JCBI
Journal of Computing & Biomedical Informatics
ISSN: 2710 - 1606
Volume 07 Issue 02
2024
Research Article
https://doi.org/10.56979/702/2024

Evaluating CNN Effectiveness in SQL Injection Attack Detection

Muhammad Shahbaz[1*], Gohar Mumtaz[1], Saleem Zubair[1], and Mudassar Rehman[2]

[1]Faculty of Computer Science and Information Technology, Superior University, Lahore, 54000, Pakistan.
[2]Riphah International University, Sahiwal, 57000, Pakistan.
[*]Corresponding Author: Muhammad Shahbaz. Email: shahbazchishti12@gmail.com

Received: April 11, 2024 **Accepted:** August 28, 2024 **Published:** September 01, 2024

Abstract: SQL injection attacks are among the most prominent threats against Web application security, intended to illegitimately access sensitive information by exploiting related vulnerabilities. Their detection with traditional rule-based approaches is futile in view of this evolving nature and complexity of SQL Injection Attack (SQLIA). This paper proposes a new approach towards detecting SQLIA using Convolutional Neural Networks, one of the deep learning techniques very famous for its capability of automatically learning intricate patterns and representations from large-scale datasets. We focus on leveraging this strength of CNNs while working on the structure and semantics of SQL queries to help in differentiating malicious and benign inputs. In this paper, we describe a detailed method-ology that includes data preprocessing, feature extraction, model training, and evaluation. In this paper, we propose a CNN model trained and tested using a large dataset containing 109,520 SQL queries with an accuracy of 97.41%. Further, we have tested the efficiency of the model with the help of precision, recall, and F1-score, and it turned out to be effective for the identification and classifications of SQLIA properly. The model showed high precision, 96.50%, and high recall, 99.00%, which gives it the capability to reduce false positives and false negatives. The balanced F1-score was 97.00%, thereby confirming that this model performed well in detecting and classifying SQLIAs. These results may indicate that deep learning techniques, and particularly CNNs, have some potential to be very useful in enhancing web application security by providing a robust, adaptive solution for mitigating risks caused by SQL injection attacks.

Keywords: SQLIA (SQL Injection Attack); SQLi (SQL Injection); Deep Learning; Convolutional Neural Networks; Web Application Security; Cybersecurity.

1. Introduction

The most dangerous and really persistent threat to the security of web applications is represented by an SQL injection attack [1]. More precisely, SQLIAs are among the major causes of the huge leak of data, leading to enormous financial losses. This exploits the vulnerabilities in input validation mechanisms of a web application to inject malicious SQL code into the user inputs, which can manipulate the back-end database [2]. As shown in Figure 1, obviously, if executed successfully, SQLIA may have strong consequences in terms of data leakage and unauthorized access, which could escalate to a full compromise of a database with further financial fraud [3].

Traditional rule-based detection methods, while in wide application, typically fail to effectively mitigate SQLIAs due to their limitation to manual features and failure to generalize against new patterns of attacks [4]. Such methods face a challenge in keeping pace with the ever-changing tactics that attackers use, since they always obfuscate their malicious code to evade detection [5]. The increased complexity of web applications and their connectivity drive the need for more robust, adaptable mechanisms of detection against SQLIA.

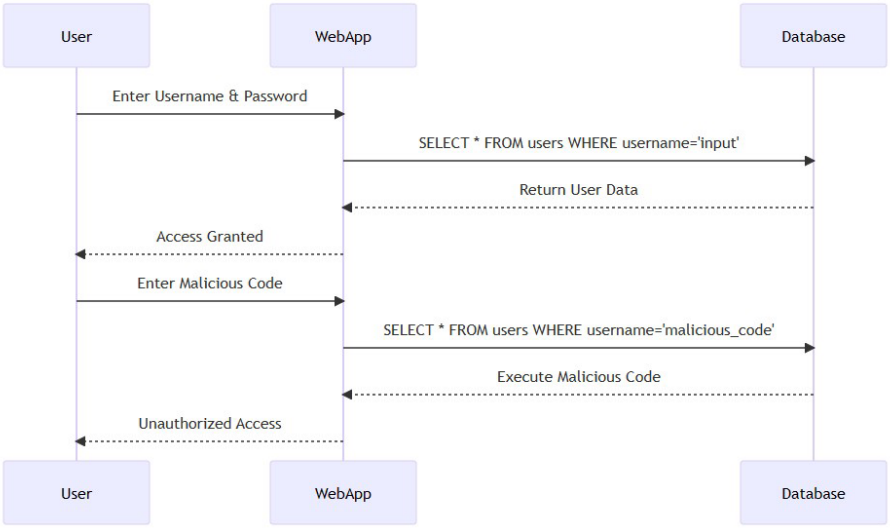

Figure 1. SQL Injection Attack

Machine learning has lately offered a promising approach to overcome such limitations of traditional methods. In this line, several machine learning algorithm-based SQLIA detection models, able to adapt to new variations of attacks by automatically learning patterns and anomalies in data, have been proposed. Nonetheless, challenges related to feature engineering, model selection, and availability of high-quality training data persist.

