electronics

MDPI

Article

Ensemble of Deep Convolutional Learning Classifier System Based on Genetic Algorithm for Database Intrusion Detection

Seok-Jun Bu, Han-Bit Kang and Sung-Bae Cho *

Department of Computer Science, Yonsei University, Seoul 03722, Korea; sjbuhan@yonsei.ac.kr (S.-J.B.); prixt@yonsei.ac.kr (H.-B.K.)
* Correspondence: sbcho@yonsei.ac.kr

Abstract: Methods of applying deep learning to database protection have increased over the years. To secure role-based access control (RBAC) by learning the mapping function between query features and roles, it is known that the convolutional neural networks combined with learning classifier systems (LCS) can reach formidable accuracy. However, current methods are focused on using a singular model architecture and fail to fully exploit features that other models are capable of utilizing. Different deep architectures, such as ResNet and Inception, can exploit different spatial correlations within the feature space. In this paper, we propose an ensemble of multiple models with different deep convolutional architectures to improve the overall coverage of features used in role classification. By combining models with heterogeneous topologies, the ensemble-LCS model shows significantly increased performance compared to previous single architecture LCS models and achieves better robustness in the case of training data imbalance.

Keywords: ensemble classifier; deep learning; genetic algorithm; learning classifier system; role-based access control; database intrusion detection

check for updates

Citation: Bu, S.-J.; Kang, H.-B.; Cho, S.-B. Ensemble of Deep Convolutional Learning Classifier System Based on Genetic Algorithm for Database Intrusion Detection. *Electronics* **2022**, *11*, 745. https://doi.org/10.3390/electronics11050745

Academic Editor: Aryya Gangopadhyay

Received: 8 July 2021
Accepted: 26 February 2022
Published: 28 February 2022

1. Introduction

Database security is a constantly changing field, with attackers searching and exploiting weak points in current defense measures and defenders that develop new methods to protect the database from newly discovered exploits and trying to fortify security measures to future threats [1]. Various methods have already been developed, ranging from simple methods, such as password protection, creating policies that determine the limits of access to the user, and more recent, advanced methods that try to implement the growing field of deep learning [2,3].

Well-known threats to relational databases can be classified into outsider attacks and insider attacks [4]. Outsider attacks, such as SQL injections, can be relatively easily detected with traditional methods. Insider attacks are malicious actions or transactions that are carried out by actors with legitimate access within the infrastructure [5,6]. Insider attacks are more challenging to model and detect [7,8], as both individual and sequential patterns of the users' queries must be identified [9]. Role-based access control (RBAC) attempts to model user-database access by assigning roles to each user and limiting what each role can access. As such, by identifying which role the user has and classifying what role the user query would require, one can develop an intruder detection system (IDS) by verifying whether the user's role and query's role match, as shown in Figure 1.

Traditionally, the query role classification model was designed using hard-coded query grammar analysis [10], often requiring an understanding of the underlying database and transaction structure to model the characteristics of queries. More sophisticated methods, such as using multiple models for parsing SQL queries, have also been applied [11]. Recently, with the development of neural networks, deep learning methods has been applied to this domain, with promising results. In combination with neural network-based

classification models, Learning Classifier Systems (LCS) that use a genetic algorithm for feature selection [12] have been used to increase the adaptability of IDS to account for changing usage patterns of both legitimate and illegitimate users.

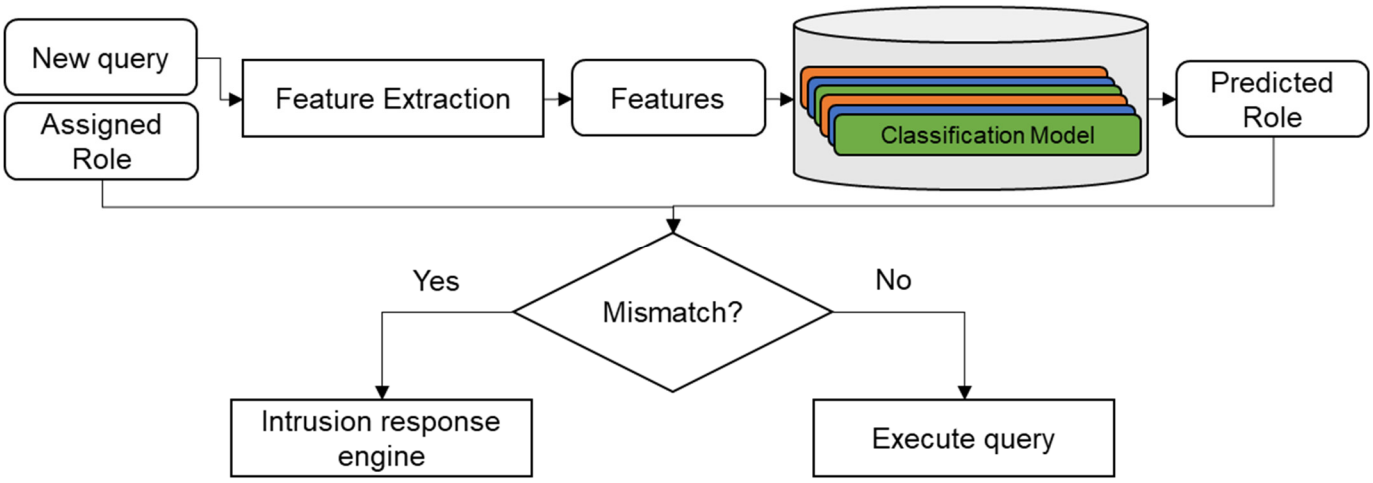

Figure 1. Diagram of an intruder detection system based on role classification.

However, even with such advances, current methods still have limitations; recent models are built on single model architectures and, therefore, cannot provide coverage for all spatial relations within the feature space. In this paper, we propose a method that mitigates this shortcoming by using multiple model architectures in conjunctions, thus increasing the system's feature space coverage. The main contributions of this paper are as follows:

- We utilize models of different architectures and their tendency to associate with TPC-E schema roles categories by collecting heterogeneous models into an ensemble.
- We exploit the effect of model ensembles targeting both learning and adaptability for RBAC-based intrusion detection.

2. Related Works

Table 1 summarizes related works on applying machine learning for SQL transaction role classification.

Before queries can be classified, it is necessary to extract salient features from diverse query inputs. Ronao et al. [4] used PCA to analyze various potential queries and managed to extract a set number of features that can be used by other classification models, such as Random Forest Ensembles. Based on these features, other classification models, such as SVM [13], MLP [14], and CNN-LSTM [15], have been used to develop more advanced classifications methods.

CNNs, compared to other methods, showed the most promising accuracy. However, human inputs and data collected in real-world environments are often incomplete or skewed. Query characteristics may also change over time. As such, further optimization of CNN models is necessary.

