet-based NIDSs such that it trains a classifier using a training dataset modified to prevent biased learning and classifies each flow using a single packet as an input to the classifier. In addition, unlike the existing methods, the proposed method reflects the classification result with the highest score among the previous packets to the classification of the current packet, in order to consider not only the characteristics of a single packet but also the overall characteristics of the flow. In addition, a validation model is proposed to prevent false positives of the proposed single-packet-based classification.

The proposed method differs from existing single-packet-based NIDSs such that a single packet is used to classify flows. The method can classify each flow by considering not only the characteristics of the single packet but also the overall characteristics of the flow by reflecting the classification result for the previous packet into the classification for the current packet.

If an existing method using flow data creates a feature for the entire traffic of a flow or collects a certain amount of traffic and subsequently creates a feature using it, a large buffer should be allocated to each flow. The number of simultaneous flows quickly increases in proportion to the size of the network; therefore, the method using a large buffer per flow has considerably low scalability in terms of the total number of concurrent flows.

However, the proposed method can support several flows that cannot be processed in the existing method since it consumes only one word of memory per flow. The major contributions of this study are as follows:

1. Presenting a DL model for NIDSs with extremely-low feature-creation overhead. Raw packet data are directly used as features to the classifier, and their output is reused as features for the validation classifier by combining with the packet header, eliminating feature-generation overhead;
2. Designing a single-packet-based classification algorithm with complexity $\theta(1)$ of memory requirement per flow. Although existing NIDS usually requires hundreds or thousands of bytes of memory to process one flow, the proposed NIDS requires only one-word memory for each flow;
3. Proposing a NIDS with a high classification accuracy and a very short detection time. By performing classification on every received packet, it has a short detection time and uses a unique validation model to maintain high classification accuracy.

The remainder of this paper is structured as follows. In Section 2, we review previous studies. The proposed method is described in Section 3. Section 4 presents a comparison of the performance of the proposed method with that of existing methods. Finally, the conclusions are presented in Section 5.

2. Existing Work

Machine learning (ML)/DL-based NIDSs have been actively studied, and various NIDSs have been proposed. The advantages and disadvantages of these ML/deep-learning-based NIDS are determined based on the features used. Therefore, in this section, ML/deep-learning-based NIDS studies are classified according to their feature characteristics, and each category is explained in detail. According to a study, the features applied to an

ML/deep-learning-based NIDS can be largely classified into three types, namely flow, packet, and flow-packet hybrid features. Each type will be described in detail.

2.1. Flow Features

The flow feature quantifies the characteristics of traffic flows between each source and destination node [12,13]. Here, a flow is a logical group of traffic distinguished by a quintuple composed of source IP, destination IP, source port, destination port, and protocol. Therefore, all packets with the same quintuple are treated as the same flow.

Intraflow features are created from each flow data and interflow features are created from the data of flows that occur between the same source-destination node. Intraflow collects all traffic from the first packet to the last packet of each flow and subsequently creates a feature using it. Typical features include the flow duration for one flow, the total number of transmitted/received packets, and the total amount of transmitted/received traffic. Whereas the interflow feature reflects the operational characteristics of all flows occurring between the same source-destination node and includes the total number of flows occurring within a specific time, the number of failed flows, and the interflow gap.

The flow feature can effectively distinguish between intrusion detection and normal flows since it compressively expresses all flow characteristics. However, since many flow features require information regarding the entire flow traffic, considerable memory is used to collect them. To solve this problem, a method of storing simple log data for packets instead of received packets is used; however, this method also has a memory complexity of $\theta(P)$, where P is the total number of packets constituting a flow. Therefore, the NIDS has a limitation on the maximum number of flows that can be supported. Considering that the number of simultaneous flows that a single NIDS must handle increases rapidly as the network grows, this is a severe limitation.

An intrusion is detected after the intrusion is completed since most intraflow features can be created after the flow ends, even if the NIDS receives a flow attempting to infiltrate the network. For a NIDS to safely protect the network, an intrusion must be detected in real time without delay after it has occurred. However, flow-feature-based NIDS cannot perform detection in real time.

In addition to the flows between specific source and destination nodes, features can be extracted from all flows that occur between multiple source nodes and one destination node. This host-specific flow feature is highly advantageous for detecting security threats, such as DDoS, which target a specific node using many nodes. However, this feature adds more computational and memory overhead since it requires an aggregate operation for each host-specific flow. The following Table 1 summarizes the characteristics of flow-feature-based datasets.

Table 1. Flow feature-based NIDS datasets.

Name	Feature Type	Feature Size	Total Feature Size	Total Class Size
CSE-CIC-IDS2018 [12]	Intraflow	76	80	14
	Interflow	4		
AWID [14]	Intraflow	263	263	13
Kyoto2016 [15]	Intraflow	17	24	2
	Interflow	7		
NSL KDD [16]	Intraflow	22	41	39
	Interflow	19		

2.2. Packet Features

Flow features inevitably require high-overhead processes, such as collection and classification by traffic flow and feature creation [12,13]. However, with the development

of highly sophisticated ML/DL models, the algorithms use raw traffic instead of high-quality features through a complex process that has been developed [2]. In this case, the system structure is considerably simpler than that of the flow features since the DL model automatically generates excellent features from low traffic and performs training and classification. In particular, packet features effectively reflect patterns for intrusion detection in the packets; therefore, they are advantageous for deep packet inspection.

The process of creating features from raw traffic involves removing fields that can create bias, such as the source and destination IPs, dividing the remaining data into bytes or words, and mapping each value to one feature. In this case, when one-hot encoding is applied to each byte value, it may be mapped to a plurality of features; however, the number of features increases, and the time required to create features also increases.

A packet feature is not generated from a single packet, but by receiving and combining multiple packets belonging to the same flow, which is similar to a flow feature. Determining whether a flow is malicious through a single packet results in low reliability because a single packet reflects only a minute portion of the characteristics of the entire flow. Therefore, traffic of a certain number or size should be collected from the beginning of the flow, through which the DL model is learned and the received packet is classified. Therefore, a buffer is required for traffic collection, which is generally "flow total traffic size $\gg$ buffer size $\gg$ total flow feature size." Consequently, since a buffer is allocated to each current active flow, a large amount of memory is consumed. Memory usage can be reduced by reducing the amount of traffic to be collected per flow; however, this reduction may result in lower intrusion detection accuracy, rendering circumvention by an attacker and allowing a successful attack. Furthermore, since intrusion is determined based on a single flow, the detection accuracy may be low in distributed attacks, such as DDoS attacks, in which attack patterns are revealed from the perspective of interflow.

Recently, research on detecting intrusion with only single packet data have been proposed. However, most of them use unmodified training datasets that include host and flow-dependent fields such as source IP, destination IP, source port, and destination port to train the classifier model [8,9]. In this case, the classifier learns the attacker's or victim's IP or port number as an important feature rather than the distinctive characteristics of intrusions from the packet data. The classifier trained in this way shows very-high detection accuracy for the test dataset built on the same network as the training dataset even with a single packet. However, such classifiers are highly inaccurate to detect intrusions in real networks where the attacker's or victim's IP or port number cannot be known in advance [10,11].

Unlike the flow feature, in the packet feature, traffic for the entire flow is not used, but only a portion of it; therefore, intrusion can be detected faster than NIDS using the flow feature by performing detection before the flow ends [2]. However, collecting traffic for the same flow can reduce the efficiency of parallel packet processing. With the increase in the amount of traffic, a parallel processing structure is used in most networks or network security systems to process packets in the lower layer at high speed. However, traffic collection for the same flow in a parallel processing system generates considerable data for transmission and reception for traffic synchronization between parallel devices. This overhead can be maximized in an environment with multiple interfaces. Therefore, the method is not optimized for a modern distributed network traffic-processing method since the flow dependency of traffic to support the parallel processing for flows cannot be completely removed.

2.3. Hybrid Features

Hybrid features are created by including both the flow and packet features [17]. Although this method has the highest overhead in generating features, it simultaneously considers the characteristics of the entire flow through the flow feature and the pattern through the packet feature to obtain a higher detection performance than a NIDS using a single-feature set. However, intrusion detection is possible only when the flow is terminated

due to the limitations of the flow feature. Thus, real-time detection cannot be supported, and applying the parallel processing method is critical. Compared with flow features or packets, limited studies have focused on hybrid features. Table 2 presents the results of a comparative analysis of research results for each feature.

Table 2. NIDS characteristics according to the feature type.