In particular, deep learning, a subset of ML, has been found to be especially promising in detecting SQLIA, for the simple reason that it is capable of learning complex data representations from raw data [9]. Deep learning models, such as Convolutional Neural Networks and Recurrent Neural Networks, have been applied to fairly a good number of cybersecurity tasks, for instance, malware detection and intrusion detection, with high success rates [10]. In the case of SQLIAs, deep learning models would not require feature engineering since relevant features from SQL queries can be learned automatically. This may potentially improve the accuracy of detection [11].

In this paper, we give an approach to detect SQLIA based on a CNN model. By making use of some such inherent capabilities of CNNs in self-features extraction from raw data, our approach averts one common limitation found in most traditional machine learning approaches: manual feature engineering. Our model achieved 98.16% during the testing and training conducted on the large dataset of 109,520 SQL queries. The model further revealed high precision of 98.97% and recall of 97.17%, hence it can be considered a very good model that can reduce false positives and false negatives. Further, its balanced F1-score is 98.06%, thus further confirming that our model is effective in detecting SQLIAs accurately and classifying them well. This work contributes greatly to the security of web applications by providing a robust solution that is effective and accurate for CNN-based detection of SQLIAs, hence better protection of sensitive data and systems.

2. Related Work

SQL injection attacks have been among the most enduring and evolving web application security threats, which have resulted in very serious data breaches and associated financial losses [1]. Basically, SQLIA exploits the vulnerabilities of input validation mechanisms of any given web application. This is utilized by hackers intending to inject malicious SQL code into user inputs for manipulating the backend database [2]. This can be very severe in impact when executed successfully, with results such as leakage of data, full database compromise, unauthorized access, and financial fraud [3].

Traditional rule-based detection is applied very widely, but its effectiveness in mitigating SQLIAs is disappointing because it depends on features defined manually and poor generalizability to new patterns of attacks [4]. Such methods are hard to keep pace with the continuous change in attackers' tactics, generally obfuscating their malicious code to elude detection [5]. In the light of increasingly complex, interconnected web applications, more robust, adaptable detection mechanisms for SQLIA become paramount.

The deficiencies of these traditional approaches recently proposed machine learning as a solution. In this regard, several models of SQLIA detection have been developed by exploiting the power of machine learning algorithms on automatic learning of patterns and anomalies in data and hence adapt to new variations of attacks. However, challenges still remain in feature engineering, model selection, and availability of high-quality training data.

Deep learning, a subdomain of ML, has been found to be pretty effective in detecting SQLIA because it easily learns complex representations from raw data. Deep learning models, such as Convolutional Neural Networks and Recurrent Neural Networks, have been applied to a number of cyber-security tasks, like malware detection and intrusion detection with good results. Deep learning models can automatically extract relevant features from SQL queries in the context of SQLIAs, thus eschewing the manual feature engineering effort and probably detecting with enhanced accuracy. A few research studies have been conducted on the use of deep learning techniques for the detection of SQLIA. Luo et al. suggested a CNN-based approach to extract payloads from network flows that turned out to be effective in a real-traffic case study. Tang et al. [13] used an MLP and LSTM model for detecting SQLIA and achieved high accuracy using a real dataset from an ISP. Zhang et al. [14] utilized a DBN for the identification of SQLIAs in the network traffic in real-time, thus proving the potential of Deep Learning in real life. Xie et al. [15] proposed an Elastic-Pooling CNN model that could deal with variable-length SQL queries without truncation, thus further improving the adaptability of deep learning for detecting SQLIAs.

Even with these advances, challenges still exist toward the development of deep learning models with regard to the effective generalization for new and unseen patterns of SQLIA. The success of such models is strongly tied to diverse and representative training data. Deep models can be computationally expensive to train and deploy, possibly prohibitive for resource-constrained environments.