Evolutionary algorithms are often utilized to optimize hyperparameters of the model to improve the overall accuracy of the model. Khare et al. [16] used Spider Monkey Optimization to reduce the dimensionality of complex input data, improving the neural network's classification accuracy for the NSL-KDD dataset. Selvakumar et al. [17] used the Firefly algorithm for dimensionality reduction on the KDD CUP 99 dataset with Bayesian Networks for network anomaly detection.

Table 1. Related works on machine learning for the SQL transaction role classification.

Approach	Detection Mechanism	Optimization Method	Objective	Accuracy
Machine Learning	PCA, Random Forest [4]	-	Query feature extraction, role classification tree ensemble	0.7682
	SVM [13]		Learn kernel-based query feature selection boundary	0.7800
	MLP [14]		Classification based on query features	0.8002
	CNN-LSTM [15]		Learn spatial relationship of query features	0.9265
CNN Optimization	CNN	ERL [18]	Optimization of learning process	0.9450
		LCS [19]	Optimization of feature selection	0.9255
		PSO [20]	Optimization of network topology	0.9400
	CNN Ensemble	GA [21]	Optimization of ensemble rules	0.9663
	CNN, Inception, ResNet	LCS [22]	Compare various GA optimized CNN models	0.9200 0.9527 0.9427

The combination of the evolutionary algorithm and CNNs can also be found in dataset query role classification research. Choi et al. [18] combined evolutionary reinforcement learning to optimize the learning process of CNNs, attempting to find the ideal learning rate for each training input using the evolutionary process. Bu et al. [19] combined the CNN model with a Learning Classifier System (CN-LCS), allowing the model to learn which input features to focus on to improve model accuracy. Kim et al. [20] used Particle Swarm Optimization to discover improved model topologies. The evolutionary algorithm can also be used in ensemble optimization, such as in the work by Bu et al. [21] to optimize ensembles of CN-LCS.

The CN-LCS method can be expanded for more advanced CNN architectures. Kim et al. [22] expanded the CN-LCS method for classification networks inspired by advanced image classification models, such as Inception Networks [23] (Inception) and Residual Networks [24] (ResNet), observing improved accuracy for models based on more advanced architecture and complexity.

The improvement of model architecture in tandem with additional optimization via evolutionary algorithms showed increased accuracy of the system. However, most research has focused on the optimization of a single model. Even in research where ensembles were used, it used a single type of architecture. We utilized multiple types of CNN model architectures, increasing the diversity of models used by the ensemble. This expands the feature analyzing the capability of the classification system.

3. The Proposed Method

The proposed method exploits the different capabilities of diverse CNN architectures by combining heterogeneous models into an ensemble. It is known that the intrusion detection system accuracy can be enhanced by utilizing the ensemble of multiple models [25,26]. The method can be divided into three components: the dataset generation and pre-processing component generates the relevant dataset based on specification and converts the raw input queries into model readable features; model generation uses a genetic algorithm to generate a population of different query feature selections and generate a corresponding population of models; finally, the ensemble collection component uses the results of each generated models to select and combine relevant models into an ensemble with respect to accuracy and diversity.

3.1. Dataset Generation and Pre-Processing

Before generating the models, it is necessary to generate the SQL transaction data that will be used during model training. Synthetic queries are generated that model the roles of the SQL transactions. These queries will mimic and model the transactions that users will have with the database in realistic scenarios while having the advantage of being balanced by class.

The TPC Benchmark E (TPC-E) [27] is an online transaction processing (OLTP) benchmark created by the Transaction Processing Performance Council (TPC). It is designed to evaluate system functionalities in a manner representative of a complex OLTP application environment, such as brokerage firms. The TPC-E schema will be used for database interaction modeling and pseudocode. The schema is based on the activities that can be observed by brokerage firms that handle transactions related to customers, markets, and finances (such as customer account, customer transaction order executions, or market-customer interactions). Table 2 is a summary of the roles within this schema, what specification each role has, and how many queries were generated for the datasets.

Table 2. Roles and specifications for the generated queries based on TPC-E [27].

Transactions	Specifications	Transactions	Specifications
Read-only Transactions		Read/Write Transactions	
Broker-volume (0)	SELECT only	Trade-order (6)	SELECT, INSERT
	714 kb		759 kb
Customer-position (1)	SELECT only	Trade-update (7)	SELECT, UPDATE
	566 kb		499 kb
Market-watch (2)	SELECT only	Data-maintenance (8)	SELECT, UPDATE
	863 kb		246 kb
Security-detail (3)	SELECT only	Market-feed (9)	SELECT, INSERT, UPDATE, DELETE
	571 kb		456 kb
Trade-status (4)	SELECT only	Trade-result (10)	SELECT, INSERT, UPDATE, DELETE
	571 kb		419 kb
Trade-lookup (5)	SELECT only	Total 11,000 queries generated	
	490 kb		

After the dataset generation, the queries are pre-processed into a set number of features. The feature extraction process follows the method proposed Ronao et al. [4], extracting a total of 277 feature values. This method was adopted by previous research by some of the authors, such as those by Bu et al. [7]., showing promising results. This query pre-processing method requires two stages:

First, a tree-wise SQL parsing process reduces the transaction queries into groups based on a defined pattern. Since SQL transactions are written in a structured language, this stage can be implemented with multiple if-then-else statements.

Second, a feature vector containing the following fields are generated: command features (SQL_CMD), projection relation features (PROJ_REL_DEC), projection attribute features (PROJ-ATTR-DEC), selection attribute features (SEL_ATTR_DEC), ORDER BY clause features (ORDBY_ATTR_DEC), GROUP BY clause features (GRPBY_ATTR_DEC), and value counter features (VALUE_CTR).

Figure 2 illustrates the query pre-processing component, and an example of a query and its extracted features is presented in Table 3.

Through this process, each SQL query can be expressed as a 277-dimensional input feature vector. This vector can then be used by the model generation process to be used for training and evaluation.

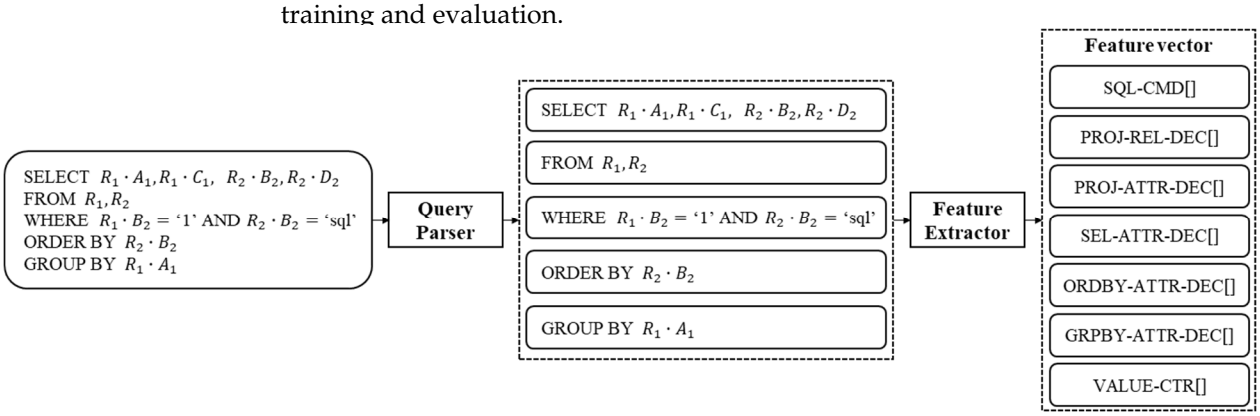

Figure 2. Query pre-processing component.