Feature Type	Pros.	Cons.
Flow feature	Stable detection rate as it reflects the characteristics of the entire flow	Feature creation requires considerable memory and computation time. Since intrusion detection is performed after the flow ends, intrusion detection is delayed.
Packet feature	Support deep packet inspection. Fast feature creation enables detection even before the end of flow	Vulnerable to detection bypass. Support for partial parallelism
Hybrid feature	Very-high detection rate	Very-high memory and computational complexity.

Recently, many studies have been proposed to solve the limitations of existing NIDS by applying various technologies in addition to research on features. Table 3 shows each characteristic of studies applying the latest technologies such as fused machine learning, blockchain, and MapReduce [18–20].

Table 3. Comparison of recent NIDS studies according to applied technology.

NIDS Type	Core Technology	Objective
Proposed	Inception-ResNet	Fast and accurate detection of network intrusion
[18]	Fused machine learning	Detecting intrusion in a heterogeneous network that consists of different source networks.
[19]	Blockchain, IoT technology	Data collecting network traffic data gathered using IoT technology without any delays and saved in blockchain technology-secured clouds.
[20]	MapReduce	Automating intrusion detection by processing a vast amount of data with MapReduce.

3. Proposed Algorithm

In the proposed method, packet data are used as a feature to determine the network intrusion whenever a single packet is received. Detecting network intrusions using a single packet is extremely difficult and the detection reliability is low. Therefore, each detection result was verified, and if the result was unreliable, the detection was attempted again when the next packet of the same flow was received. The previous detection result was included as an additional input feature when classifying the next packet. However, using considerable memory to store existing detection results is a great burden for increasing the number of concurrent NIDS flows. Therefore, the existing detection results are stored with minimal information, enabling large-capacity concurrent-flow processing.

Figure 1 illustrates the overall structure of the proposed model, which determines whether a single packet is malicious by using a packet classifier composed of an Inception response and a fully connected network (FCN). In general, a NIDS creates flow entries matched for each currently active flow, to collect information about the flow and applies policies to the flow. The proposed method extends the flow entry to store the intermediate classification result, i.e., class ID and score. Whenever the NIDS receives a new packet, it searches for a matching flow entry. If found, it indicates that the packet belongs to an existing flow. On the other hand, if the search fails, a new entry is created to process a new flow.

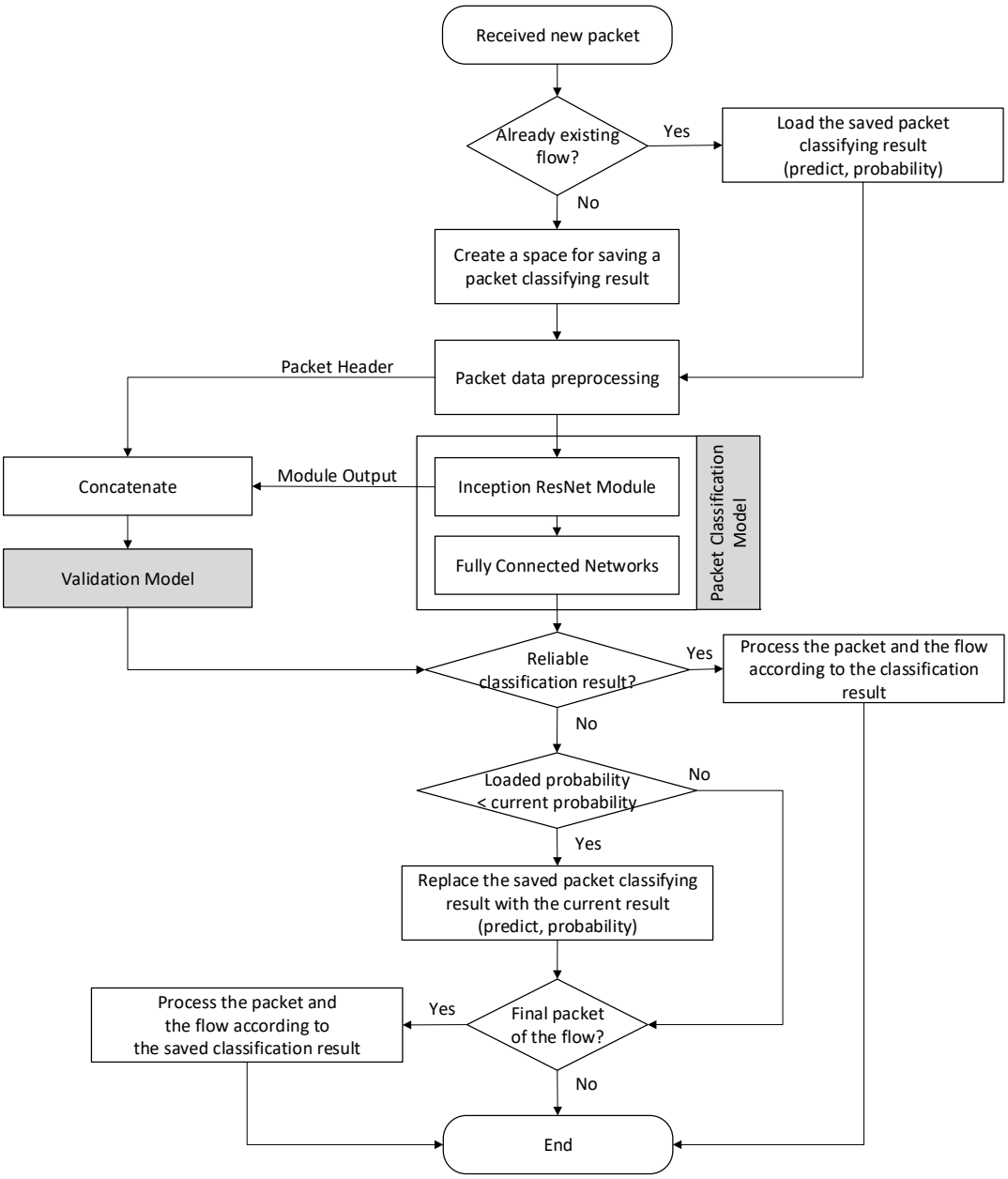


Figure 1. Flowchart of the proposed NIDS. The gray blocks represent the two classifiers that make up the proposed NIDS.

In the proposed method, whenever a packet is received, the previous classification results in the flow entry and packet data are combined into the input of the packet classifier called the Inception-ResNet classifier. The output of Inception-ResNet and the packet header are combined into the input to another classifier called the validation model. The output of Inception-ResNet is again given as an input to the FCN and the classification result is obtained. If the output of the validation model is that the classification result is reliable, the result is stored in the flow entry as the final classification result.

On the other hand, if the output is that it is unreliable, and the current classification score is higher than the score stored in the entry, the stored result is updated with the current one. If every output of the validation model is that it is unreliable until the flow ends, the terminated flow is classified as the class stored in the entry.

When determining the network intrusion for an entire flow with only one packet, a high probability of misclassifying the packet exists owing to the lack of flow information.

In the proposed method, three strategies are used to compensate for this weakness. Now, each strategy used in the proposed method will be described in detail.

First, only the class and probability values among the classification results of the packets prior to the currently received packet are stored, and this classification class is used as an additional feature to classify the current packet. This phenomenon improves the accuracy when determining a flow with only a single packet. Moreover, in this method, only the classification class and probability value are stored for one flow with considerable memory saved compared with the method of storing existing packet data or the entire classification result with a large dimension. Only approximately 40 MB of memory is required to support 10 M concurrent flows due to the additional usage of approximately four bytes per flow. Therefore, the total memory size required to classify one flow is the sum of four bytes from the flow entry and the packet-buffer size to store the current received packet.

Second, the proposed method verifies the reliability of the classification results of the received packet using a validation model, which is implemented as a one-class model and determines whether to trust the classification results. If the validation model is determined to be trustworthy, the received packet and corresponding flow are processed according to the results of the packet-classification model. However, if the validation model determines the result is unreliable, then the result of the packet-classification model is stored for the classification of the next packet, and the received packet is forwarded.

Finally, in the proposed method, all received packets are classified through a packet classifier and validation model; however, even the last packet of a flow can be determined as unreliable by the validation model. In this case, the classification result with the highest probability for all packets was selected as the final result. Classification failure is avoided by selecting the class with the highest probability among unreliable results. Therefore, if the validation model determines that the classification result for the currently received packet is unreliable, the probability value of the previously stored classification result is compared with that of the current result. The previously stored classification result is updated as the current classification result only when the current probability value is high. Each module of the proposed method is described in detail. Algorithm 1 shows how the proposed method classifies a flow for packets belonging to a flow. It sequentially shows how the Inception-ResNet classifier uses each packet and preclassified class as input features, validates the classifier's results with DeepSVDD, and finally returns the validated classification result or the classification result with the highest score for the flow. Each module of the proposed method is further described in detail.