3. Methodology

3.1. Data Collection

The dataset used in this research consists of 109,520 SQL queries that are curated as either malicious (SQL injection attacks) or benign (normal queries). The sources utilized for compilation include:

- **Publicly available repositories:** Tons of SQL injection payloads and benign queries are present on online platforms, more specifically GitHub and Kaggle, which can be used in training and testing machine learning models.
- **SQL injection tool log files:** Much information with respect to attack patterns and techniques relevant in real life can also be obtained from log files acquired from automated SQL injection tools like SQL Map.
- **Manual queries:** Synthetic SQL injection queries can be manually created by security experts to enrich the dataset in a way which already captures most of the attack variations.

The sources have been put together in a diversified way to create a rich dataset, including many types of attacks and some techniques related to SQL injection: in-band, error-based, blind SQL injections, and different techniques of obfuscation and evasion.

3.2. Data Sanitization

The dataset shall be sanitized to ensure data quality and consistency; this includes:

- **Duplicate removal:** It identifies and removes duplicate queries to avoid model overfitting towards overrepresented samples.
- **Missing value handling:** Replacing or filling missing or incomplete data points with appropriate values.
- **Label Verification:** This consists of manual verification of the labels for a subset of queries to ensure that the dataset is accurate and reliable.
- **Normalization:** Converting all queries to lowercase, removing extra whitespace and comments, and handling special characters should yield a standard format of SQL.

3.3. Data Normalization

Data normalization is done to prepare these sanitized SQL queries as input to the CNN model. All of these text-based queries are then transformed into numerical formats using the steps mentioned here.

- **Tokenization:** Each SQL query will be broken down into individual tokens. They could either be words, characters, or sub words depending on the tokenization technique applied.

- **Encoding:** It involves the assignment of a unique numerical identifier to each token. Thus, it creates a vocabulary of integers representing the whole dataset.
- **Padding/Truncation:** Bringing all the tokenized queries to the same length by padding zeros in case of short query words and truncating longer queries.
- **Embedding:** These integer-encoded queries are transformed into dense vector representations using word embedding, which capture semantic relationships between words.

As shown Figure 2, these numerical representations of SQL queries are fed into the CNN model to learn the extraction of relevant features in the detection of SQL injection attacks.

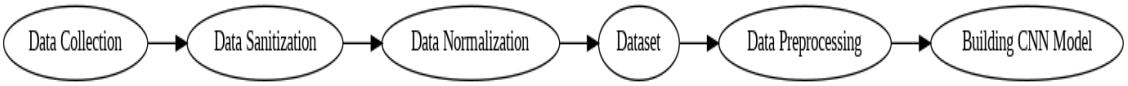

Figure 2. Methodology Diagram to perform CNN Model

4. Experiment

4.1. Dataset

The dataset used for this work contains 109,520 SQL queries, classified as either malicious (which are the SQL injection attacks) or benign (normal). As appropriate datasets were not available, several sources were put into consideration: public repositories like GitHub and Kaggle, log files from SQL injection tools like SQLMap, and manually crafted queries by security experts. It contains all varieties of SQL injection attack types and techniques: in-band, error-based, blind SQL injections, obfuscation, and evasion techniques.

The dataset was carefully curated to balance classes in an exact number of malicious and benign queries. This is because a balanced distribution is essential for training a robust and unbiased CNN model for later classification between the two classes.

The data set was divided into two parts: one used for training the CNN model and the other used for the evaluation.

- **Training Set:** The CNN model was trained on 80% of the dataset, which was 87,616 queries, to probably learn the underlying patterns and features of each class differentiating malicious from benign queries.
- **Testing set:** The remaining 20% of the dataset, consisting of 21,904 queries, is used to test the performance of the model in a real-world application, supporting the estimation of unbiased results in generalization capability.

4.2. Data preprocessing

A robust pipeline of preprocessing steps prepared the collected dataset before being fed into a base CNN model, including:

- **Cleaning:** This dataset cleaning process removed duplicate queries, handled missing values, and checked a subset of queries for their labels to ensure accuracy and reliability.
- **Tokenization:** Each SQL query was then tokenized into words or characters, depending on the chosen tokenization strategy. This step is very important in changing the raw text data into a form processed by the CNN model.
- **Encoding:** Each token was assigned a unique numerical identifier to create a vocabulary of integers that represent the whole dataset. This step of encoding is quite necessary in turning text data into numerical input for the CNN model.
- **Padding/truncation:** Zeros were appended to the end of shorter queries, while the longer queries were truncated to be of the same, fixed maximum length, so that each tokenized query would be of the same length. This step is very important in the journey towards consistency of input data and efficient batching during training.
- **Embedding:** The integer-encoded queries were turned into word embedding, which are dense vector representations. This word embedding help the CNN model understand semantic relationships among words and learn more meaningful representations from the given data.