Table 3. Example query and its extracted features.

SQL Query	Feature Fields	Feature Element	Feature Values
SELECT $R_1 \cdot A_1, R_1 \cdot C_1, R_2 \cdot B_2, R_2 \cdot D_2$ FROM $R_1 \cdot R_2$ WHERE $R_1 \cdot B_2$ = '1' AND $R_2 \cdot B_2$ = 'sql' ORDER BY $R_2 \cdot B_2$ GROUP BY $R_1 \cdot A_1$	SQL-CMD[]	Query mode c	1
		Query length, Q_L	Character count
	PROJ-REL-DEC[]	Number of projected relations, P_R	2
		Position of projected relations, P_{RID}	[0011] = 3
	PROJ-ATTR-DEC[]	Number of attributes in projection clause, P_A	4
		Number of projection clause attributes per table, $P_A[]$	[2, 2]
		Position of projection clause attributes, $P_{AID}[]$	[1010, 0101] = [10, 5]
	SEL-ATTR-DEC[]	Number of attributes in selection clause, S_A	2
		Number of attributes in selection clause per table, $S_A[]$	[1, 1]
		Position of selection clause attributes, $S_{AID}[]$	[0100, 0100] = [4, 4]
	ORDER-ATTR-DEC[]	Number of attributes in ORDER BY clause, O_A	1
		Number of attributes in ORDER BY clause per table, $O_A[]$	[0, 1]
		Position of ORDER BY clause, $O_{AID}[]$	[0000, 0100] = [0, 4]
	GRPBY-ATTR-DEC[]	Number of GROUP BY clause attributes, G_A	1
		Number of GROUP BY clause attributes per table, $G_A[]$	[1, 0]
		Position of GROUP BY clause attributes, $G_{AID}[]$	[1000, 0000] = [8, 0]

Table 3. *Cont.*

SQL Query	Feature Fields	Feature Element	Feature Values
	VALUE-CTR[]	Number of string values, S_V	1
		Length of string values, S_L	3
		Number of numeric values, N_V	1
		Number of JOINs, J	0
		Number of ANDs and ORs, AO	1

3.2. Genetic Algorithm-Based Model Generation

It is preferable that the generated sets of models use different input features but still maintain high accuracy. To achieve this, a genetic algorithm-based LCS is used. Each of the 277 input features is assigned to a bit in a 277-bit long chromosome. For each chromosome, a CNN model is trained using the selected features as the input and the transaction role as the target output. The accuracy of each model is used as the fitness of their respective chromosome, which is then used for the evolutionary selection process. By repeating this process, the LCS generates a set of input feature selection rules and CNN models paired with each rule. This model generation process is illustrated in Figure 3.

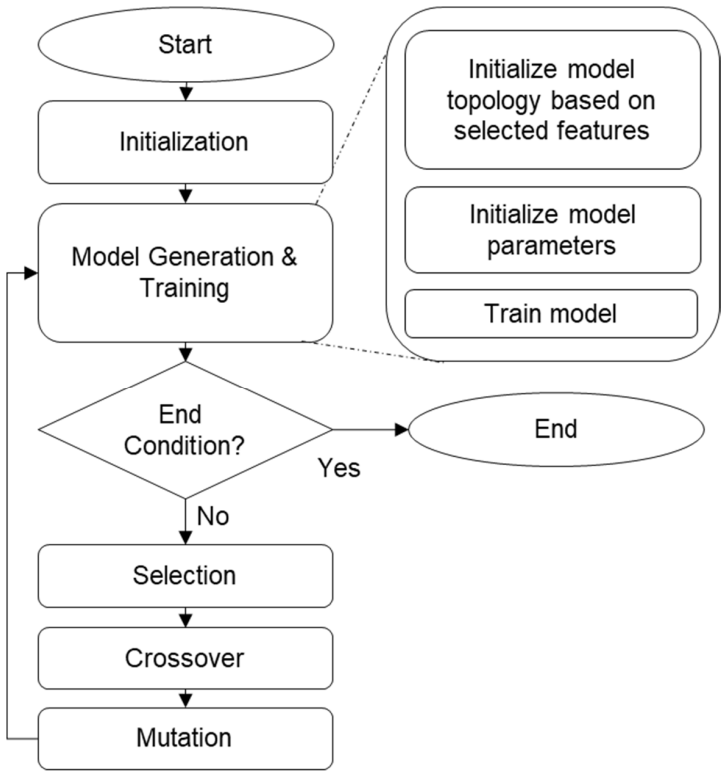

Figure 3. Flowchart of the model generation process.

As a result of using GA for feature selection, multiple models with the same architecture but different parameters are generated. These models are collected into a set of potential models that will be used for the ensemble model. In addition to generating multiple models of the same architecture, models with different CNN architectures are

trained in separate LCS processes and collected into the same set. This increases the number of potential models and improves the diversity in model types.

The architecture of each CNN used is detailed in Tables 4–7, with the signature blocks of each model illustrated in Figure 4. CNN, Inception, and ResNet-based models are based on the architecture used by Kim et al. [22] The DenseNet classifiers are based on the Dense Network image classification model [28], which connects each layer within the Dense block with multiple residual connections.

Table 4. Summary of the default hyperparameters of the CNN classifier.

Type	Configuration	Activation
Convolution	Filter 32 × 1 × 2, stride 1 × 1	tanh
Convolution	Filter 32 × 1 × 2, stride 1 × 1	tanh
Convolution	Filter 32 × 1 × 2, stride 1 × 1	tanh
Maxpool	Pooling size 1 × 2, stride 1 × 1	-
Fully-connected	256 hidden units	ReLU
Fully-connected	11 hidden units	Softmax

Table 5. Summary of the default hyperparameters of the Inception classifier.

Type	Configuration	Activation
Convolution	Filter 32 × 1 × 2, stride 1 × 1	tanh
Maxpool	Pooling size 1 × 2, stride 1 × 1	-
	Inception module	
Convolution	Filter 32 × 1 × 2, stride 1 × 1	tanh
Maxpool	Pooling size 1 × 2, stride 1 × 1	-
	Inception module	
Fully-connected	256 hidden units	ReLU
Fully-connected	11 hidden units	Softmax

Table 6. Summary of the default hyperparameters of the ResNet classifier.

Type	Configuration	Activation
Convolution	Filter 32 × 1 × 2, stride 1 × 1	tanh
	Residual block	
	Residual block	
Maxpool	Pooling size 1 × 2, stride 1 × 1	-
Fully-connected	256 hidden units	ReLU
Fully-connected	11 hidden units	Softmax

Table 7. Summary of the default hyperparameters of the DenseNet classifier.