Algorithm 1: Proposed Flow Classification

Input: Packets belonging to a flow, $\{P_1, P_2, \cdots, P_n\}$
Output: Class ID for the flow, R_{class}

```
stored_R_score = 0
stored_ R_class = Null
For i = 1 to n
    R_class, R_score = Inception_ResNet(P_i, stored_R_class)
    R_valid_score = DeepSVDD(P_i)
    If R_valid_score > Threshold
        Return R_class
    EndIf
    If R_score > stored_R_score
        stored_R_score= R_score
        stored_R_class= R_class
    EndIf
EndFor
Return R_class
```

3.1. Packet-Classification Model

The packet-classification model consists of a stem block, Inception-ResNet module, reduction module, and FCN. The Inception-ResNet v1 model has a small number of parameters and is suitable for NIDS since it has high accuracy and improved learning speed compared with existing inception models [21]. In the Inception-ResNet module, convolutional filters of various sizes are combined with residual connections. Such connections not only improve the structure-induced performance degradation of DL but also reduce the overall training time. The following figure details the structure of the packet-classification model.

As displayed in Figure 2, the Inception-ResNet module consists of A, B, and C, and the reduction module consists of A and B. Inception-ResNet modules A and C were connected in a three-unit structure, whereas modules B and B were connected in a five-unit structure. The reduction module was placed between the Inception-ResNet modules to reduce the data dimension. Each module of the packet-classification model is described in detail.

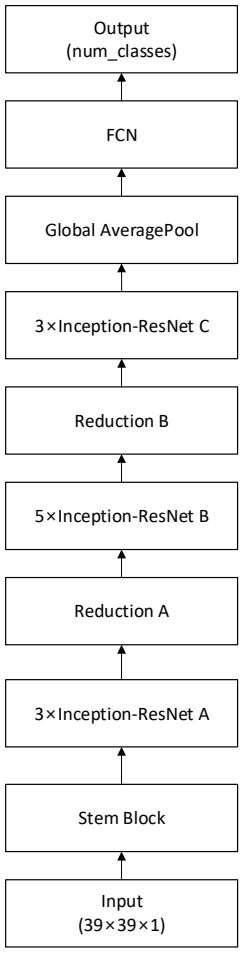


Figure 2. Detailed structure of packet-classification model.

3.1.1. Stem Block

As displayed in Figure 3, the stem block is the data-input part of the model and receives 39 × 39 × 1-dimension data. By contrast, the output of the stem block is in the form of 18 × 18 × 256. A block comprises four 2D convolution layers, all of which use ReLU as the activation function. The stride is set to two in the last convolution layer, which is the output part of the stem block, and to one layer in the other layers. Figure 3 displays the structure of the entire layer. The right-hand side of the figure shows the output data dimensions of each layer. All convolution layers of the stem block set the padding option to be valid (V).

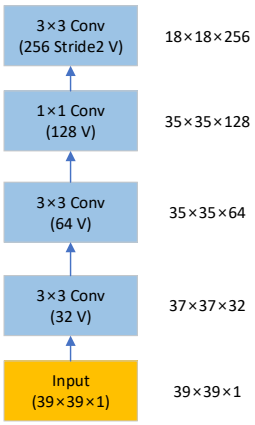


Figure 3. Stem block.

3.1.2. Inception-ResNet Module and Reduction Module

As displayed in Figure 2, three Inception-ResNet modules exist in the proposed packet-classification model. All Inception-ResNet A modules had the same input and output data types. The reduction module is connected to the Inception-ResNet module and reduces the data dimension.

3.1.3. Inception-ResNet A

The Inception-ResNet A module uses the stem output in the packet-classification model as the input (referred to as $\text{input}_{\text{InceptionA}}$). Therefore, both the input and output of module A have dimensions of $18 \times 18 \times 256$. The input of the Inception-ResNet A module was replicated into three types of convolutional hidden layers and used as input values. The three output values were used as inputs to the convolution layer without an activation function after passing through the concatenate layer. The output value of this convolutional layer performed an additional operation with the first input value of the module. The final result of the module was the value obtained by applying the ReLU function to the result of the addition operation [22]. All three types of convolutional hidden layers in the module use ReLU as an activation function and the same option for padding. In this study, three Inception-ResNet modules were stacked and used. Figure 4 displays the structure of the module.

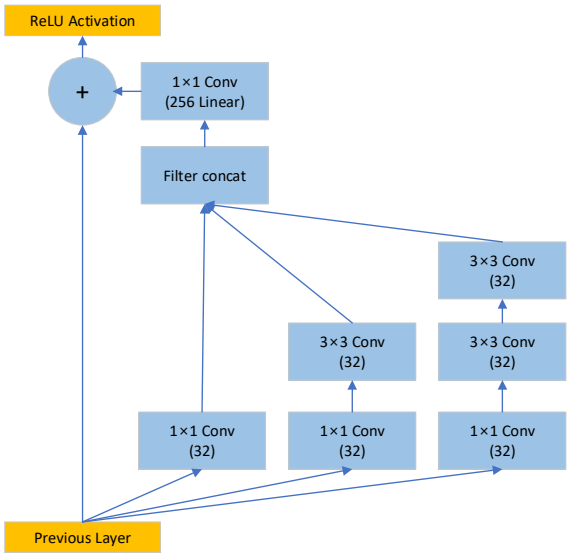


Figure 4. Inception-ResNet A module.

Let us say that the input of the module is $\text{input}_{\text{InceptionA}}$, and the output is $\text{output}_{\text{InceptionA}}$. The outputs of the three hidden layers are expressed as follows

$h_1 = \text{Conv}(1 \times 1,\ 32)(\text{input}_{\text{InceptionA}})$,

$h_2 = \text{Conv}(3 \times 3,\ 32)(\text{Conv}(1 \times 1,\ 32)(\text{input}_{\text{InceptionA}}))$,

$h_3 = \text{Conv}(3 \times 3,\ 32)(\text{Conv}(3 \times 3,\ 32)(\text{Conv}(1 \times 1,\ 32)(\text{input}_{\text{InceptionA}})))$

The ReLU activation function was applied to the convolution layers of h_1, h_2, and h_3. If the output of the convolution layer without an activation function is $\text{output}_{\text{conv\_linear}}$, then it is defined as follows:

$$\text{output}_{\text{conv}_{\text{linear}}} = \text{Conv}(1 \times 1,\ 256)(\text{Concat}(h_1, h_2,\ h_3))$$

Finally, $\text{output}_{\text{InceptionA}}$ is formulated as follows:

$$\text{output}_{\text{InceptionA}} = \text{ReLU}(\text{input}_{\text{InceptionA}} + \text{output}_{\text{conv}_{\text{linear}}})$$

3.1.4. Reduction A

The Reduction A module is connected to the output after repeating the Inception-ResNet A module three times, which reduces the form of the data from 18×18, the output form of Inception-ResNet A, to 8×8. Inside the module, the input data are duplicated and transmitted to three types of hidden layers, and the output of each hidden layer is merged and output through the concatenated layer. As displayed in Figure 5, the input shape was $18 \times 18 \times 256$, and the output shape was $8 \times 8 \times 896$.

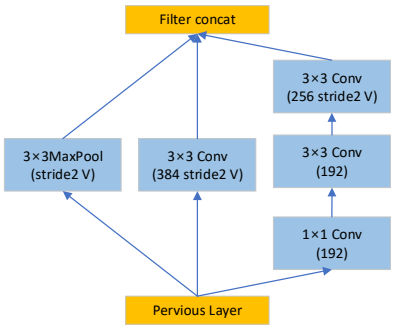


Figure 5. Reduction A module.