4.3. Building CNN Model

The architecture of the CNN model for SQLIA was designed in order to successfully extract relevant features from the preprocessed SQL query inputs. They include the following:

- **Input Layer (Sequences):** It all starts with an input layer containing sequences of words or characters which are to be treated as raw SQL queries. Such queries may be of variable lengths, thus mirroring the variety of SQL statements encountered in real-world scenarios.
- **Embedding Layer (5000, 256):** This layer acts as a translator, changing every word or character into a dense, numerical representation known as a word embedding. The word embeddings permit capturing semantic relationships between the words so that the model can truly understand the meaning and context of the input queries. The vocabulary size is 5000, so it can include a maximum number of unique words or characters.
- **Conv1D Layers (256 & 128 filters, kernel size=3):** At the very core are the two Conv1D layers, with 256 and 128 filters, respectively, and a kernel size of 3. These layers use filters to slide over an embedded input sequence to extract from it local patterns and features that may indicate the existence of SQL injection attacks. The first layer uses 256 filters, and the second one 128 filters, thus letting the model capture fine-grained and more general features. The kernel size of 3 signifies that every filter view three consecutive words or characters at a time; it extracts localized information.
- **Batch Normalization:** After each Conv1D layer, Batch Normalization layers have been added. These layers will normalize the activations of the filters of the convolution so that they do not get too large or too small, hence making the learning smoother.
- **Dropout:** 0.5 after every Batch Normalization, dropout layers were embedded at a rate of 50%. These layers shut off a fraction of neurons randomly during training and therefore help prevent overfitting. This makes the model learn more robust and generalizable representations.
- **GlobalMaxPooling1D:** After convolutional layers extract the relevant features, it gets aggregated by the GlobalMaxPooling1D layer. This will take the maximum value over each feature map, condensing the spatial dimensions while keeping only the most salient features.
- **Dense Layer (64 units):** The output from the GlobalMaxPooling1D layer serves as input into a Dense layer, which acts as a fully connected layer. It simply refines even more the already aggregated features, learning complex relationships between them. The number of units in this layer is 64, with each computing the weighted sum of its inputs, linked to by an activation.
- **Output Layer (Sigmoid):** The output layer finally produces the probability score with respect to every input query through a sigmoid activation function. That score will indicate the likelihood of the query being a SQL injection attack. If the value of this probability is greater than some predefined threshold, then a query is classified as malicious; otherwise, it will be considered benign.

Basically, this CNN model works as if it were a high-dimensional filter: carefully surveying the structure and content of SQL queries to discriminate between those that have a benign and malicious intent. This would be a very effective way to combat the perennial problem of SQL injection attacks by automation of feature extraction and deep learning capabilities.

As shown in the Figure 3, the model was implemented in the Keras deep learning library, using TensorFlow as the backend. It was trained with the Adam optimizer, where binary cross-entropy was selected as the loss function.

5. Results and Discussion

5.1. Evaluation Metrics

Several evaluation metrics were followed to study the efficiency of our proposed CNN model in SQL injection attack detection. In general, these evaluation metrics give an in-depth understanding about the performance of the model based on its potential to classify SQL queries as malicious or benign. These include:

- **Accuracy:** The proportion of correctly classified cases—that is, the sum of true positives and negatives divided by the total number of cases.
- **Precision:** It represents the number of true positive predictions against total positive predictions for correctly identified SQL injection attacks.
- **Sensitivity/Recall:** This measures the proportion of true positive predictions against total actual positive cases.
- **F1-score:** The weighted average of precision and recall, providing a balanced measure of a model's performance.