	Configuration	Activation
Convolution	Filter 32 × 1 × 2, stride 1 × 1	tanh
	Dense block	
	Dense block	
Maxpool	Pooling size 1 × 2, stride 1 × 1	-
Fully-connected	256 hidden units	ReLU
Fully-connected	11 hidden units	Softmax

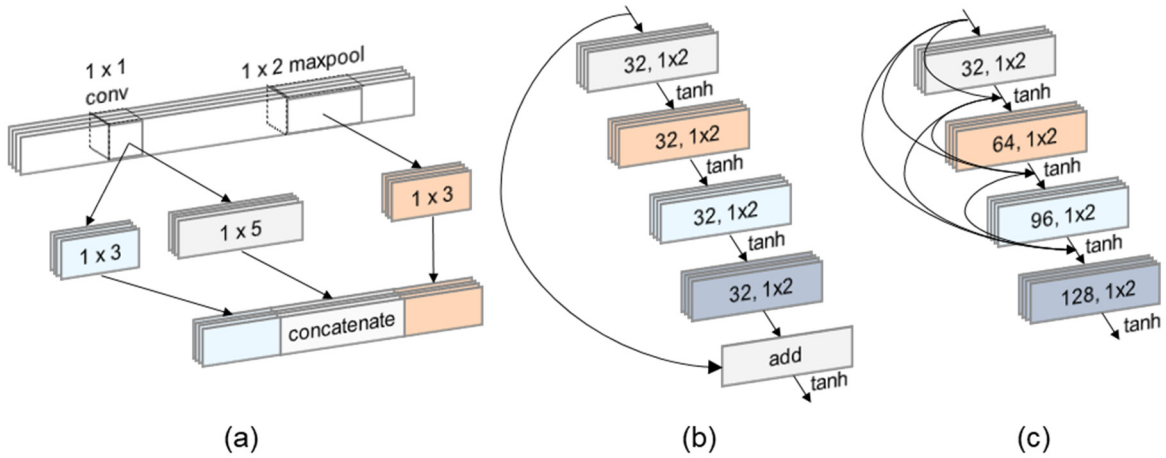

Figure 4. Signature blocks within the CNN models: (**a**) Inception block, (**b**) ResNet block, (**c**) DenseNet block.

During training, the result of each final model is recorded. This result will be used by the model ensemble component to calculate the diversity between each model. The overall structure of the model generation component is illustrated in Figure 5.

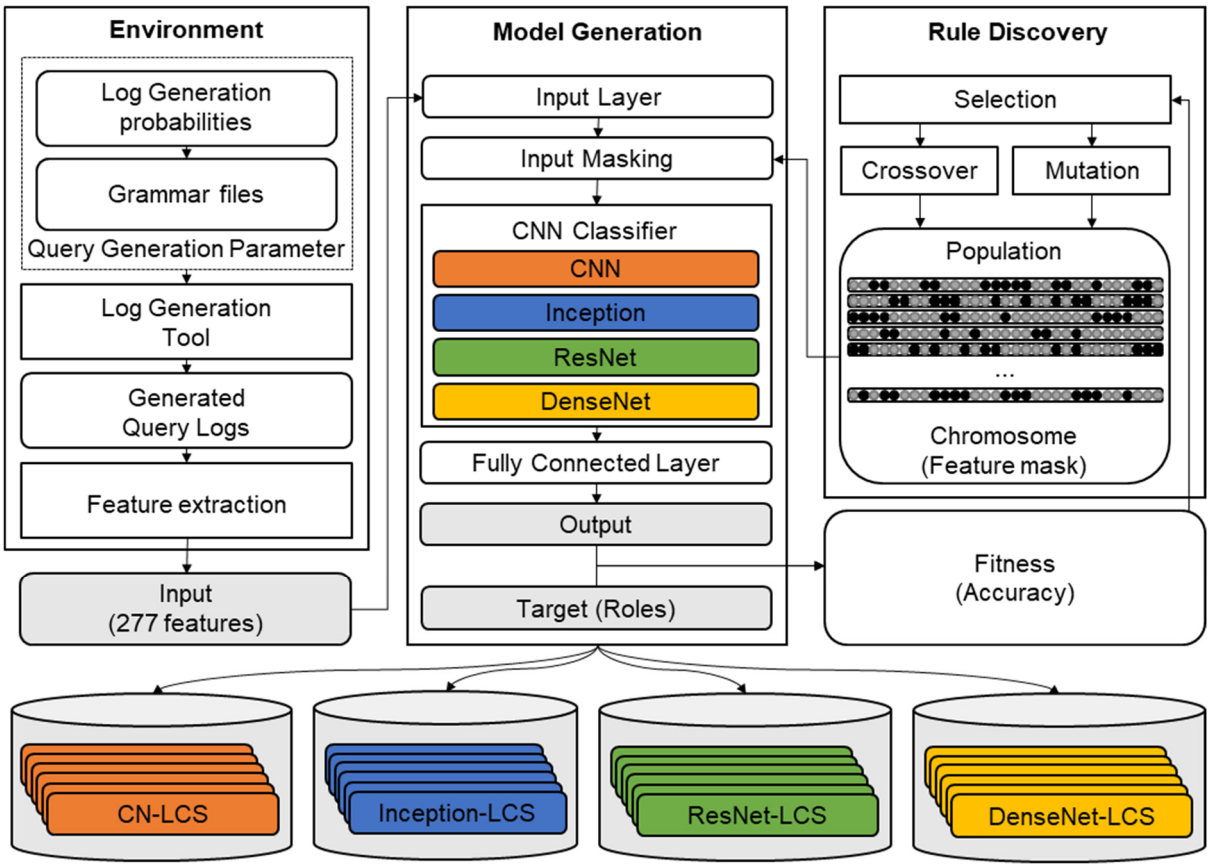

Figure 5. Diagram of the model generation component.

3.3. Model Ensemble

Simply collecting all the models into a single ensemble is not enough to utilize the characteristics of each model, as it runs the risk of the minority model being suppressed by the majority. It is preferable that each model in the ensemble is individually accurate, and at the same time, show errors at different input instances, thus displaying diversity

in results [29]. To enhance such diversity within the ensemble, the average Q-statistic (Q_{av}) [30] is used. Yule's Q-statistic [31] is calculated as denoted in Equation (1), based on the output between each pair of models (D_i, D_k) in the candidate set, where D_i denotes ith classifier and N denotes the number of observations as denoted in Table 8.

$$Q_{i,k} = \frac{N^{11}N^{00} - N^{01}N^{10}}{N^{11}N^{00} + N^{01}N^{10}} \tag{1}$$

Table 8. Relevant number of observations between two models used in Q-statistic calculation.

