Application of Deep Learning Algorithm for Web Shell Detection in Web Application Security System

Rezky Yuranda[1]*, Edi Surya Negara[2]
Master of Information Engineering Study Program, Bina Darma University[1], [2]
Palembang, Indonesia
yurandarezky@gmail.com [1] e.s.negara@binadarma.ac.id [2]

***Abstract*— A web shell is a script executed on a web server, often used by hackers to gain control over an infected server. Detecting web shells is challenging due to their complex behavior patterns. This research focuses on using a deep learning approach to detect web shells on the ISB Atma Luhur web server, aiming to develop a model capable of precise detection. By training the model with labeled PHP files, malicious web shells are distinguished from benign files. The study is crucial for enhancing the server's security, preventing hacker attacks, and safeguarding sensitive data. Through preprocessing techniques such as opcode extraction and feature selection, useful pattern recognition for web shell detection is achieved. Training deep learning models like CNN and RNN with LSTM on processed data leads to accuracy evaluation using classification metrics. The CNN model demonstrates superior performance in detection, emphasizing the effectiveness of deep learning for web shell detection. The research contributes to enhancing security in web-based applications, protecting against cyber threats like web shells..**

Keywords*— W*ebshell, Deep Learning, CNN, RNN, LSTM

I. INTRODUCTION

A system is a collection of two or more components or subsystems that interact with each other to achieve a goal. In the context of an organization, a system comprises various components such as people, computers, information technology, and workflows. This system processes data into information to reach the established goals and objectives [1].

One notable application of this technology is web-based applications. These applications provide substantial benefits in terms of accessibility, as they can be accessed from any location and at any time, given that an internet connection and a web browser are available. Additionally, they eliminate the need for local installation, further enhancing their convenience and ease of use. [2].

Although web-based applications offer advantages in terms of accessibility, they also face several cyber threats. One such threat is the web shell, which poses a serious security risk to web-based applications [3]. A web shell is a script that can be executed by a web server, granting the user who has access to the server the ability to execute commands. An example of this is a PHP shell, which represents a type of web-based shell implementation. When a PHP shell is successfully uploaded, it allows an attacker to take control of the system or perform malicious actions, leading to web shell attacks or remote code execution [4].

Web shells are typically used by malicious actors to gain full control over infected servers. Detecting web shells presents a significant challenge due to the complex and dynamic patterns of behavior exhibited by these scripts [5]. In this study, a deep learning approach is employed to detect web shells on the ISB Atma Luhur web application server.

ISB Atma Luhur is also undergoing a migration process from desktop-based applications to web-based applications. This transition introduces new security challenges, including threats from web shells. Additionally, the increasing number of incidents involving web applications targeted by hacks and the dissemination of online gambling ads through web-based platforms underscores the critical need for enhanced security measures in web applications [6].

The primary objective of this research is to develop a deep learning model capable of detecting web shells with high precision. The choice of deep learning is based on previous studies comparing traditional machine learning and deep learning for text classification [7], [8]. This model will be trained using a labeled dataset of PHP files, where the label "malicious" indicates the presence of a web shell and the label "benign" indicates its absence.

For this study, the models to be evaluated are Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN) with Long Short-Term Memory (LSTM). Previous research has shown that LSTM models have achieved significant accuracy in similar contexts [9], [10] . Likewise, CNNs have proven effective for text classification based on prior studies [10]. By integrating these two approaches, it is anticipated that a deeper understanding of their application in this relevant context can be gained.

This study will focus on comparing the accuracy levels between LSTM and CNN models. This approach is crucial for identifying the relative strengths of each model within the chosen application context [11]. By evaluating their performance, it is expected to reveal both the advantages and limitations of each model, as well as potential optimization strategies to enhance their effectiveness in complex text classification tasks.

In previous research, Tianmin [5] conducted a study on webshell detection using Machine Learning methods by applying opcode+N-Gram+TF-IDF for sample characterization

p-ISSN 2301-7988, e-ISSN 2581-0588
DOI : 10.32736/sisfokom.v13i3.2234, Copyright ©2024
Submitted : July 26, 2024, Revised : August 26, 2024, Accepted : August 28, 2024, Published : November 20, 2024

and algorithms such as XGBoost, MLP, RF, and NB for model training. The results showed that the optimal detection model based on the XGBoost algorithm achieved an accuracy of over 97% under experimental conditions

Ngoc-Hoa [12] also proposed a model for detecting malicious code in PHP source files using a deep learning approach. This method involves pattern matching techniques by applying Yara rules to build malicious and benign datasets, converting PHP source code into numerical opcode sequences, and using a Convolutional Neural Network (CNN) model to predict whether a PHP file contains malicious code such as a webshell. Experimental results showed that the proposed method achieved an accuracy of 99.02% with a false positive rate of 0.85%.