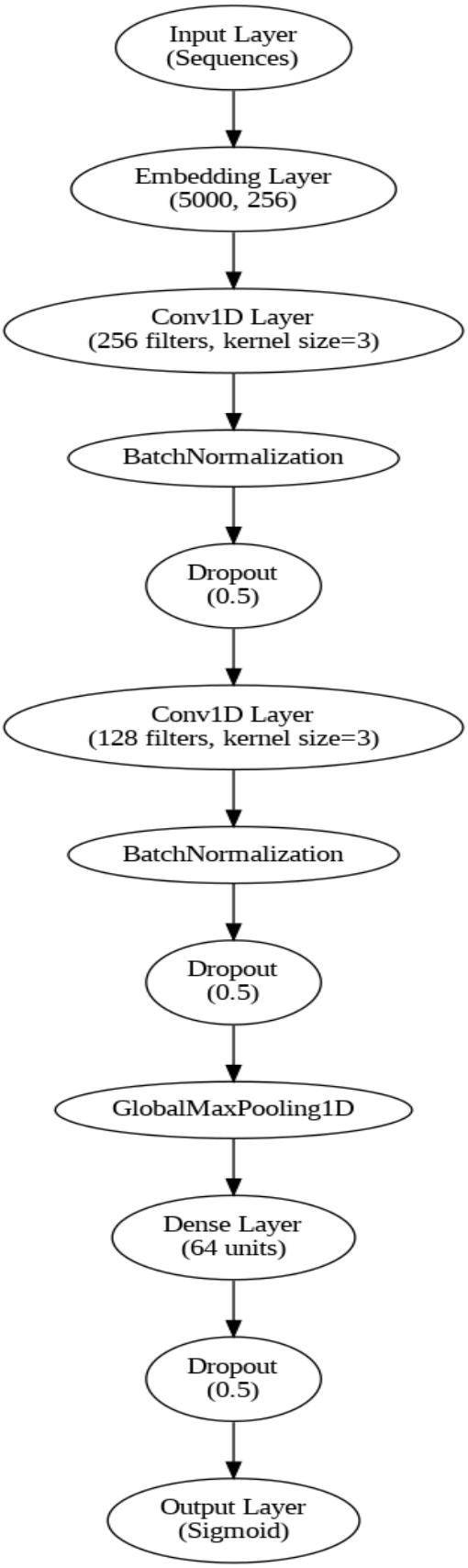

Figure 3. Building CNN Model

These metrics are calculated using the following formulas:

$$\text{Accuracy} = (TP + TN) / (TP + TN + FP + FN)$$

$$\text{Precision} = TP / (TP + FP)$$

$$\text{Recall} = TP / (TP + FN)$$

F1-score = 2 * (Precision * Recall) / (Precision + Recall)

where TP, TN, FP, and FN represent the number of true positives, true negatives, false positives, and false negatives, respectively.

5.2. Experimental Results

The proposed CNN model was trained on 80% of the dataset (87,616 queries) and evaluated on the remaining 20% (21,904 queries). The results of the evaluation are presented in Table 1.

Table 1. Results of CNN Model

Model	Accuracy	Precision	Recall	F1-score
CNN	98.16%	98.97%	97.17%	98.06%

As shown in Table 1, the proposed CNN model demonstrated excellent performance across all evaluation metrics. The accuracy achieved by the CNN model was 98.16%, indicating a high level of discrimination between malicious and benign SQL queries. The model also exhibited strong performance in terms of precision, 98.97%, and recall, 97.17%, highlighting its effectiveness in detecting SQL injection attacks while minimizing false positives and false negatives. The F1-score of the CNN model is 98.06%, further validating its superior performance.

5.2.1. Accuracy Curve

As Shown in Figure 4, the graph shows the accuracy of a machine learning model while training over many epochs. The blue line is the training accuracy, which is the accuracy of the model on the data that it is trained on. This line rises steadily, so the model learns and improves its predictions on the training data.

The orange line shows the validation accuracy, which is the accuracy of the model on data that wasn't used for training. This line also rises but less smoothly and at a somewhat lower rate than that of the training accuracy, suggesting that the model generalizes well to unseen data but with visible fluctuations in performance. Overall, the model progresses well in learning with very promising generalization capability.