	D_k **Correct (1)**	D_k **Wrong (0)**
D_i correct (1)	N^{11}	N^{10}
D_i wrong (0)	N^{01}	N^{00}
Total, $N = N^{00} + N^{01} + N^{10} + N^{11}$		

The Q-statistic measures how correlated the results of the two classifiers are; when the two classifiers correctly predict the roles of the same instances, the Q value approaches 1; when the classifiers commit errors at different instances, the Q value approaches -1. Afterward, starting with the model with the highest accuracy, a set of models are collected such that Q_{av} of the ensemble is reduced until the number of models within the ensemble is more or equal to the minimum number of required models, and no other models can be added to the set without increasing the ensemble's Q_{av}:

$$Q_{av} = \frac{2}{L(L-1)} \sum_{i=1}^{L-1} \sum_{k=i+1}^{L} Q_{i,k} \tag{2}$$

The details of the ensemble collection process for selecting the heterogeneous model considering diversity are described in Algorithm 1. When using the ensemble, the results of each model are aggregated using weighted hard voting, with the accuracy of each model used as the weight. Weighted hard voting can be calculated as denoted in Equation (3), where X is the input to the ensemble, y is the resulting role classified by the ensemble, and $p_i(y|X)$ is the probability distribution of each model. Figure 6 illustrates the overall component structure.

$$y = \text{argmax}(\sum_{i=1}^{L} w_i \cdot onehot(\text{argmax}(p_i(y|X)))) \tag{3}$$

Algorithm 1. Ensemble Collection

$S \leftarrow \{\}$, $P \leftarrow$ Collection of generated models;
$x \leftarrow$ best accuracy model in P;
// Remove model with best accuracy from P and add to S
$S \leftarrow S \cup \{x\}, P \leftarrow P - \{x\}$;
while $||S||$ < target ensemble size **do**
Calculate Q_{av} for each model in P;
$x \leftarrow$ model with lowers Q_{av};
$S \leftarrow S \cup \{x\}$, $P \leftarrow P - \{x\}$;
end
Result: S

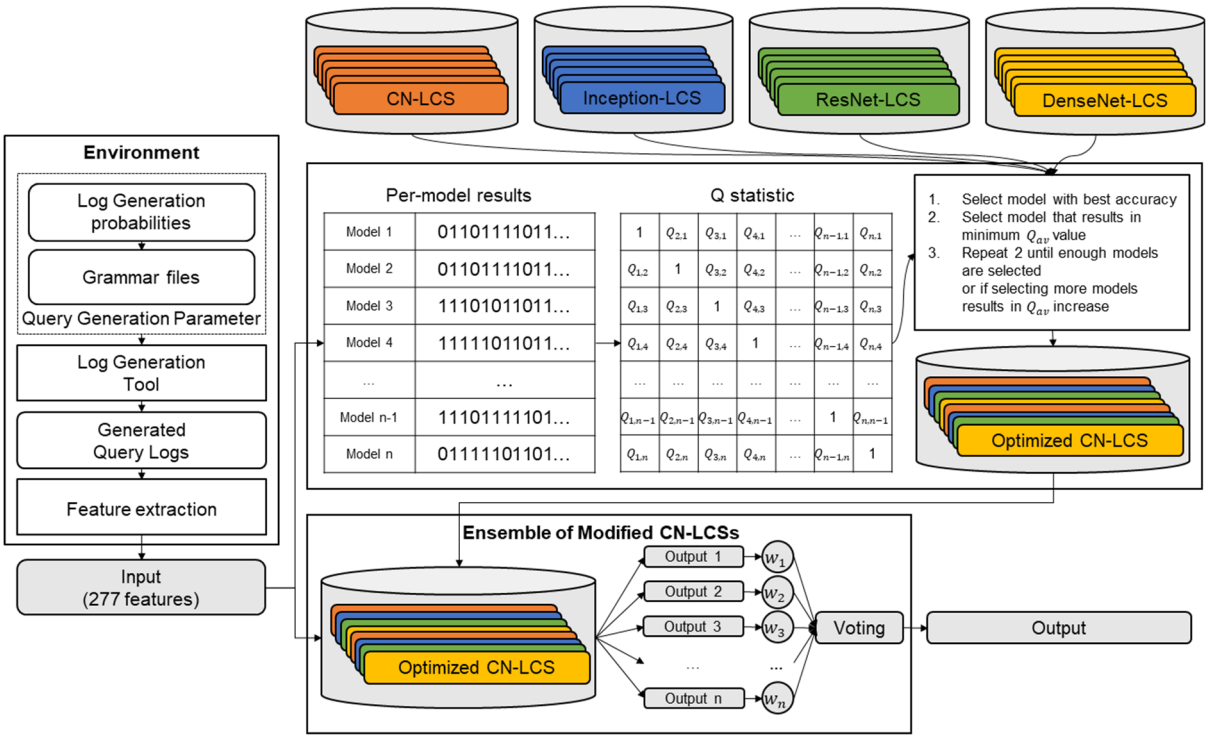

Figure 6. Diagram of the model ensemble component.

4. Experimental Results

In this section, we present how the ensemble models the SQL transactions and evaluate the performance with 10-fold cross-validation in terms of classification accuracy, which is followed by quantitative comparison with the relevant deep learning models, including previous CN-LCS models.

4.1. Implementation

Deep learning models were implemented using Pytorch [32], with chromosomes stored using Numpy [33]. Models that are not neural network-based were implemented using the default methods provided by the Scikit-learn package [34]. Hyperparameters for each model's training are in Table 9.

Table 9. Model Training Hyperparameters.

Genetic Algorithm Hyperparameter	
Number of Generations	50
Number of Population	10
Number of Elites	2
Crossover Rate	0.5
Mutation Rate	0.01
Selection Method	Roulette wheel
Crossover Method	One-point crossover
CNN Training Hyperparameter	
Optimizer	Adam
Learning Rate	0.001
Number of Epochs	500 with early stopping

During dataset generation, 1000 queries per role (total of 11,000 queries) are generated and pre-processed. The dataset is split 10-fold, using 90% of the dataset for training and the other 10% for testing for each run.

4.2. SQL Transaction Modeling Performance

Figure 7 compares the performance of existing machine learning-based query classification models, including our proposed method (Ensemble-LCS). The proposed method achieves the highest classification accuracy of 0.975, which is higher than other CN-LCS models that use a single architecture for classification.