This research is crucial for strengthening the security of the ISB Atma Luhur web application server. The results are expected to be used to protect the server from cyber threats and prevent damage to sensitive data..

II. RESEARCH METHODOLOGY

This research was conducted from April to July at ISB Atmaluhur, located at JL. Jendral Sudirman, Pangkalpinang City, Bangka Belitung Islands Province. The research location was chosen due to its relevance to the research objectives and the availability of the necessary data.

A. *Research Variables*

The research variables for the study on web shell detection using a deep learning approach can be divided into two main categories: independent variables and dependent variables. The independent variables are the input factors in this research and are not influenced by other variables, such as opcode and opcode extraction features. The dependent variable is the label indicating whether the PHP code extracted from the opcode is malicious or benign [13].

B. *Data Collection Methods*

In this research, data on web shells and normal PHP were downloaded from various available online sources. The web shells were collected from 17 project repositories, resulting in a total of 6021 web shells, which were then processed to 2918. The normal PHP files were sourced from CMS and several other open-source projects like WordPress, Yii2, Smarty, and CodeIgniter, in line with what is used at ISB Atma Luhur. The outcome was 2791 normal PHP files.

C. *Data Analysis Techniques*

In this research, data analysis techniques were implemented to detect the presence of web shells using a deep learning approach. An overview of the applied data analysis techniques can be seen in Figure 1.

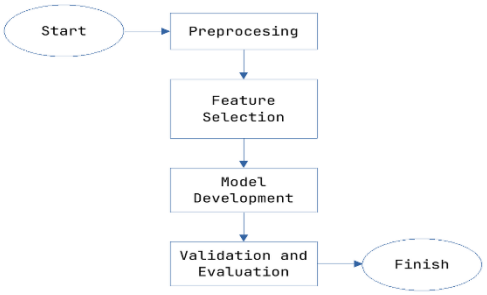

Fig 1. Research Flowchart

1. *Data Analysis Techniques*

At this stage, the collected PHP web shell files undergo a cleaning process. The objective is to extract PHP web shell files into opcodes in order to obtain a series of features that can detect potentially dangerous PHP web shells in the form of tokens. The preprocessing phase is illustrated in the figure.

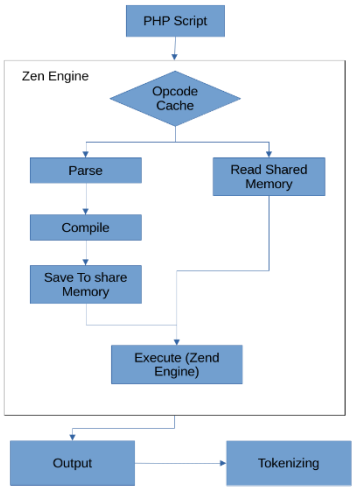

Fig 2. Preprocessing Data

PHP files will be extracted through the process of opcode extraction. Opcodes are the basic machine instructions generated from PHP code after it has been compiled or interpreted [5], [14].

After the extraction process, tokenization is performed. During this process, numbers, punctuation, and other characters deemed irrelevant for opcode processing are removed. Tokenization is crucial in data preprocessing.

2. *Feature Selection*

Feature selection is a crucial step aimed at enhancing and optimizing the efficiency of algorithms by simplifying index evaluation. Given the many opcode features with limited vocabulary, which result in sparse word vectors, it is essential to trim unnecessary features to maintain computational efficiency without compromising performance. To achieve this, Word2Vec, a method for word representation in a multidimensional vector space proposed by Mikolov [15], will be utilized. Specifically, the Continuous Bag of Words (CBOW) architecture will be employed to build the vector model.

p-ISSN 2301-7988, e-ISSN 2581-0588
DOI : 10.32736/sisfokom.v13i3.2234, Copyright ©2024
Submitted : July 26, 2024, Revised : August 26, 2024, Accepted : August 28, 2024, Published : November 20, 2024

3. Model Development

This process employs deep learning models to determine which model achieves the highest accuracy. The models considered are Convolutional Neural Networks (CNN) [10] and Recurrent Neural Networks (RNN) with Long Short-Term Memory (LSTM) [9], [10] for web shell detection. These models will be trained using the processed data, which will be divided into 80% training data and 20% testing data, to recognize patterns associated with web shells.