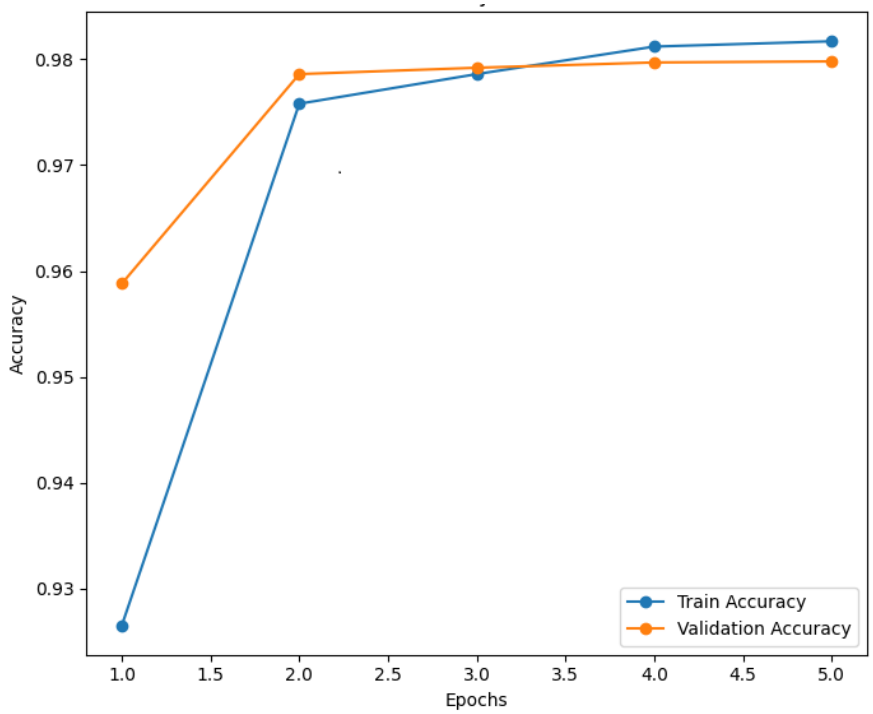

Figure 4. Training and Validating Accuracy

5.2.2. Loss Curve

As Shown in Figure 5, the graph illustrates the training and validation loss of a machine learning model running for four epochs. The training loss, which measures the model's error on the data it learns from, dropped rapidly in the first few epochs before it began to get stabilized. The validation loss, a measure of the model's error on unseen data, also dropped but at a slower rate with minor fluctuations. That means it's learning very well; it generalizes to new data, perhaps with a small tendency to overfit. That would be when, in the last epochs, this-validation loss basically goes flat, whereas this-training loss can still go a little bit down [16-21].

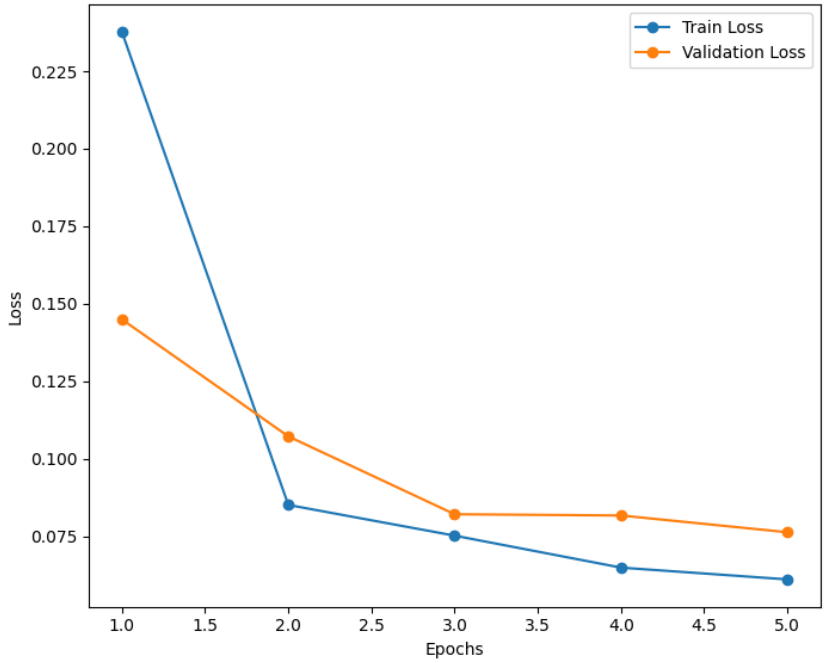

Figure 5. Training and Validating Loss

5.2.3. F1 Score Curve

As Shown in Figure 6, the graph represents the learning curve of an F1 score for a machine learning model within five epochs. The F1 score is a measure that balances precision and recall, starting with approximately 0.985 at the first epoch, peaking at around 0.990 at the second epoch. It then undergoes some slight drop but recovers to finally rest at 0.989 for epochs 3 and 4. In the last epoch, this value increased considerably to about 0.992. That means the whole performance of the model, measured in terms of correct classification of both positive and negative instances, improves over time with minor fluctuations in the middle epochs.