Let us assume that the module input is $\text{input}_{\text{ReductionA}}$ and the module output is $\text{output}_{\text{ReductionA}}$. The hidden layers are expressed as follows:

$H_1 = \text{MaxPool}(3 \times 3,\ \text{stride}\ 2,\ \text{valid})$,

$H_2 = \text{Conv}(3 \times 3,\ 384,\ \text{stride}\ 2,\ \text{valid})$,

$H_3 = \text{Conv}(3 \times 3,\ 256,\ \text{stride}\ 2,\ \text{valid})((\text{Conv}(3 \times 3,\ 192)(\text{Conv}(1 \times 1,\ 192)))$

The output of Reduction A is expressed as follows:

$$\text{output}_{\text{ReductionA}} = \text{Concat}(H_1(\text{input}_{\text{ReductionA}}),\ H_2(\text{input}_{\text{ReductionA}}),\ H_3(\text{input}_{\text{ReductionA}}))$$

3.1.5. Inception-ResNet B

The Inception-ResNet B module provides the output of the Reduction A module as the input in the packet-classification model. Therefore, the input and output of module B had the same $8 \times 8 \times 896$ shape. The input of the Inception-ResNet B module was replicated into two types of convolutional hidden layers and used as input values. Subsequently, the two output values were used as the input values of the convolutional layer without an activation function after passing through the concatenate layer, and the output values become the input values of the module and the addition operation. The result of the module was the value obtained by applying the ReLU activation function to the result of the addition operation. In all three types of convolutional hidden layers in the module.

ReLU was used as an activation function and the same option for padding. Five Inception-ResNet B modules were stacked and used. Figure 6 illustrates the structure of the module.

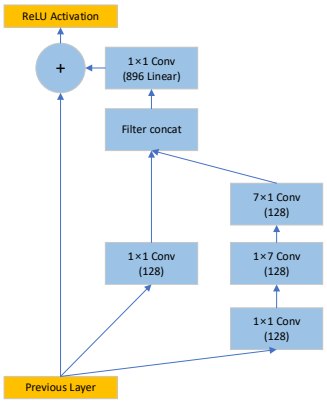


Figure 6. Inception-ResNet B.

3.1.6. Reduction B Module

The Reduction B module is connected to the output after repeating the Inception-ResNet B module five times and reduces the output of Inception-ResNet B from 8×8 to 3×3. Inside the module, input data are duplicated and transmitted to four types of hidden layers, and the output of each hidden layer is merged and output through the concatenate layer. As displayed in Figure 7, the input and output shapes of the module were $8 \times 8 \times 896$ and $3 \times 3 \times 1792$, respectively.

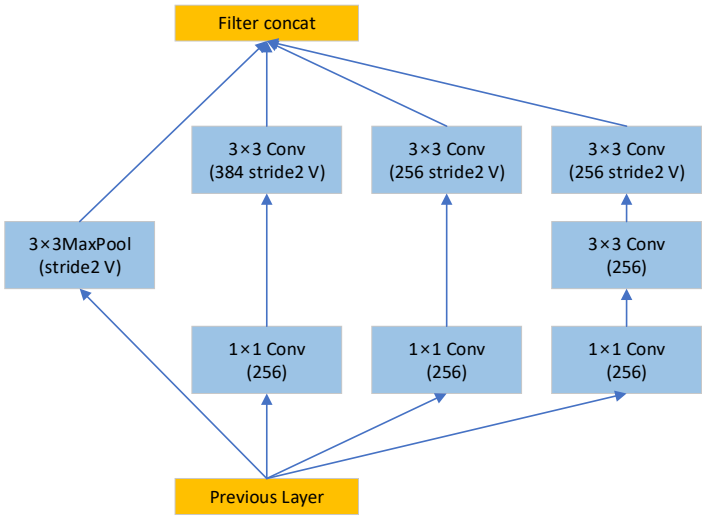


Figure 7. Reduction B.

3.1.7. Inception-ResNet C Module

In the Inception-ResNet C module, the output of the reduction module is used as the input to the packet-classification model. Therefore, the input and output of module C had the same shape of $3 \times 3 \times 1792$. The input of the Inception-ResNet C module was replicated with two types of convolutional hidden layers and used as input values. The two output values were used as input values for the convolution layer without an activation function after passing through the concatenated layer. The output values of the entire module were added to the input values of the module. The final result of the module is the value of applying the ReLU activation function after addition. All four types of convolutional hidden layers in the module used ReLU as an activation function as well as the same option for padding. Three Inception-ResNet C modules were stacked and used.

Figure 8 illustrates the overall structure. In the convolution layer, the ReLU activation function was used, whereas the convolution layer, after the filter concat layer, did not use the activation function.

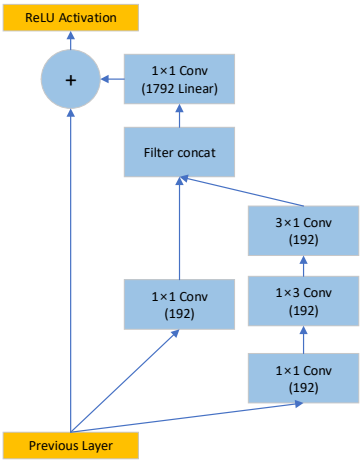


Figure 8. Inception-ResNet C.

3.1.8. FCN Block

The FCN block was in the output part of the packet-classification model. After Inception-ResNet C, the shape of the input data, $3 \times 3 \times 1792$, was converted to 1792 using the global average pooling layer before the data was delivered to the FCNs as shown in Figure 9 [23]. After passing through the four dense layers after the pooling layer, the result revealed whether the current packet is normal or malicious. The internal dropout rate was 20%, and the dense layer used the ReLU activation function and had as many outputs as the number of labels in the classification dataset. Finally, the Softmax activation function was used [24].

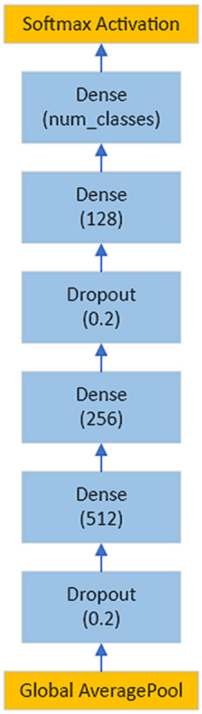


Figure 9. Structure of a fully connected network block.

3.2. Validation Model

The validation model determines the reliability of the result of the packet-classification model and determines whether to classify the corresponding flow according to the classification result of the current packet or postpone the classification until the next packet is received. Specifically, the validation model assists the packet classifier in classifying a flow with high accuracy by deferring classification to early packets that are likely to be misjudged.

In the proposed method, the validation model is implemented using DeepSVDD, a DL-based one-class classifier [25]. When DeepSVDD is trained with data consisting of one class, DeepSVDD's DNN is trained such that the data are mapped as close to the central point of the hypersphere as possible. Therefore, data other than those in the learned class tended to be outside the hypersphere. Using these characteristics, the proposed method determines whether the classified result from the packet classifier is reliable by learning DeepSVDD using packets misclassified by the packet-classification model.

The input data of DeepSVDD have a total size of 1822 bytes by adding the Inception output value (1792 bytes) of the packet-classification model and the header (30 bytes) of the current packet.

$$\text{input}_{\text{DeepSVDD}} = \text{Concat}(\text{GlobalMaxPool}(i_r),\ p_r),$$

where the current packet is p_r and the Inception output value is i_r (shape $= 3 \times 3 \times 1792$). Figure 10 illustrates the detailed DNN structure of DeepSVDD [26].

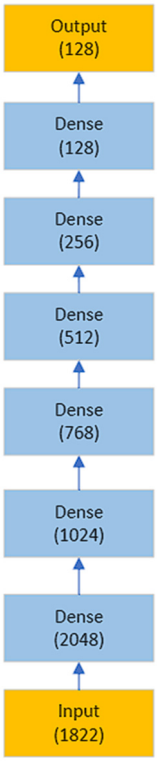


Figure 10. Structure of the DNN of DeepSVDD.

After the packet-classification model completes learning, it classifies the training data and generates training data for verification ($x_{train_{fail}}$), which comprises misclassified data. The radius of the hypersphere is selected such that the validation model that has learned the validation model with this training data has the highest F1-score.

4. Performance Evaluation

To analyze the performance of the proposed method in detail, two datasets, namely CIC-IDS2017 and CSE-IDS2018, were used and compared with existing algorithms in terms of memory usage, detection accuracy, and detection time [27]. In the conventional method, the rotation forest, which is a representative of the ML algorithm, along with DNN and CNN, which are the most well-known DL models, were selected [28,29]. We will describe the experimental environment and show the comparison results for the performance of each algorithm in terms of memory requirement, detection accuracy, and detection time.