Fig. 1: CNN Architecture

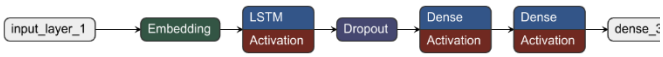

Fig. 2: RNN with LSTM Architecture

Fig 3 illustrates the architecture of a CNN consisting of several layers, starting with the input layer, followed by the Encoding layer, Conv1D, GlobalMaxPooling1D, two Dense layers, and finally the output layer. Fig 4 depicts the architecture of an RNN with LSTM, which includes the input layer, followed by the embedding layer, then the LSTM (Long Short-Term Memory) block, and three Dense layers with activation functions applied after the LSTM and on each Dense layer. The final part is a Dense layer with activation function 3, representing the output layer of this network

4. Validation and Evaluation

After completing the training and testing phases, the next step is to perform validation and evaluation using commonly applied metrics in classification, namely accuracy, precision, recall, and F1-score. These metrics will assist in assessing the model's success in predicting labels with high accuracy and in understanding how well the model avoids potential prediction errors [16].

Additionally, this study will evaluate the impact of the number of epochs and batch size on model performance, aiming to determine the combination that yields the best accuracy. This approach is expected to provide a comprehensive understanding of the effectiveness of the developed model.

D. Implementation

The implementation of web shell detection will be directly carried out by the researcher through an audit of a predetermined web application. After the testing phase is complete, the researcher will manually verify the findings to ensure that the suspected files are indeed web shells. This process aims to enhance detection accuracy and provide a deeper understanding of the potential threats in the application.

III. RESULT AND DISCUSSION

A. Data Collection

The data collection process resulted in a total of 2918 files labeled as "malicious" for web shells and 2793 files labeled as "benign" for normal PHP scripts. Each sample is categorized into two separate categories: "malicious" for web shells and "benign" for normal PHP.

TABLE I. DATA SOURCE

Sample	*Source*	*Amount*
Web Shell	https://github.com/Cyc1e183/PHP-Webshell-Dataset	2918
CMS/Framework	https://github.com/WordPress/WordPress	1447
	https://github.com/smarty-php/smarty	166
	https://github.com/yiisoft/yii2	1036

B. Data Preprocessing

At this stage, the collected data will be processed. The data preprocessing stage involves several steps:

1. PHP Opcode Extraction

The PHP opcode extraction stage is the initial step in data preprocessing, where PHP files are converted into opcode using the VLD (Vulcan Logic Dumper) extension. PHP files that are erroneous or inaccessible will be removed. An example of the PHP opcode extraction result can be seen in Figure 5 below.

```
~ » php -d vld.active=1 -d vld.execute=0 -f shell.php
Finding entry points
Branch analysis from position: 0
1 jumps found. (Code = 62) Position 1 = -2
filename:       /home/tabun/shell.php
function name:  (null)
number of ops:  6
compiled vars:  none
line     #* E I O op                           fetch          ext  return  operands
-------------------------------------------------------------------------------------
    1     0  E >   INIT_FCALL                                               'system'
          1        FETCH_R                      global             ~0      '_GET'
          2        FETCH_DIM_R                                     ~1      ~0, 'cmd'
          3        SEND_VAL                                                ~1
          4        DO_ICALL
          5      > RETURN                                                  1

branch: #  0; line:     1-    1; sop:     0; eop:     5; out0:  -2
path #1: 0,
```

Fig. 3: PHP Opcode Extraction Result

2. Tokenizing

In this process, numbers, punctuation, and other characters irrelevant to PHP opcode processing will be removed. This study utilizes the createToken function to filter opcodes using a set of PHP opcode keywords. After tokenization, the data will be stored in three columns: filename for the file name, result for the tokenized opcode results, and status, which indicates the security category with a value of 0 for malicious and 1 for benign. The final dataset can be viewed in the figure below.