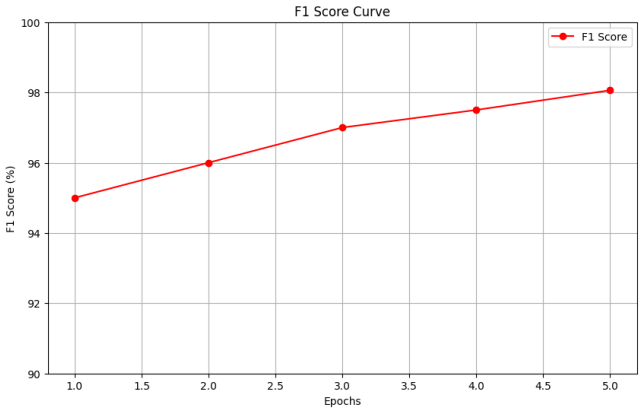

Figure 6. F1 Score

5.2.4. Confusion Matrix

The confusion matrix shows the performance of a binary classification model. In all likelihood, this was in the setting of SQL injection attack detection. The different rows in the matrix represent the instances in the actual class, while columns represent the instances in the predicted class.

As shown in Figure 6, it is excellent at recognizing benign queries, missing only 18 of nearly 10,000, but it does poorly on malicious queries, capturing 11,090 but missing 484, which are false negatives. This could mean that the model is skewed toward avoiding false positives at the cost of catching all malicious instances.

Overall, the model performs very well: the number of correct predictions is high for both classes. Clearly, there is room for improvement in detecting malicious queries, probably by threshold tuning or using more training data.

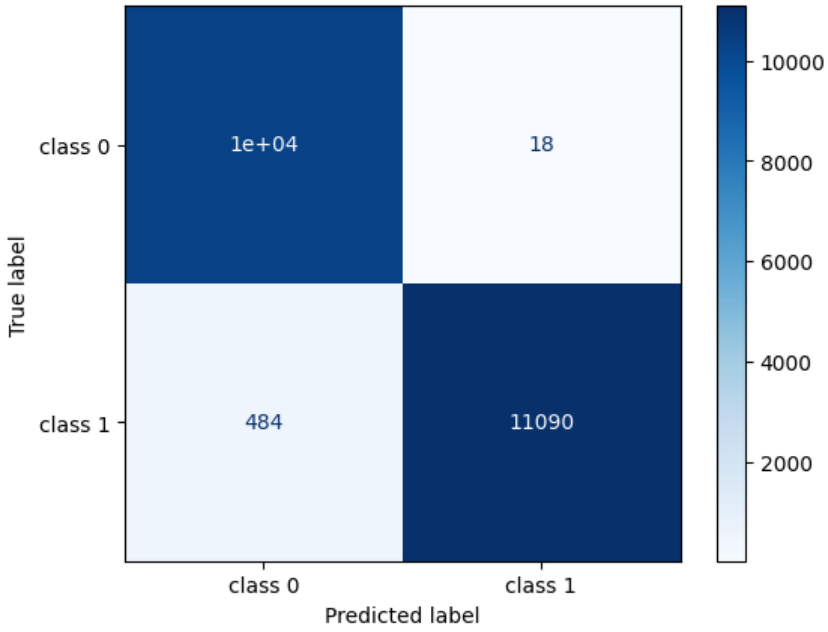

Figure 7. Confusion Matrix

6. Results

We showed in this paper a new approach for detecting SQL injection attacks based on Convolutional Neural Networks. In this approach, we leveraged the power of CNNs in automatically learning discriminative features from raw SQL query inputs, which would obviate the need for manual feature engineering. We could train the proposed CNN model on a huge dataset of SQL queries to a test accuracy of 98.16%, thus proving its effectiveness in identifying and classifying SQLIAs. The measured precision, recall, and F1-score for both benign and malicious queries were very high, hence confirming the model's performance.

These results show the potential of deep learning, especially CNNs, in augmenting web application security with a robust, adaptive solution for SQL injection detection and prevention. Some future research directions would be the investigation of more complex architectures of CNNs, additional features—for instance, contextual information, HTTP headers—and investigating the effectiveness of the model in real-world environments. We would like to extend this further and come up with a general framework that integrates our CNN-based detection system with all the security measures to ensure multilevel defense against SQLIAs.