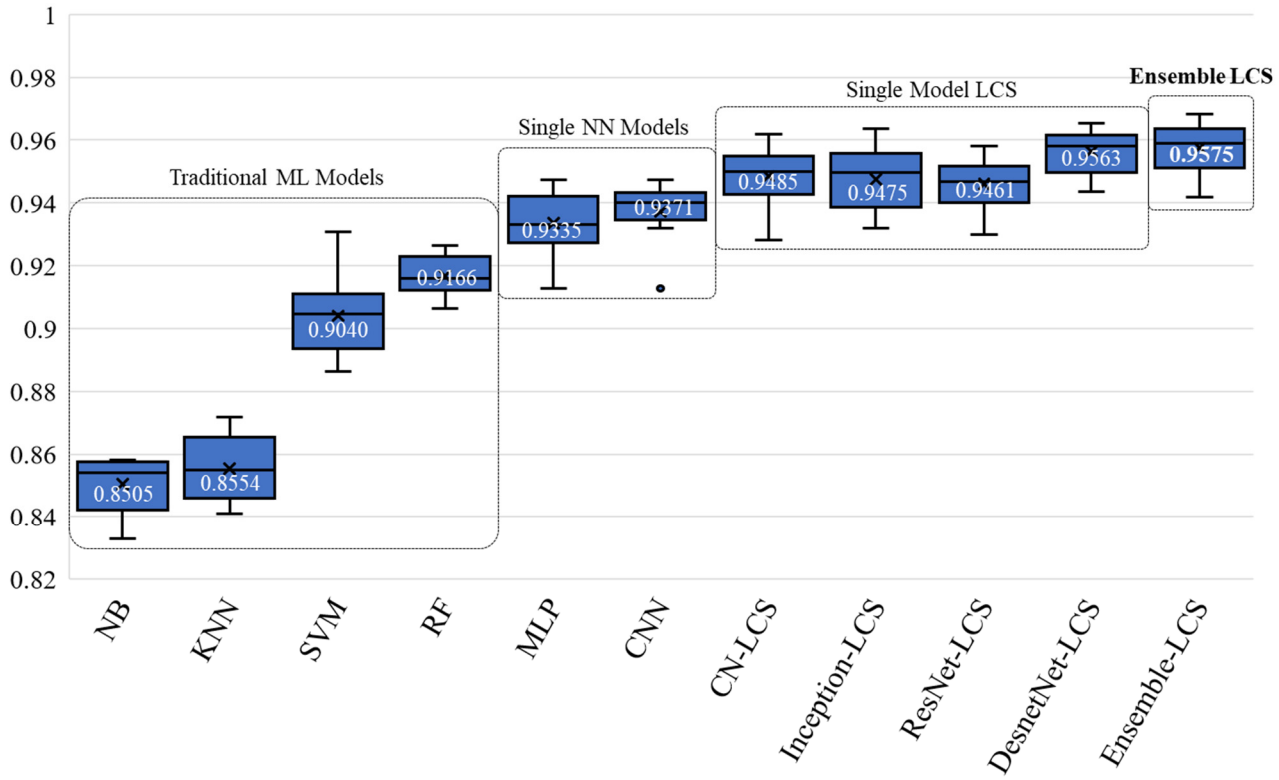

Figure 7. Ten-fold cross-validation with other machine-learning-based classifiers, including the existing combination of GA and deep learning.

Tables 10–14 are the confusion matrix of each LCS model. Darker color indicates how many samples corresponds to the (True, Predicted) pair. While all models show relatively high accuracy, single architecture LCS models showed error instances roles 2–10, with a tendency to classify Trade-status (4) roles as Trade-update (7), Trade-order (6) as Trade-result (10), Trade-update (7) as Trade-lookup (5), and Trade-result (10) as Trade-order (6). The ensemble-LCS has reduced overall misclassification. It still misclassifies in cases where all single architecture LCS misclassified, but the reduction in error for roles that single-model LCS struggled, such as role 10, the ensemble-LCS was able to compensate, thus improving overall accuracy.

In several instances, the query was correctly classified by only one of the CN-LCS architectures. Each CN-LCS architecture has an affinity to a certain TPC-E role category and thus had a better chance of correctly classifying queries that were related to that category. The ensemble-LCS was capable of correctly classifying instances, as shown in Table 15. This shows that the ensemble can utilize the varying feature classification characteristic of each of the architectures to reach a more accurate result.

Table 10. CN-LCS Confusion Matrix.

True \ Predicted	0	1	2	3	4	5	6	7	8	9	10
0	1000	0	0	0	0	0	0	0	0	0	0
1	0	1000	0	0	0	0	0	0	0	0	0
2	0	0	981	0	0	0	5	7	7	0	0
3	0	0	6	990	2	0	0	1	1	0	0
4	0	1	0	1	937	17	0	43	0	0	1
5	0	5	0	0	1	985	0	0	0	9	0
6	0	0	0	0	2	0	840	6	0	0	152
7	0	0	0	0	61	126	0	809	0	4	0
8	0	0	0	0	0	0	0	11	989	0	0
9	0	1	0	0	7	25	1	9	0	957	0
10	0	0	0	0	7	0	28	6	2	11	946

Table 11. Inception-LCS Confusion Matrix.

True \ Predicted	0	1	2	3	4	5	6	7	8	9	10
0	1000	0	0	0	0	0	0	0	0	0	0
1	0	998	2	0	0	0	0	0	0	0	0
2	0	0	997	0	0	0	0	2	1	0	0
3	0	0	8	989	0	0	0	2	1	0	0
4	0	1	11	0	929	14	0	45	0	0	0
5	0	5	0	0	3	991	0	1	0	0	0
6	0	1	10	0	4	1	907	2	0	0	75
7	0	0	15	0	19	106	0	858	0	2	0
8	0	0	8	0	1	0	0	11	972	0	8
9	0	0	0	0	0	34	0	18	0	948	0
10	0	0	4	0	7	5	124	15	1	11	833

Table 12. ResNet-LCS Confusion Matrix.

True \ Predicted	0	1	2	3	4	5	6	7	8	9	10
0	1000	0	0	0	0	0	0	0	0	0	0
1	0	1000	0	0	0	0	0	0	0	0	0
2	0	0	980	1	0	1	0	12	1	5	0
3	0	0	5	993	1	0	0	0	1	0	0
4	0	1	0	1	923	14	0	60	0	1	0
5	0	5	0	0	2	960	0	16	0	17	0
6	0	0	2	0	1	0	913	0	0	1	83
7	0	0	2	0	16	92	0	879	0	11	0
8	0	0	3	0	0	0	2	11	984	0	0
9	0	0	1	0	0	15	0	7	1	974	2
10	0	0	0	0	3	3	131	6	0	56	801

Table 13. DenseNet-LCS Confusion Matrix.

True \ Predicted	0	1	2	3	4	5	6	7	8	9	10
0	1000	0	0	0	0	0	0	0	0	0	0
1	0	1000	0	0	0	0	0	0	0	0	0
2	0	0	992	1	0	0	0	7	0	0	0
3	0	0	1	995	2	0	0	1	1	0	0
4	0	1	0	0	893	7	0	95	0	4	0
5	0	5	0	0	0	951	0	20	0	24	0
6	0	0	0	0	2	0	890	2	0	0	106
7	0	0	0	0	1	83	0	902	0	14	0
8	0	0	0	0	0	0	0	11	989	0	0
9	0	1	0	0	0	3	0	1	0	995	0
10	0	0	0	0	4	0	52	9	0	23	912

Table 14. Ensemble-LCS Confusion Matrix.