4.1. Evaluation Environment

The number of flows and packets for each dataset are shown in Tables 4 and 5. Source IP, destination IP, source port, and destination port fields were removed to avoid generating biased trained models from all datasets. For each dataset, training, validation, and test datasets were created at a ratio of 6:2:2 to evaluate the performance. We only show the results for test datasets due to the similarity between each result, and we will show other results when needed. The packets stored in the packet capture file included in each dataset were classified into each flow according to the quintuple using CICFlowmeter, and all packets belonging to the same flow were assigned the same label as that of the flow [30].

Table 4. CIC-IDS2017 dataset.

Label	Total Number of Flows	Total Number of Packets
Benign	20,000	128,515
Bot	1500	7485
DDoS	10,000	45,315
DoS GoldenEye	2500	5295
DoS Hulk	15,000	38,630
DoS Slowhttptest	3500	25,360
DoS slowloris	3500	28,605
FTP-Patator	5000	66,980
PortScan	15,000	30,250
SSH-Patator	3500	98,130
Web Attack Brute Force	1000	17,695
Total	80,500	492,260

Table 5. CSE-CIC-IDS2018 dataset.

Label	Total Number of Flows	Total Number of Packets
Benign	20,000	163,535
Bot	12,000	55,475
BruteForce-FTP	8500	17,000
BruteForce-SSH	8500	201,380
BruteForce-WEB	500	33,445
DDOS-HOIC	12,000	37,700
DDoS-LOIC-HTTP	12,000	215,240
DoS-GoldenEye	6500	39,150
DoS-Hulk	12,000	29,440
DoS-SlowHTTPTest	8500	17,000
DoS-Slowloris	5000	49,365
Total	105,500	858,730

The detailed learning method for the proposed model is as follows: the cross-entropy loss model is used in the packet-classification model, and the ReduceLROnPlateau callback function is used to dynamically adjust the learning rate. The learning rate starts from 1×10^{-4}, and if the learning accuracy does not improve, it decreases by 0.1 to 1×10^{-7}. ADAM was used as the optimization function for learning, and the batch size was set to 256. In the validation model, the batch size was set to 256, which was the same as that of

the packet-classification model, and the learning rate was set to 1×10^{-4}. The validation model also used ADAM as the optimization function.

4.2. Memory Requirements

Since the existing method creates a feature for a flow after storing all packets, a space equal to the total traffic of the flow per flow is used. However, the proposed method requires only four bytes to store the class and probability information to be used when classification fails for all packets as well as memory to process the currently received packet. Therefore, the memory required per flow is equal to the size of the largest packet in the flow. Figures 11 and 12 display the memory usage of the proposed method and the existing method according to the flow length.

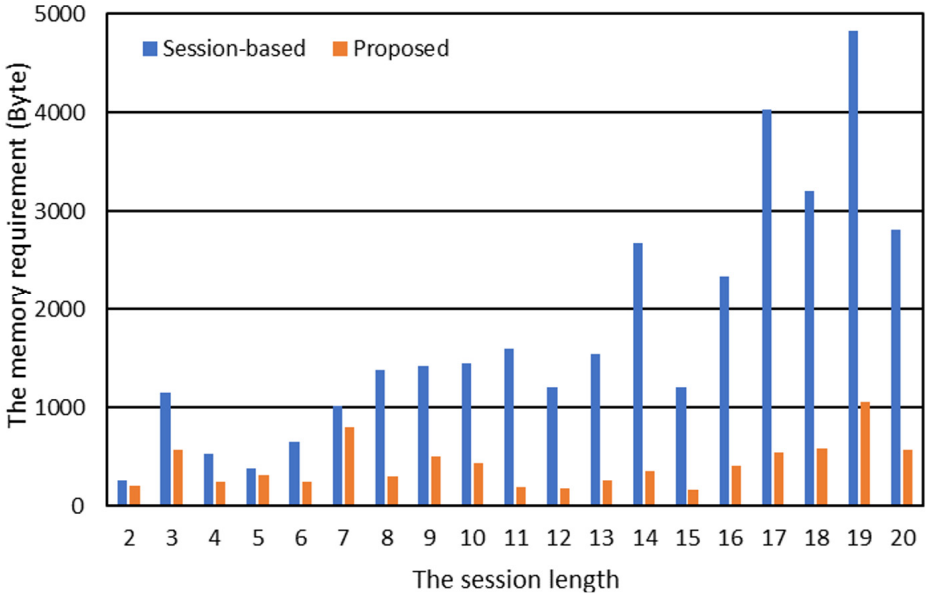


Figure 11. Average memory requirement per flow according to the flow length for the CIC-IDS2017 dataset.

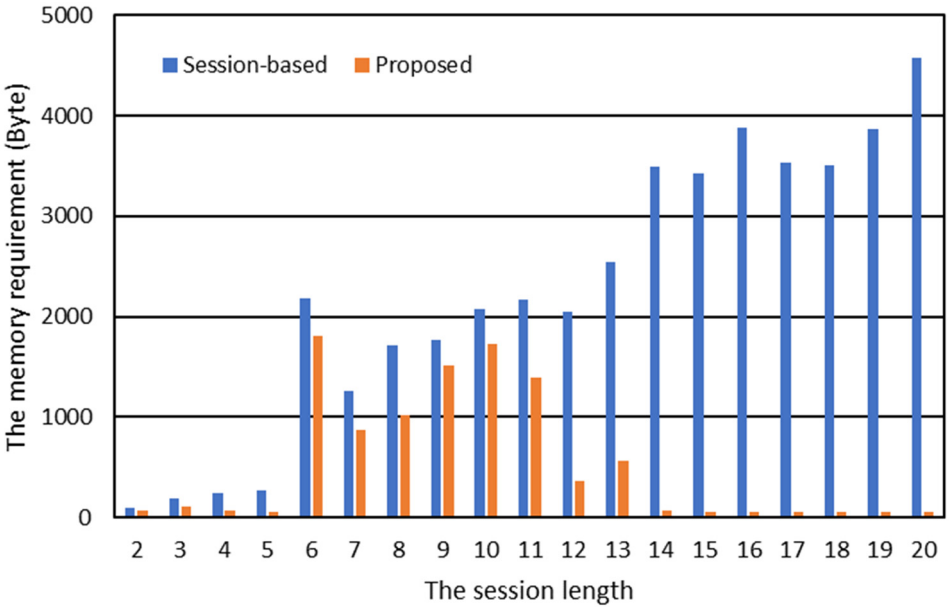


Figure 12. Average memory requirement per flow according to the flow length for the CSE-CIC-IDS2018 dataset.

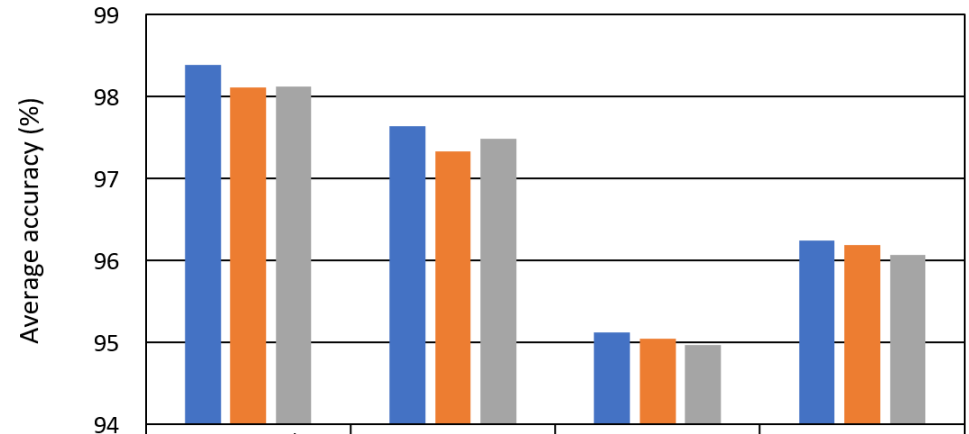
In Figure 11, the existing method consumes 351 bytes on average for entire flow lengths, while the proposed method requires only 80 bytes on average for the CIC-IDS2017 dataset. In Figure 12, the existing and proposed methods consume 450 bytes and 103 bytes on average, respectively, for CSE-CIC-IDS2018. Therefore, we can conclude that our approach reduces the memory requirement by 77% on average.

As displayed in Figures 11 and 12, the average amount of memory used according to the flow length tends to increase overall in the case of the existing algorithms. However, the proposed method does not exceed a certain size because the maximum memory requirement per flow is the largest single packet size within the flow. The amount of memory required to process one flow is critical for processing numerous concurrent flows. Therefore, the proposed method is advantageous for processing large-capacity concurrent flows since it can support the same concurrent flows with less memory.