References

1. OWASP. (2017). *OWASP Top 10, 2017*. [Online]. Available: https://www.owasp.org/index.php/Top_10-2017_Top_10
2. Q. Li, F. Wang, J. Wang, and W. Li, "LSTM-based SQL injection detection method for intelligent transportation system," *IEEE Trans. Veh. Technol.*, vol. 68, no. 5, pp. 4182–4191, May 2019.
3. H. Zhang, J. Zhao, B. Zhao, X. Yan, H. Yuan, and F. Li, "SQL injection detection based on deep belief network," *ACM Int. Conf. Proceeding Ser.*, 2019.
4. A. Luo, W. Huang, and W. Fan, "A CNN-based approach to the detection of SQL injection attacks," in *Proc. 18th IEEE/ACIS Int. Conf. Comput. Inf. Sci. (ICIS)*, Jun. 2019, pp. 320–324.
5. M. T. Muslihi and D. Alghazzawi, "Detecting SQL injection on web application using deep learning techniques: A systematic literature review," in *Proc. 3rd Int. Conf. Vocational Educ. Electr. Eng. (ICVEE)*, Oct. 2020, pp. 23–32.
6. M. Alghawazi, D. Alghazzawi, and S. Alarifi, "Detection of SQL injection attack using machine learning techniques: A systematic literature review," *J. Cybersecur. Priv.*, vol. 2, no. 4, pp. 764–777, Sep. 2022.
7. S. S. A. Krishnan, A. N. Sabu, P. P. Sajan, and A. L. Sreedeep, "SQL injection detection using machine learning," *Rev. Gestão Inovação Tecnol.*, vol. 11, no. 3, pp. 300–310, 2021.
8. J. K. R. Jothi, P. Beriwal, B. Saravana Balaji, A. Amarajan, and N. Pandey, "An efficient SQL injection detection system using deep learning," in *Proc. Int. Conf. Comput. Intell. Knowl. Econ. (ICCIKE)*, Mar. 2021, pp. 442–445.
9. W. Zhang, Y. Li, X. Li, M. Shao, Y. Mi, H. Zhang, and G. Zhi, "Deep neural network-based SQL injection detection method," *Security and Communication Networks*, vol. 2022, 2022, Art. no. 4836289.
10. X. Xie, C. Ren, Y. Fu, J. Xu, and J. Guo, "SQL injection detection for web applications based on elastic-pooling CNN," *IEEE Access*, vol. 7, pp. 151475–151481, 2019.
11. M. A. Al-Maliki and M. N. Jasim, "Review of SQL injection attacks: Detection, to enhance the security of the website from client-side attacks," *Int. J. Nonlinear Anal. Appl.*, vol. 13, no. 1, pp. 3773–3782, 2022.
12. A. Ketema, "Developing SQL injection prevention model using deep learning technique," Ph.D. dissertation, St. Mary's University, Twickenham, London, 2022.
13. D. Chen, Q. Yan, C. Wu, and J. Zhao, "SQL injection attack detection and prevention techniques using deep learning," *J. Phys.: Conf. Ser.*, vol. 1757, no. 1, p. 012055, 2021.
14. P. Tang, W. Qiu, Z. Huang, H. Lian, and G. Liu, "Detection of SQL injection based on artificial neural network," *Knowl.-Based Syst.*, vol. 190, p. 105528, 2020.
15. M. Hirani, A. Falor, H. Vedant, P. Mehta, and D. Krishnan, "A deep learning approach for detection of SQL injection attacks using convolutional neural networks," in *Proc. 4th Int. Conf. Trends Electron. Informat. (ICOEI)*, Jun. 2020, pp. 1183-1188.
16. Zoraez, A. M., Siddiqui, M., Imran, A., Yasin, A. U., Butt, A. H., & Paracha, Z. J. (2022). Performance evaluation of classification algorithms for intrusion detection on NSL-KDD using Rapid Miner. International Journal of Innovations in Science & Technology, 4(1), 135-146. 50sea.
17. Baig, M. S., Imran, A., Yasin, A. U., Butt, A. H., & Khan, M. I. (2022). Natural language to SQL queries: A review. International Journal of Innovations in Science & Technology, 4, 147-162. 50sea.
18. Shah, A. M., Aljubayri, M., Khan, M. F., Alqahtani, J., Sulaiman, A., & Shaikh, A. (2023). ILSM: Incorporated Lightweight Security Model for Improving QOS in WSN. Computer Systems Science & Engineering, 46(2).
19. Ullah, T., Khan, J. A., Khan, N. D., Yasin, A., & Arshad, H. (2023). Exploring and mining rationale information for low-rating software applications. Soft Computing, 1-26.
20. Khan, N. D., Khan, J. A., Li, J., Ullah, T., Alwadain, A., Yasin, A., & Zhao, Q. (2024). How do crowd-users express their opinions against software applications in social media? A fine-grained classification approach. IEEE Access.
21. Fatima, E., Kanwal, H., Khan, J. A., & Khan, N. D. (2024). An exploratory and automated study of sarcasm detection and classification in app stores using fine-tuned deep learning classifiers. Automated Software Engineering, 31(2), 69.