	Filename	Result	Status
0	4e0f32bd4ea70c2afd7c9ce6eb643cec.php	DECLARE_CLASS RETURN RECV RECV RETURN RETURN I...	1
1	6c205875f5c645dc7929fe001df5334e.php	DECLARE_CLASS RETURN RECV RETURN RECV FETCH_OB...	1
2	5033b90aa7ea377c4c3f4d7441bab3df.php	INCLUDE_OR_EVAL INIT_FCALL_BY_NAME INIT_FCALL_...	1
3	0e14ed8da14c1d70ace2f4722562095c.php	INCLUDE_OR_EVAL INIT_FCALL_BY_NAME DO_FCALL BO...	1
4	244487d41a9a95adbcbed6f65c88e840.php	ASSIGN INIT_FCALL SEND_VAL DO_ICALL JMPZ INCLU...	1

Fig. 4: Preprocessed Data

After removing duplicate data, the dataset consists of 2,216 benign entries (68.6%) and 1,018 malicious entries (31.4%). The majority of the data is benign, with a smaller proportion of malicious entries. The diagram below visualizes the balance between benign and malicious categories in the cleaned dataset.

p-ISSN 2301-7988, e-ISSN 2581-0588
DOI : 10.32736/sisfokom.v13i3.2234, Copyright ©2024
Submitted : July 26, 2024, Revised : August 26, 2024, Accepted : August 28, 2024, Published : November 20, 2024

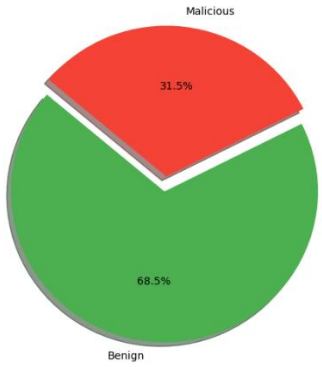

Fig. 5: Data Distribution

C. Feature Extraction

In this stage, the textual data resulting from the preprocessing step will be transformed into numeric representations understandable by deep learning models. This process consists of two key steps, primarily utilizing the Word2Vec model and constructing an embedding matrix.

1. Word Embedding Generation

Feature extraction employs the Word2Vec model to generate vector representations of words from the previously processed opcodes. The Word2Vec method used is the Continuous Bag-of-Words (CBOW) model with vector_size 100, window=10, min_count=5.

2. Embedding Matrix

After obtaining vector representations for each word, the next step is to construct an embedding matrix as the initialization weights for the embedding layer in the deep learning model. This process utilizes the Tokenizer from the Keras library to convert text into numeric representations, where each number represents a word in the dictionary. Once the text is converted into sequences of numbers, pad_sequences() is applied to ensure each sequence has a uniform length of 200 words. Finally, the embedding matrix (embed_matrix) is created as the embedding layer in the model, with dimensions corresponding to the vocab_size (vocabulary size) and dimension (embedding vector dimension).

D. Data Splitting

```
Y=keras.utils.to_categorical(df['Status'])
x_train,x_test,y_train,y_test=train_test_split(pad_rev,Y,test_size=0.20,random_state=42)
```

Fig. 6: Data Splitting

The model is divided using Keras's to_categorical function to convert categorical values into one-hot encoding, where [1] (benign) is converted to [0,1] and [0] (malicious) to [1,0]. The data is then split using scikit-learn's train_test_split function with test_size=0.20, so 20% of the data is used for testing and 80% for training.

E. Model Development

After the feature extraction stage is complete, the deep learning model construction begins. Two types of models are developed: a baseline CNN and an RNN with LSTM. The development process includes model initialization and training.

1. Model Architecture

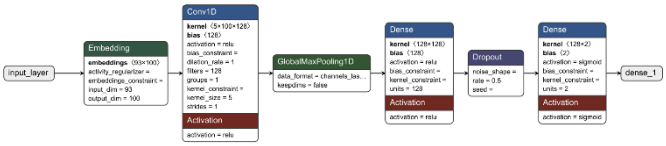

Fig. 7: CNN Model Architecture

The described Convolutional Neural Network (CNN) architecture begins with an input layer that receives text data in the form of word vectors or embedding matrices. This data is then processed by 1D convolutional layers, consisting of two consecutive Conv1D layers, each with a kernel size of 5, 128 filters, and a ReLU activation function. These layers capture local features from the text by performing one-dimensional convolution. After convolution, a GlobalMaxPooling1D layer is applied to reduce data dimensions by taking the maximum value from each feature, which reduces the number of parameters and helps prevent overfitting.