In addition, the proposed method is advantageous for parallel processing. In a parallel-processing-based NIDS, where several proposed classifiers run on different processors, we assume that one classifier processes one received packet, and the classifier requires a classification result for the previous packet. The synchronization of the classification result of the previous packet is required between processors, and the overhead is small since the amount of data to be synchronized in the proposed method is as small as one word. Therefore, unlike other existing methods, the proposed method has excellent scalability since it is optimized for parallel processing.

4.3. Detection Rate

Figure 13 shows the average classification accuracy for the learning, validation, and test datasets when using CIC-IDS2017 and CSE-CIC-IDS2018. All NIDSs provide similar performance for the three datasets, indicating that all classifiers used in the performance evaluation are well trained. In addition, it is confirmed that the proposed method can provide very stable and high detection performance by showing the highest accuracy regardless of the type of dataset.

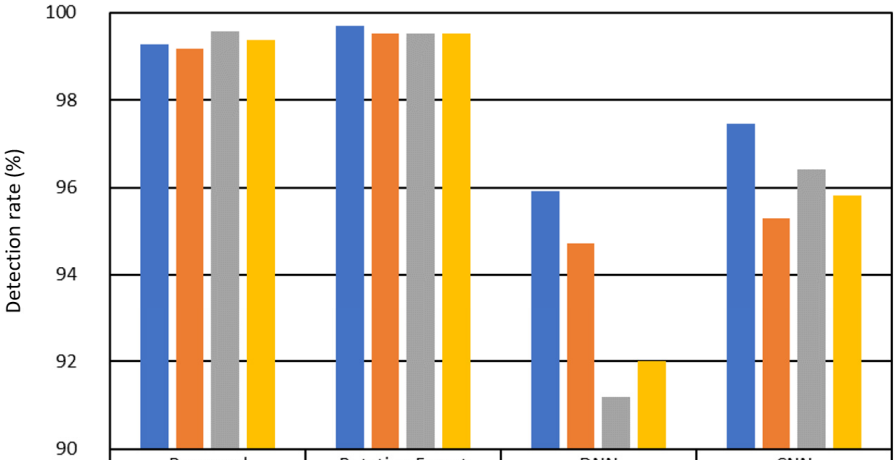


	Proposed	Rotation Forest	DNN	CNN
Training	98.39%	97.64%	95.12%	96.24%
Validation	98.11%	97.33%	95.05%	96.19%
Test	98.13%	97.48%	94.97%	96.06%

Figure 13. Average intrusion detection rate of training, validation, and test dataset for the CIC-IDS2017 and CSE-CIC-IDS2018.

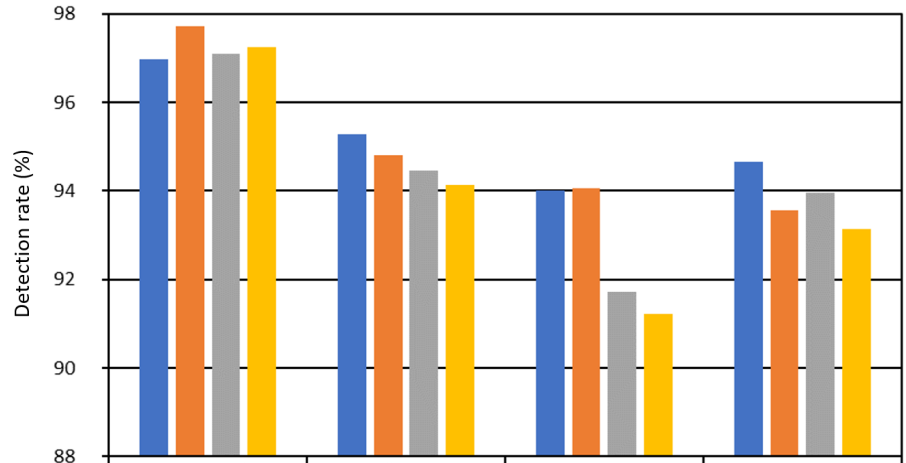
Detecting intrusions using packet features is more challenging than detecting intrusions using flow features. Moreover, achieving a high detection performance is difficult compared with the conventional method that detects intrusion using all packets in a flow since the proposed method detects intrusion for each received packet. Nevertheless,

Figures 14 and 15 reveal that the proposed method achieves an equivalent or higher detection accuracy than the existing methods. In particular, CSE-CIC-IDS2018 achieved the highest F1 score with a large performance difference. Therefore, on average as shown in Figure 16, an F1-score that is at least 1.5% higher than those of the existing three methods, namely rotation forest, DNN, and CNN, was achieved. This result indicated that the proposed Inception-ResNet-based packet classification and validation models are effective for intrusion detection.



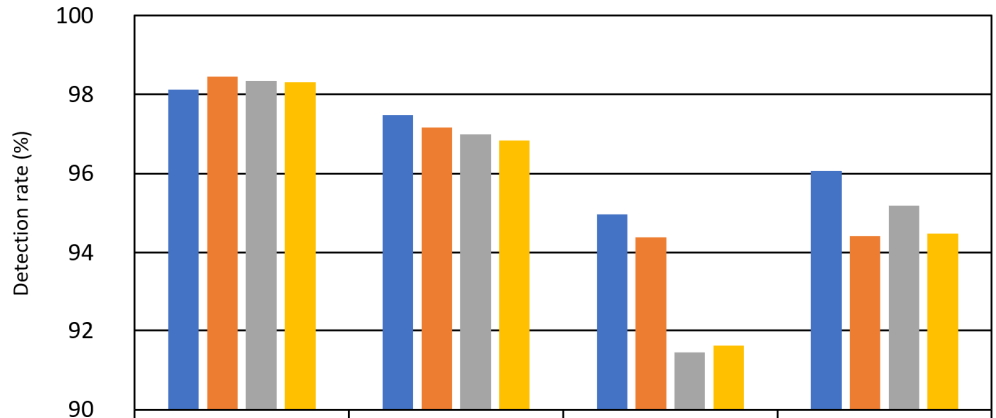
	Proposed	Rotation Forest	DNN	CNN
Accuracy	99.27%	99.69%	95.93%	97.46%
Precision	99.18%	99.53%	94.72%	95.29%
Recall	99.59%	99.52%	91.20%	96.41%
F1-Score	99.38%	99.53%	92.02%	95.83%

Figure 14. Average intrusion detection rate for the CIC-IDS2017 dataset.



	Proposed	Rotation Forest	DNN	CNN
Accuracy	96.99%	95.28%	94.00%	94.67%
Precision	97.73%	94.81%	94.06%	93.55%
Recall	97.11%	94.45%	91.71%	93.97%
F1-Score	97.24%	94.14%	91.22%	93.13%

Figure 15. Average intrusion detection rate for the CSE-CIC-IDS2018 dataset.



	Proposed	Rotation Forest	DNN	CNN
Accuracy	98.13%	97.48%	94.97%	96.06%
Precision	98.46%	97.17%	94.39%	94.42%
Recall	98.35%	96.99%	91.46%	95.19%
F1-Score	98.31%	96.83%	91.62%	94.48%

Figure 16. Average intrusion detection rate for all datasets.