True \ Predicted	0	1	2	3	4	5	6	7	8	9	10
0	1000	0	0	0	0	0	0	0	0	0	0
1	0	1000	0	0	0	0	0	0	0	0	0
2	0	0	992	1	0	0	0	5	2	0	0
3	0	0	3	996	0	0	0	0	1	0	0
4	0	1	0	2	912	10	0	72	0	3	0
5	0	5	0	0	0	962	0	13	0	20	0
6	0	0	0	0	1	0	880	1	0	0	118
7	0	0	1	0	7	94	0	883	0	15	0
8	0	0	0	0	0	0	0	11	989	0	0
9	0	1	0	0	0	8	0	0	0	991	0
10	0	0	0	0	3	0	46	2	0	22	927

A one-way chi-square test with 10 degrees of freedom (number of roles = 1) is performed between the ensemble-LCS and single-model LCS to determine whether the ensemble-LCS shows a significant difference in distribution. The results are shown in Table 16. The ensemble-LCS shows a significant difference, meaning the increase in performance compared to single-architecture models is significant. Thus, we can conclude that the ensemble-LCS shows a significant improvement over previous LCS models.

Table 15. Query samples correctly classified by Model-LCS trained on incomplete query inputs.

Prior Correctly Classified Model	Relevant Features of Query	Associated TPC-E Role Category	Correctly Classified by Ensemble
CNN	projectionNum.holding_summary; projectionId.holding_summary; projectionNum.trade; projectionId.trade; whereClauseNum; andOrNum; stringValuesLength; stringValueNum; numericValueNum	Customer	Yes

Table 15. *Cont.*

Prior Correctly Classified Model	Relevant Features of Query	Associated TPC-E Role Category	Correctly Classified by Ensemble
CNN	projectionNum.cash_transaction; projectionId.cash_transaction; projectionNum.settlement; projectionId.settlement; projectionNum.trade_history; projectionId.trade_history; whereClauseNum; orderByNum; orderByNum.cash_transaction; orderById.cash_transaction; andOrNum; stringValuesLength; stringValueNum; numericValueNum	Customer	Yes
Inception	projectionNum.security; projectionId.security; projectionNum.trade; projectionId.trade; projectionNum.trade_type; projectionId.trade_type; whereClauseNum; orderByNum; andOrNum; stringValuesLength; stringValueNum; numericValueNum	Trade	Yes
Inception	projectionNum.trade; projectionId.trade; whereClauseNum; orderByNum; andOrNum; stringValuesLength; stringValueNum; numericValueNum	Trade	Yes
ResNet	projectionNum.last_trade; projectionId.last_trade; stringValuesLength; stringValueNum	Market	Yes
ResNet	projectionNum.last_trade; projectionId.last_trade; projectionNum.security; projectionId.security; projectionNum.trade_history; projectionId.trade_history; whereClauseNum; orderByNum; andOrNum; stringValuesLength; stringValueNum; numericValueNum	Market	Yes

Table 16. The chi-square values CN-LCS ensemble and single-model CN-LCS.

	Comparative	Chi-Square Statistic	*p*-Value	Significance
Ensemble-LCS	CN-LCS	51.3146	1.5278×10^{-7}	$p < 0.01$
	Inception-LCS	19.1691	0.0382	$p < 0.05$
	ResNet-LCS	16.7888	0.0792	$p < 0.1$
	DenseNet-LCS	23.8290	0.0081	$p < 0.01$

4.3. Model Adaptability Performance

To measure the adaptability of the Ensemble-LCS model, we create an imbalanced dataset. Queries with Role 10 (Trade-Result) are divided based on the *queryMode* feature. Approximately 50% of all Role 10 queries have a *queryMode* value of 1, followed by 2 to 4. The models are trained on a dataset where Role 10 inputs with *queryMode* value 1 is removed, simulating incomplete or imbalanced input data. Then, the models are evaluated with a dataset with all Role 10 inputs included.

Table 17 shows the result of the experiment. The ensemble-LCS shows higher accuracy for Role 10 queries compared to other models. While the accuracy for other roles was lower than for the CN-LCS model, it still showed relatively higher accuracy than other models. Showing that the ensemble-LCS model has better robustness against data imbalance.

Table 17. Accuracy of each Model-LCS trained on incomplete query inputs.

Model	Role 10 Accuracy	Other Role Accuracy
CN-LCS	0.5000	0.8889
Inception-LCS	0.5020	0.8465
ResNet-LCS	0.3050	0.7363
DenseNet-LCS	0.5260	0.8734
Ensemble-LCS	0.6640	0.8874

5. Concluding Remarks

In this paper, we proposed an ensemble-LCS model for database IDS using RBAC-based query classification. This method combines multiple heterogeneous model architectures into an ensemble to exploit each model's characteristics while supplementing each model's weakness. The proposed method showed improved accuracy compared to single-model-based LCS methods; we verified the statistical significance of the proposed method using 10-fold cross-validation and chi-square validation. The proposed method also displayed adaptability when the provided data was imbalanced. This shows that diversity-guided ensemble-LCS can be used to develop improved IDS using RBAC-based query role classification.

Future work will include introducing additional CNN models to be included in the ensemble to improve overall diversity and increase classification accuracy. Additionally, the proposed method will have to be tested on other datasets to validate whether the proposed method can be generalized to other database schemas.

Author Contributions: S.-J.B.; H.-B.K. and S.-B.C. have participated in preparing the research from the beginning to the end, such as establishing research design, methods, and analysis. S.-J.B.; H.-B.K. and S.-B.C. discussed and finalized the analysis results to prepare the manuscript according to the progress of research. All authors contributed equally to this work. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by an IITP grant funded by the Korean government (MSIT) (No. 2020-0-01361, Artificial Intelligence Graduate School Program (Yonsei University)) and Air Force Defense Research Sciences Program funded by Air Force Office of Scientific Research.

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

References

1. Khraisat, A.; Gondal, I.; Vamplew, P.; Kamruzzaman, J. Survey of intrusion detection systems: Techniques, datasets and challenges. *Cybersecurity* **2019**, *2*, 20. [CrossRef]
2. Sagar, R.; Jhaveri, R.; Borrego, C. Applications in Security and Evasions in Machine Learning: A Survey. *Electronics* **2020**, *9*, 97. [CrossRef]
3. Rawat, D.B. Journal of Cybersecurity and Privacy: A New Open Access Journal. *J. Cybersecur. Priv.* **2021**, *1*, 195–198. [CrossRef]
4. Ronao, C.A.; Cho, S.B. Anomalous query access detection in RBAC-administered databases with random forest and PCA. *Inf. Sci.* **2016**, *369*, 238–250. [CrossRef]
5. Bertino, E.; Sandhu, R. Database security-concepts, approaches, and challenges. *IEEE Trans. Dependable Secur. Comput.* **2005**, *2*, 2–19. [CrossRef]
6. Safa, N.S.; Maple, C.; Watson, T.; Von Solms, R. Motivation and opportunity based model to reduce information security insider threats in organisations. *J. Inf. Secur. Appl.* **2018**, *40*, 247–257. [CrossRef]
7. Bu, S.-J.; Cho, S.-B. A convolutional neural-based learning classifier system for detecting database intrusion via insider attack. *Inf. Sci.* **2020**, *512*, 123–136. [CrossRef]
8. Saxena, N.; Hayes, E.; Bertino, E.; Ojo, P.; Choo, K.-K.R.; Burnap, P. Impact and Key Challenges of Insider Threats on Organizations and Critical Businesses. *Electronics* **2020**, *9*, 1460. [CrossRef]