Next, the model includes two consecutive Dense layers. The first Dense layer has 128 units with a ReLU activation function and a dropout rate of 0.5 to prevent overfitting. The second Dense layer has 2 units with a sigmoid activation function, suitable for binary classification. Dropout is applied between these layers to reduce the risk of overfitting by randomly ignoring a portion of neurons during training. The ReLU activation function helps address the vanishing gradient problem and speeds up convergence, while the sigmoid activation function produces output in the range of 0 to 1. Finally, the output layer generates the final predictions, classifying the text into two categories, such as positive and negative.

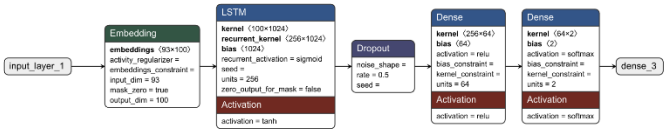

Fig. 8: RNN With LSTM Model Architecture

The RNN architecture with LSTM begins with the first layer being the Embedding Layer, which transforms input word indices into 50-dimensional vectors. With a specified input length and input dimension of 93, this layer converts word representations into a more meaningful form for further processing. Next, an LSTM Layer with 256 units returns sequences (return_sequences=True), allowing the model to retain sequential information while applying L2 regularization to the kernel, recurrent, and bias, and using the tanh activation function to process and filter sequential information.

Following that, a Dense Layer consists of 256 units, with a kernel size of 256x256 and bias size of 256x1, applying a max_norm(3) constraint on the kernel and bias and using the

p-ISSN 2301-7988, e-ISSN 2581-0588
DOI : 10.32736/sisfokom.v13i3.2234, Copyright ©2024
Submitted : July 26, 2024, Revised : August 26, 2024, Accepted : August 28, 2024, Published : November 20, 2024

ReLU activation function to introduce non-linearity. A Dropout Layer is applied with a dropout rate of 0.5 to reduce overfitting by randomly ignoring a portion of neurons during training, enhancing model generalization. An additional Dense Layer with a kernel size of 64x2 and bias size of 2x1 processes features from the previous layers and prepares data for the output layer. Finally, the Output Layer with 3 units determines the model's final output dimensions, enabling predictions in three different categories. The diagram visually illustrates how various layers can be arranged and connected to form a complex deep learning model.

2. *Training Result*

The model training is conducted by training the preprocessed data prepared in the data preprocessing stage. This process involves training over 20 epochs, meaning the entire training dataset is processed 20 times. During each epoch, the data is divided into small batches of size 64, allowing the model to be updated gradually and efficiently.
The overall training results can be seen in Figure 11 for the CNN model and Figure 12 for the RNN with LSTM model.

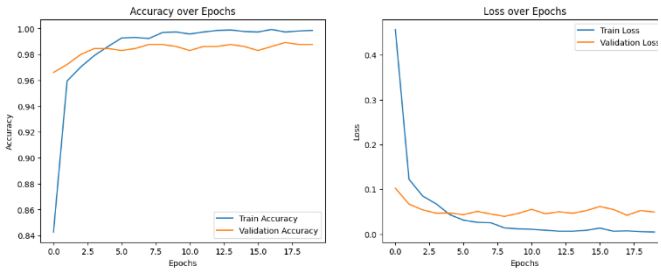

Fig. 9: Acuracy over epochs & loss over epochs CNN model

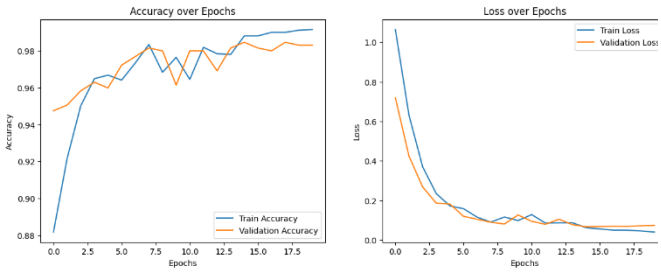

Fig. 10: Acuracy over epochs & loss over epochs RNN with LSTM Model

3. Testing Result

After the training process is completed, the model is then tested. The data used for testing is the testing data prepared beforehand in the preprocessing stage. The results of the testing can be seen in Table II.

TABLE II. TESTING MODEL RESULT

Model	*Accuracy*	*Loss*
CNN	0.9876	0.0490
RNN + LSTM	0.9830	0.0731

CNN shows a lower loss value compared to RNN, 0.0490 versus 0.0731. A lower loss indicates that CNN is more effective in minimizing prediction errors. Additionally, CNN also achieved a higher accuracy of 98.76%, compared to 98.30% achieved by RNN. Higher accuracy indicates that CNN is overall better in this classification context.