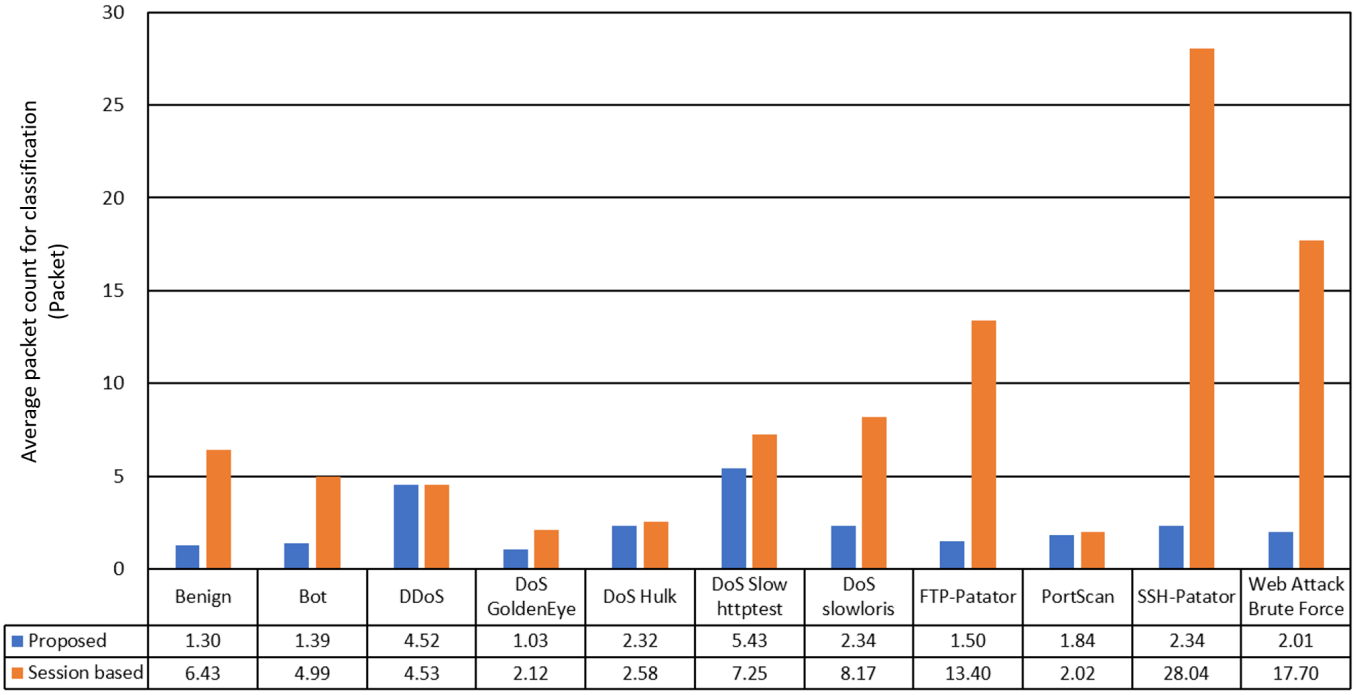
4.4. Detection Time

Because the detection time is a value that depends on interpacket arrival time, this parameter is not suitable for performance comparison. For example, suppose an intrusion is detected 10 s after the flow starts. If the interpacket arrival time of the flow is reduced in half, the flow will be detected in 5 s.

Therefore, in this experiment, the number of packets received until classifying the flow was compared as a new metric for the detection time. Let us call the metric average packet count. Since the average packet count is independent of the interpacket arrival time, it is more advantageous to compare and analyze detection times. The smaller average packet count means that the NIDS has a shorter detection time.

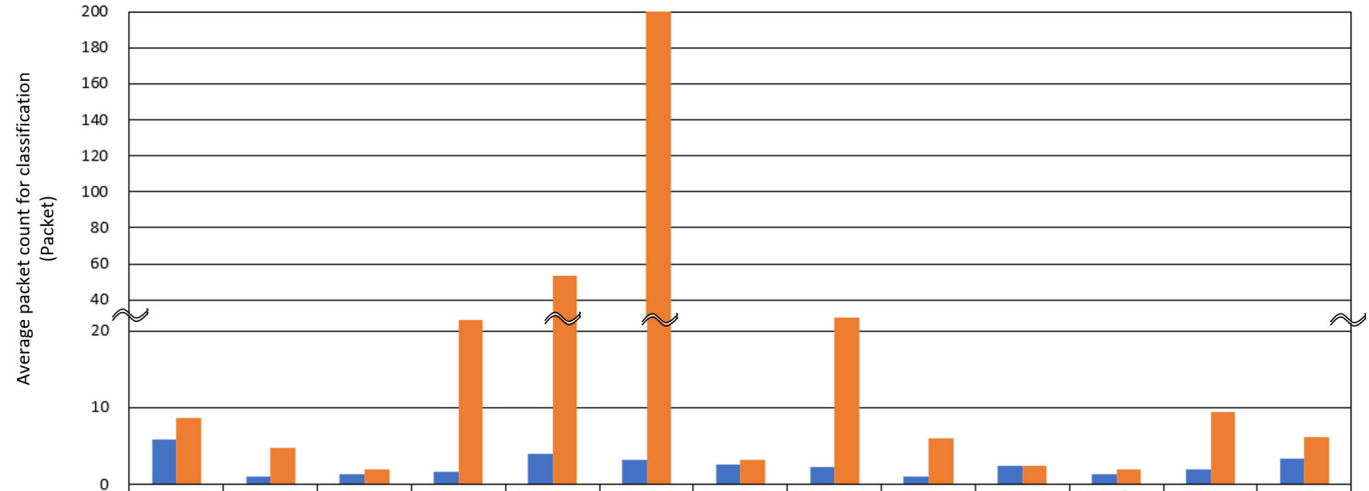
Figures 17 and 18 detail the detection times of the existing flow-based method and the proposed method for each class. With the exception of a few classes, the proposed method can detect most intrusions using only three or fewer packets. In the case of the SSH-Patator of CIC-IDS2017, the proposed method detects 12 times faster than the existing flow-based methods. Furthermore, the proposed method detected BruteForce XSS 63 times faster in the CSE-CIC-IDS2018 dataset. For the entire datasets for CIC-IDS2017 and CSE-CIC-IDS2018, the proposed method reduces the detection time by 73% and 92% on average, respectively, compared to the existing method.

Unlike existing NIDS, the proposed method performs intrusion detection on every incoming packet, which improves detection time. However, flow classification based on a single packet is prone to false detection since the amount of information used for classification is insufficient. To solve this problem, the proposed method uses the classification result of the previous packet to classify the current packet and verify the classification result again to prevent accuracy degradation. Considering that the proposed method shows a detection rate almost similar to that of the rotation forest in previous experiments, the proposed approach has a strong advantage in detecting intrusions without delay, avoiding sacrificing the detection rate.



	Benign	Bot	DDoS	DoS GoldenEye	DoS Hulk	DoS Slow httptest	DoS slowloris	FTP-Patator	PortScan	SSH-Patator	Web Attack Brute Force
Proposed	1.30	1.39	4.52	1.03	2.32	5.43	2.34	1.50	1.84	2.34	2.01
Session based	6.43	4.99	4.53	2.12	2.58	7.25	8.17	13.40	2.02	28.04	17.70

Figure 17. Average packet count required for the intrusion detection of each class for the CIC-IDS2017 dataset.



	Benign	Bot	BruteForce-FTP	BruteForce-SSH	BruteForce-WEB	BruteForce-XSS	DDOS-HOIC	DDoS-LOIC-HTTP	DoS GoldenEye	DoS Hulk	DoS Slow HTTPTest	DoS Slowloris	SQL-Injection
Proposed	5.85	1.00	1.30	1.70	4.02	3.25	2.50	2.33	1.00	2.44	1.35	1.93	3.33
Session based	8.58	4.69	2.00	21.43	53.50	205.18	3.25	23.16	6.07	2.48	2.00	9.44	6.20

Figure 18. Average packet count required for intrusion detection of each class for the CSE-CIC-IDS2018 dataset.

5. Conclusions

Detecting intrusions in every received packet and simultaneously minimizing the amount of memory required to process each flow are critical to detect an intrusion in real time in NIDSs. Although studies have been conducted on improving the detection accuracy of the existing ML-based NIDS, few studies have focused on the short detection time, or the amount of memory, required for detection. Therefore, in this study, we proposed a DL-based NIDS that supports fast detection and minimizes memory requirements. The proposed method has the same or higher detection accuracy than existing methods. Fur-

thermore, the proposed method can protect the network since it has a very-short detection time that simultaneously detects intrusions within three packets on average. Furthermore, existing flow-based NIDSs require a large amount of memory to store packet data, whereas the proposed method requires only one-word memory to store information on past traffic. Moreover, the proposed method can significantly increase the maximum number of simultaneous flows handled by one NIDS since the memory requirement for each flow is constant, regardless of the length of the flow. As the network size and number of users increase, the NIDS should handle more traffic and concurrent flows. Additionally, with the emergence of various security attacks, more sophisticated and complex classifiers are required for fast and accurate intrusion detection. Parallel processing is essential in the NIDS to support such high performance. The proposed method is suitable for parallel processing with a low synchronization overhead. Therefore, this study can resolve the technical problems associated with NIDSs.

Author Contributions: T.K. and W.P. wrote the manuscript and conducted research. All authors have read and agreed to the published version of the manuscript.

Funding: This study was supported by the National Research Foundation of Korea (NRF) NRF2022R1A2C1011774.