9. Pan, Z.; Hariri, S.; Pacheco, J. Context aware intrusion detection for building automation systems. *Comput. Secur.* **2019**, *85*, 181–201. [CrossRef]
10. Buehrer, G.; Weide, B.W.; Sivilotti, P.A. Using parse tree validation to prevent SQL injection attacks. In Proceedings of the 5th International Workshop on Software Engineering and Middleware, Lisbon, Portugal, 5–6 September 2005; pp. 106–113.
11. Ahmim, A.; Maglaras, L.; Ferrag, M.A.; Derdour, M.; Janicke, H. A novel hierarchical intrusion detection system based on decision tree and rules-based models. In Proceedings of the 2019 15th International Conference on Distributed Computing in Sensor Systems (DCOSS), Santorini Island, Greece, 29–31 May 2019; pp. 228–233.
12. Xue, B.; Zhang, M.; Browne, W.N.; Yao, X. A survey on evolutionary computation approaches to feature selection. *IEEE Trans. Evol. Comput.* **2015**, *20*, 606–626. [CrossRef]
13. Kim, M.Y.; Lee, D.H. Data-mining based SQL injection attack detection using internal query trees. *Expert Syst Appl.* **2014**, *41*, 5416–5430. [CrossRef]
14. Dam, H.H.; Abbass, H.A.; Lokan, C.; Yao, X. Neural-based learning classifier systems. *IEEE T Knowl. Data En.* **2008**, *20*, 26–39. [CrossRef]
15. Kim, T.-Y.; Cho, S.-B. CNN-LSTM Neural Networks for Anomalous Database Intrusion Detection in RBAC-Administered Model. In *International Conference on Neural Information Processing*; Springer: Cham, Germany, 2019; pp. 131–139. [CrossRef]
16. Khare, N.; Devan, P.; Chowdhary, C.L.; Bhattacharya, S.; Singh, G.; Singh, S.; Yoon, B. SMO-DNN: Spider Monkey Optimization and Deep Neural Network Hybrid Classifier Model for Intrusion Detection. *Electronics* **2020**, *9*, 692. [CrossRef]
17. Selvakumar, B.; Muneeswaran, K. Firefly algorithm based feature selection for network intrusion detection. *Comput. Secur.* **2019**, *81*, 148–155. [CrossRef]
18. Choi, S.-G.; Cho, S.-B. Evolutionary Reinforcement Learning for Adaptively Detecting Database Intrusions. *Log. J. Igpl.* **2020**, *28*, 449–460. [CrossRef]
19. Bu, S.-J.; Cho, S.-B. A Hybrid System of Deep Learning and Learning Classifier System for Database Intrusion Detection. In *Hybrid Artificial Intelligent Systems*; Springer: Cham, Switzerland, 2017; pp. 615–625.
20. Kim, T.-Y.; Cho, S.-B. Particle Swarm Optimization-Based CNN-LSTM Networks for Anomalous Query Access Control in RBAC-Administered Model. In *Hybrid Artificial Intelligent Systems*; Springer: Cham, Switzerland, 2019; pp. 123–132.
21. Bu, S.-J.; Cho, S.-B. Genetic Algorithm-Based Deep Learning Ensemble for Detecting Database Intrusion via Insider Attack. In *Hybrid Artificial Intelligent Systems*; Springer: Cham, Switzerland, 2019; pp. 145–156.
22. Kim, J.Y.; Cho, S.B. Exploiting deep convolutional neural networks for a neural-based learning classifier system. *Neurocomputing* **2019**, *354*, 61–70. [CrossRef]
23. Szegedy, C.; Liu, W.; Jia, Y.; Sermanet, P.; Reed, S.; Anguelov, D.; Erhan, D.; Vanhoucke, V.; Rabinovich, A. Going deeper with convolutions. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Boston, MA, USA, 8–10 June 2015; pp. 1–9.
24. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 27–30 June 2016; pp. 770–778.
25. Khraisat, A.; Gondal, I.; Vamplew, P.; Kamruzzaman, J.; Alazab, A. Hybrid intrusion detection system based on the stacking ensemble of c5 decision tree classifier and one class support vector machine. *Electronics* **2020**, *9*, 173. [CrossRef]
26. Khraisat, A.; Gondal, I.; Vamplew, P.; Kamruzzaman, J.; Alazab, A. A Novel Ensemble of Hybrid Intrusion Detection System for Detecting Internet of Things Attacks. *Electronics* **2019**, *8*, 1210. [CrossRef]
27. Chen, S.; Ailamaki, A.; Athanassoulis, M.; Gibbons, P.B.; Johnson, R.; Pandis, I.; Stoica, R. TPC-E vs. TPC-C: Characterizing the new TPC-E benchmark via an I/O comparison study. *ACM Sigmod. Record* **2011**, *39*, 5–10. [CrossRef]
28. Huang, G.; Liu, Z.; Van Der Maaten, L.; Weinberger, K.Q. Densely connected convolutional networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, HI, USA, 21–24 July 2017; pp. 4700–4708.
29. Díez-Pastor, J.F.; Rodríguez, J.J.; García-Osorio, C.I.; Kuncheva, L.I. Diversity techniques improve the performance of the best imbalance learning ensembles. *Inf. Sci.* **2015**, *325*, 98–117. [CrossRef]
30. Kuncheva, L.I.; Whitaker, C.J. Measures of Diversity in Classifier Ensembles and Their Relationship with the Ensemble Accuracy. *Mach. Learn.* **2003**, *51*, 181–207. [CrossRef]
31. Yule, G.U. VII. On the association of attributes in statistics: With illustrations from the material of the childhood society, &c. *Philos. Trans. R. Soc. London. Ser. A Contain. Pap. A Math. Or Phys. Character* **1900**, *194*, 257–319.
32. Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L. Pytorch: An imperative style, high-performance deep learning library. *Adv. Neural Inf. Processing Syst.* **2019**, *32*, 8026–8037.
33. Harris, C.R.; Millman, K.J.; van der Walt, S.J.; Gommers, R.; Virtanen, P.; Cournapeau, D.; Wieser, E.; Taylor, J.; Berg, S.; Smith, N.J. Array programming with NumPy. *Nature* **2020**, *585*, 357–362. [CrossRef] [PubMed]
34. Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V. Scikit-learn: Machine learning in Python. *J. Mach. Learn. Res.* **2011**, *12*, 2825–2830.