F. Evaluation

TABLE III. EVALUATION MATRIX

Matrix	**CNN Model**	**RNN Model**
Precision (kelas 0)	0.97	0.98
Recall (kelas 0)	0.99	0.96
F1-score (kelas 0)	0.98	0.97
Precision (keas 1)	1.00	0.98
Recall (kelas 1)	0.99	0.99
F1-Score (kelas 1)	0.99	0.99
Macro Average F1-Score	0.99	0.98
Weighted Average F1-Score	0.99	0.98

In terms of precision, recall, and F1-score, CNN demonstrates slightly better performance for class 0 (malicious), with a precision of 0.97, recall of 0.99, and an F1-score of 0.98. Conversely, RNN has a slightly higher precision of 0.98 for the same class, but its recall is slightly lower at 0.96, resulting in an F1-score of 0.97. For class 1 (benign), both models exhibit nearly equivalent performance, but CNN has a slight edge in precision and recall, with an F1-score of 0.99, compared to RNN's F1-score of 0.98. Overall, CNN shows higher macro and weighted average F1-scores of 0.99, compared to RNN's 0.98. This indicates that CNN has more consistent and balanced performance across all classes.

TABLE IV. CONFUSING MATRIX RESULT

Model	***Prediction***	***Actual***	
		True	***False***
CNN	**True**	452 (**TP**)	2 (**FP**)
	False	6 (**FN**)	188 (**TN**)
RNN + LSTM	**True**	453 (**TP**)	7 (**FP**)

p-ISSN 2301-7988, e-ISSN 2581-0588
DOI : 10.32736/sisfokom.v13i3.2234, Copyright ©2024
Submitted : July 26, 2024, Revised : August 26, 2024, Accepted : August 28, 2024, Published : November 20, 2024

	False	4 (**FN**)	183 (**TN**)

The confusion matrix provides additional insights into the types of errors made by each model. CNN shows fewer false positives (2) compared to RNN (7), indicating that CNN is less likely to incorrectly classify negative cases as positive. Additionally, CNN has more true negatives (188) than RNN (183). However, CNN also records more false negatives (6) compared to RNN (4), suggesting that CNN misses true positive cases slightly more often. True positives (451 for CNN and 453 for RNN) are nearly equivalent, indicating that both models perform similarly well in detecting positive cases.

In Table IV, the number of epochs and batch_size are also compared to achieve the highest accuracy. For the CNN algorithm, the highest accuracy achieved is 0.985468 (98.54%) with 100 epochs and a batch size of 32. Meanwhile, for the RNN algorithm, the highest accuracy obtained is 0.982743 (98.27%) with 70 epochs and a batch size of 32.

TABLE V. CONFUSING MATRIX RESULT

Epoch	**CNN**			**RNN**		
	32	**64**	**128**	**32**	**64**	**128**
10	0.981453	0.981453	0.982998	0.978362	0.976816	0.975270
20	0.982998	0.984544	0.982998	0.975270	0.982998	0.975270
30	0.984544	0.984544	0.986090	0.976816	0.976816	0.976816
40	0.984544	0.982998	0.982998	0.984544	0.978362	0.956723
50	**0.989181**	0.987635	0.984544	0.979907	0.981453	0.972179
60	0.984544	0.986090	0.986090	**0.987635**	0.978362	0.984544
70	**0.989181**	0.986090	0.987635	0.972179	0.981453	0.979907
80	0.979907	0.984544	0.987635	0.976816	0.984544	0.981453
90	0.975270	0.984544	0.984544	0.982998	0.986090	0.978362
100	0.986090	0.986090	0.986090	0.975270	0.982998	0.978362

In Table V, the number of epochs and batch_size are also compared to achieve the highest accuracy. For the CNN algorithm, the highest accuracy achieved is 0.989181 (98.91%) with 50 epochs and a batch size of 32. Meanwhile, for the RNN algorithm, the highest accuracy obtained is 0.987635 (98.76%) with 60 epochs and a batch size of 32.