Data Availability Statement: The datasets utilized in this paper are CIC-IDS2017 dataset (https://www.unb.ca/cic/datasets/ids-2017.html (accessed on 13 April 2023)) and CSE-CIC-IDS2018 dataset (https://www.unb.ca/cic/datasets/ids-2018.html (accessed on 13 April 2023)).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Roesch, M. Snort: Lightweight intrusion detection for networks. In Proceedings of the LISA '99: 13th Systems Administration Conference USENIX, Seattle, WA, USA, 7–12 November 1999; pp. 229–238.
2. Wang, W.; Sheng, Y.; Wang, J.; Zeng, X.; Ye, X.; Huang, Y.; Zhu, M. HAST-IDS: Learning hierarchical spatial-temporal features using deep neural networks to improve intrusion detection. *IEEE Access* **2017**, *6*, 1792–1806. [CrossRef]
3. Seelammal, C.; Devi, K.V. Computational intelligence in intrusion detection system for snort log using Hadoop. In Proceedings of the International Conference on Control, Instrumentation, Communication and Computational Technologies ICCICCT, Kumaracoil, India, 16–17 December 2016; pp. 642–647. [CrossRef]
4. Bilge, L.; Dumitras, T. Before we knew it: An empirical study of zero-day attacks in the real world. In Proceedings of the 2012 ACM Conference on Computer and Communications Security, Raleigh, NC, USA, 16–18 October 2012; pp. 833–844. [CrossRef]
5. Al-Qatf, M.; Lasheng, Y.; Al-Habib, M.; Al-Sabahi, K. Deep learning approach combining sparse autoencoder with SVM for network intrusion detection. *IEEE Access* **2018**, *6*, 52843–52856. [CrossRef]
6. Ahmad, I.; Basheri, M.; Iqbal, M.J.; Rahim, A. Performance comparison of support vector machine, random forest, and extreme learning machine for intrusion detection. *IEEE Access* **2018**, *6*, 33789–33795. [CrossRef]
7. Belouch, M.; Elhadaj, S. Comparison of ensemble learning methods applied to network intrusion detection. In Proceedings of the ICC '17: Second International Conference on Internet of Things, Data and Cloud Computing, ACM ICC, 2012, Cambridge, UK, 22–23 March 2017; pp. 1–4. [CrossRef]
8. Hwang, R.-H.; Peng, M.-C.; Nguyen, V.-L.; Chang, Y.-L. An LSTM-Based Deep Learning Approach for Classifying Malicious Traffic at the Packet Level. *Appl. Sci.* **2019**, *9*, 3414. [CrossRef]
9. Ge, M.; Syed, N.F.; Fu, X.; Baig, Z.; Robles-Kelly, A. Towards a deep learning-driven intrusion detection approach for Internet of Things. *Comput. Networks* **2021**, *186*, 107784. [CrossRef]
10. Leevy, J.L.; Khoshgoftaar, T.M. A survey and analysis of intrusion detection models based on CSE-CIC-IDS2018 Big Data. *J. Big Data* **2020**, *7*, 104. [CrossRef]
11. Disha, R.A.; Waheed, S. Performance analysis of machine learning models for intrusion detection system using Gini Impurity-based Weighted Random Forest (GIWRF) feature selection technique. *Cybersecurity* **2022**, *5*, 1. [CrossRef]
12. Soheily-Khah, S.; Marteau, P.; Béchet, N. Intrusion Detection in Network Systems Through Hybrid Supervised and Unsupervised Machine Learning Process: A Case Study on the ISCX Dataset. In Proceedings of the 1st International Conference on Data Intelligence and Security (ICDIS), South Padre Island, TX, USA, 8–10 April 2018; pp. 219–226. [CrossRef]
13. Yuan, Y.; Huo, L.; Hogrefe, D. Two layers multi-class detection method for network intrusion detection system. In Proceedings of the IEEE Symposium on Computers and Communications ISCC, Heraklion, Greece, 3–6 July 2017; pp. 767–772. [CrossRef]
14. Vijayakumar, S.; Sannasi, G. Machine Learning Approach to Combat False Alarms in Wireless Intrusion Detection System. *Comput. Inf. Sci.* **2018**, *11*, 67–81. [CrossRef]

15. Protic, D.; Stankovic, M. Anomaly-Based Intrusion Detection: Feature Selection and Normalization Influence to the Machine Learning Models Accuracy. *Eur. J. Eng. Form. Sci.* **2019**, *2*, 101–106. [CrossRef]
16. Dhanabal, L.; Shantharajah, S.P. A Study on NSL-KDD Dataset for Intrusion Detection System Based on Classification Algorithms. *Int. J. Adv. Res. Comput. Commun. Eng.* **2015**, *4*, 446–452.
17. Kim, T.; Pak, W. Hybrid Classification for High-Speed and High-Accuracy Network Intrusion Detection System. *IEEE Access* **2021**, *9*, 83806–83817. [CrossRef]
18. Farooq, M.S.; Abbas, S.; Sultan, A.K.; Khan, M.A.; Mosavi, A. A Fused Machine Learning Approach for Intrusion Detection System. *Comput. Mater. Contin.* **2023**, *74*, 2607–2623. [CrossRef]
19. Nasir, M.U.; Khan, S.; Mehmood, S.; Khan, M.A.; Zubair, M.; Hwang, S.O. Network Meddling Detection Using Machine Learning Empowered with Blockchain Technology. *Sensors* **2022**, *22*, 6755. [CrossRef]
20. Asif, M.; Abbas, S.; Khan, M.A.; Fatima, A.; Khan, M.A.; Lee, S.-W. MapReduce based intelligent model for intrusion detection using machine learning technique. *J. King Saud Univ. Comput. Inf. Sci.* **2022**, *34*, 9723–9731. [CrossRef]
21. Szegedy, C.; Ioffe, S.; Vanhoucke, V.; Alemi, A. Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning. *arXiv* **2016**, arXiv:1602.07261. [CrossRef]
22. Adigun, O.; Kosko, B. Deeper Neural Networks with Non-Vanishing Logistic Hidden Units: NoVa vs. ReLU Neurons. In Proceedings of the 20th IEEE International Conference on Machine Learning and Applications (ICMLA), Pasadena, CA, USA, 13–16 December 2021; pp. 1407–1412. [CrossRef]
23. Sainath, T.N.; Vinyals, O.; Senior, A.; Sak, H. Convolutional, Long Short-Term Memory, fully connected Deep Neural Networks. In Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP), South Brisbane, Australia, 19–24 April 2015; pp. 4580–4584. [CrossRef]
24. Goodfellow, I.; Bengio, Y.; Courville, A. Chapter 6.2.2.3 Softmax Units for Multinoulli Output Distributions. In *Deep Learning*; MIT Press: Cambridge, MA, USA, 2016; pp. 180–184.
25. Zhang, Z.; Deng, X. Anomaly detection using improved deep SVDD model with data structure preservation. *Pattern Recognit. Lett.* **2021**, *148*, 1–6. [CrossRef]
26. Gu, J.; Zhu, M.; Zhou, Z.; Zhang, F.; Lin, Z.; Zhang, Q.; Breternitz, M. Implementation and evaluation of deep neural networks (DNN) on mainstream heterogeneous systems. In Proceedings of the 2014 5th Asia-Pacific Workshop on Systems (APSys '14), Beijing, China, 25–26 June 2014; pp. 1–7. [CrossRef]
27. Sharafaldin, I.; Lashkari, A.H.; Ghorbani, A.A. Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization. In Proceedings of the 2018 4th International Conference on Information Systems Security and Privacy (ICISSP), Madeira, Portugal, 22–24 January 2018. [CrossRef]
28. Rodriguez, J.J.; Kuncheva, L.I.; Alonso, C.J. Rotation Forest: A New Classifier Ensemble Method. *IEEE Trans. Pattern Anal. Mach. Intell.* **2006**, *28*, 1619–1630. [CrossRef] [PubMed]
29. Yamashita, R.; Nishio, M.; Do, R.K.G.; Togashi, K. Convolutional neural networks: An overview and application in radiology. *Insights Imaging* **2018**, *9*, 611–629. [CrossRef] [PubMed]
30. Gil, G.D.; Lashkari, A.H.; Mamun, M.; Ghorbani, A.A. Characterization of Encrypted and VPN Traffic Using Time-Related Features. In Proceedings of the 2nd International Conference on Information Systems Security and Privacy (ICISSP), Rome, Italy, 19–21 February 2016; pp. 407–414. [CrossRef]