G. Implementation

```
(venv) r3z@devz0ne [02:22:14 PM] [~/Documents/PyThon/opcode]
-> % python webshelldetection.py -d /mnt/alumni
```

Fig. 11: Insturctions for using the application

The command above will perform a check for web shells in the directory /mnt/alumni. After the checking process is complete, it will save logs named nama_directory_log.txt, nama_directory_malicious.txt,and nama_directory_benign.txt.

```
File: /mnt/alumni/wp-admin/includes/plugin.php, Classification: Benign, Probability: 100.00%
File: /mnt/alumni/wp-admin/index.php, Classification: Benign, Probability: 100.00%
File: /mnt/alumni/wp-admin/privacy-policy-guide.php, Classification: Benign, Probability: 100.00%
File: /mnt/alumni/wp-admin/options-media.php, Classification: Benign, Probability: 100.00%
File: /mnt/alumni/wp-admin/edit-form-comment.php, Classification: Benign, Probability: 100.00%
File: /mnt/alumni/wp-admin/media-upload.php, Classification: Benign, Probability: 100.00%
File: /mnt/alumni/wp-admin/export.php, Classification: Benign, Probability: 100.00%
File: /mnt/alumni/infokos.php, Classification: Webshell, Probability: 95.96%
File: /mnt/alumni/wp-comments-post.php, Classification: Benign, Probability: 100.00%
File: /mnt/alumni/wp-login.php, Classification: Benign, Probability: 100.00%
File: /mnt/alumni/wp-cron.php, Classification: Benign, Probability: 100.00%
File: /mnt/alumni/index.php, Classification: Benign, Probability: 99.98%

Total Webshell Files: 24
Total Benign Files: 1645
Execution Time: 78.02 seconds
```

Fig. 12: Results of web shell checking

From the implementation results, it is predicted that the total number of web shells is 24, and the total number of benign files is 1645. After manual inspection, one valid web shell was found by this model.

Before you begin to format your paper, first write and save the content as a separate text file. Keep your text and graphic files separate until after the text has been formatted and styled. Do not use hard tabs, and limit use of hard returns to only one return at the end of a paragraph. Do not add any kind of pagination anywhere in the paper. Do not number text heads-the template will do that for you.

Finally, complete content and organizational editing before formatting. Please take note of the following items when proofreading spelling and grammar:

IV. CONCLUSION

The results of testing and evaluating the CNN and RNN with LSTM models for webshell detection show that CNN performs slightly better compared to RNN + LSTM. CNN achieves an accuracy of 98.76% and a loss of 0.0490, whereas RNN + LSTM attains an accuracy of 98.30% and a loss of 0.0731. The lower loss for CNN indicates its superior ability to minimize prediction errors.

In terms of precision, recall, and F1-score, CNN also excels, particularly for class 0 (malicious) with a precision of 0.97, recall of 0.99, and an F1-score of 0.98. In contrast, RNN achieves a precision of 0.98, recall of 0.96, and an F1-score of 0.97. For class 1 (benign), although both models are nearly equivalent, CNN slightly outperforms with higher precision and recall, achieving an F1-score of 0.99 compared to RNN's 0.98.

Evaluation using the confusion matrix reveals that CNN has

p-ISSN 2301-7988, e-ISSN 2581-0588
DOI : 10.32736/sisfokom.v13i3.2234, Copyright ©2024
Submitted : July 26, 2024, Revised : August 26, 2024, Accepted : August 28, 2024, Published : November 20, 2024

fewer false positives (2) compared to RNN (7), indicating that CNN is more accurate in classifying negative cases. However, CNN records more false negatives (6) compared to RNN (4).

Overall, the CNN model proves superior in webshell detection compared to RNN with LSTM, particularly in terms of accuracy, loss, and evaluation metrics such as precision and recall. This research successfully achieves its primary goal of developing a deep learning model capable of detecting webshells with high precision. Additionally, these findings have the potential to enhance the security of web application servers at ISB Atma Luhur by providing a more effective solution for identifying and mitigating webshell threats

REFERENCES

[1] T. Sutabri, A. Wijaya, M. I. Herdiansyah, og E. S. Negara, "Evaluasi Risiko Celah Keamanan Aplikasi E-Office menggunakan Metode OWASP", EDUMATIC, bd. Vol. 8 No. 1, s. 113–122, jun. 2024, doi: 10.29408/edumatic.v8i1.25463.

[2] P. D. Yuningsih og L. A. Utami, "Sistem Informasi Online Booking Berbasis Web Pada Pheo Studi Salon", JURNAL TEKNOINFO, bd. 18, s. 193–200, 2024.

[3] S. Hartono og K. Khotimah, "Deteksi dan Mitigasi Serangan Backdoor Menggunakan Python Watchdog", Jurnal Sienna, bd. 3, s. 1, 2022.

[4] I. Putra, "Live Forensics untuk mengenali Karakteristik Serangan File Upload Guna Meningkatkan Keamanan pada Web Server: Indonesia", jiip, bd. 6, nr. 6, s. 4387–4394, jun. 2023, doi: 10.54371/jiip.v6i6.2173.

[5] G. Tianmin, Z. Jiemin, og M. Jian, "Research on Webshell Detection Method Based on Machine Learning", i 2019 3rd International Conference on Electronic Information Technology and Computer Engineering (EITCE), Xiamen, China: IEEE, okt. 2019, s. 1391–1394. doi: 10.1109/EITCE47263.2019.9094767.

[6] "GroboganKab-CSIRT". Set: 26. august 2024. [Online]. Tilgængelig hos: https://csirt.grobogan.go.id/posts/website-kampus-rentan-diserang-dan-dijadikan-jadi-situs-judi-online

[7] C. N. Kamath, S. S. Bukhari, og A. Dengel, "Comparative Study between Traditional Machine Learning and Deep Learning Approaches for Text Classification", i Proceedings of the ACM Symposium on Document Engineering 2018, Halifax NS Canada: ACM, aug. 2018, s. 1–11. doi: 10.1145/3209280.3209526.

[8] M. Zulqarnain, R. Ghazali, Y. M. M. Hassim, og M. Rehan, "A comparative review on deep learning models for text classification", IJEECS, bd. 19, nr. 1, s. 325, jul. 2020, doi: 10.11591/ijeecs.v19.i1.pp325-335.

[9] W. K. Sari, D. P. Rini, R. F. Malik, og I. S. B. Azhar, "Klasifikasi Teks Multilabel pada Artikel Berita Menggunakan Long Short- Term Memory dengan Word2Vec", Jurnal RESTI, bd. 4, nr. 2, 2020.

[10] P. Semberecki og H. Maciejewski, "Deep Learning methods for Subject Text Classification of Articles", præsenteret ved 2017 Federated Conference on Computer Science and Information Systems, Annals of Computer Science and Information Systems, sep. 2017, s. 357–360. doi: 10.15439/2017F414.

[11] I. P. Putri, T. Terttiaavini, og N. Arminarahmah, "Analisis Perbandingan Algoritma Machine Learning untuk Prediksi Stunting pada Anak: Comparative Analysis of Machine Learning Algorithms for Predicting Child Stunting", MALCOM, bd. 4, nr. 1, s. 257–265, jan. 2024, doi: 10.57152/malcom.v4i1.1078.

[12] N.-H. Nguyen, V.-H. Le, V.-O. Phung, og P.-H. Du, "Toward a Deep Learning Approach for Detecting PHP Webshell", i Proceedings of the Tenth International Symposium on Information and Communication Technology - SoICT 2019, Hanoi, Ha Long Bay, Viet Nam: ACM Press, 2019, s. 514–521. doi: 10.1145/3368926.3369733.

[13] L. T. Flannelly, K. J. Flannelly, og K. R. B. Jankowski, "Independent, Dependent, and Other Variables in Healthcare and Chaplaincy Research", Journal of Health Care Chaplaincy, bd. 20, nr. 4, s. 161–170, okt. 2014, doi: 10.1080/08854726.2014.959374.

[14] A. Karunaratne, "How to dump and inspect PHP OPCodes", PHP.Watch. Set: 26. august 2024. [Online]. Tilgængelig hos: https://php.watch/articles/php-dump-opcodes

[15] T. Mikolov, K. Chen, G. Corrado, og J. Dean, "Efficient Estimation of Word Representations in Vector Space", 6. september 2013, arXiv: arXiv:1301.3781. Set: 25. juli 2024. [Online]. Tilgængelig hos: http://arxiv.org/abs/1301.3781

[16] S. Sudianto, A. D. Sripamuji, I. R. Ramadhanti, R. R. Amalia, J. Saputra, og B. Prihatnowo, "Penerapan Algoritma Support Vector Machine dan Multi-Layer Perceptron pada Klasifikasi Topik Berita", Jurnal Nasional Pendidikan Teknik Informatika : JANAPATI, bd. 11, nr. 2, s. 84–91, aug. 2022.

p-ISSN 2301-7988, e-ISSN 2581-0588
DOI : 10.32736/sisfokom.v13i3.2234, Copyright ©2024
Submitted : July 26, 2024, Revised : August 26, 2024, Accepted : August 28, 2024, Published : November 20, 2024