electronics

MDPI

Article

Detection of Domain Name Server Amplification Distributed Reflection Denial of Service Attacks Using Convolutional Neural Network-Based Image Deep Learning

Hoon Shin [1], Jaeyeong Jeong [1,2], Kyumin Cho [3], Jaeil Lee [4], Ohjin Kwon [5] and Dongkyoo Shin [1,2,6,*]

[1] Department of Computer Engineering, Sejong University, Seoul 05006, Republic of Korea; kadosu@sju.ac.kr (H.S.); jaeyeong@sju.ac.kr (J.J.)
[2] Department of Convergence Engineering for Intelligent Drones, Sejong University, Seoul 05006, Republic of Korea
[3] Data Innovation Center, Financial Security Institute, Yongin-si 16881, Gyeonggi-do, Republic of Korea; gmcho@fsec.or.kr
[4] SmileGate, Seongnam-si 13493, Gyeonggi-do, Republic of Korea; jaeillee@smilegate.com
[5] Department of Electrical Engineering, Sejong University, Seoul 05006, Republic of Korea; ojkwon@sejong.ac.kr
[6] Cyber Warfare Research Institute, Sejong University, Seoul 05006, Republic of Korea
* Correspondence: shindk@sejong.ac.kr

Abstract: Domain Name Server (DNS) amplification Distributed Reflection Denial of Service (DRDoS) attacks are a Distributed Denial of Service (DDoS) attack technique in which multiple IT systems forge the original IP of the target system, send a request to the DNS server, and then send a large number of response packets to the target system. In this attack, it is difficult to identify the attacker because of its ability to deceive the source, and unlike TCP-based DDoS attacks, it usually uses the UDP protocol, which has a fast communication speed and amplifies network traffic by simple manipulating options, making it one of the most widely used DDoS techniques. In this study, we propose a simple convolutional neural network (CNN) model that is designed to detect DNS amplification DRDoS attack traffic and has hyperparameters adjusted through experiments. As a result of evaluating the accuracy of the proposed CNN model for detecting DNS amplification DRDoS attacks, the average accuracy of the experiment was 0.9995, which was significantly better than several machine learning (ML) models in terms of performance. It also showed good performance compared to other deep learning (DL) models, and, in particular, it was confirmed that this simple CNN had the fastest time in terms of execution compared to other deep learning models by experimentation.

Keywords: AI security; artificial intelligence; machine learning; deep learning; CNN; DDoS; DRDoS; DNS amplification DRDoS; image processing; image classification

Academic Editor: Wajeb Gharibi

Received: 9 November 2024
Revised: 20 December 2024
Accepted: 23 December 2024
Published: 27 December 2024

Citation: Shin, H.; Jeong, J.; Cho, K.; Lee, J.; Kwon, O.; Shin, D. Detection of Domain Name Server Amplification Distributed Reflection Denial of Service Attacks Using Convolutional Neural Network-Based Image Deep Learning. *Electronics* **2025**, *14*, 76. https://doi.org/10.3390/electronics14010076

1. Introduction

The increasing frequency and sophistication of DRDoS attacks are posing a serious challenge to maintaining the integrity and availability of online services. These attacks exploit normal protocols to overwhelm targeted systems with amplified traffic and disguise the source of the attack, making defense and attribution difficult.

Major incidents such as the Spamhaus, Dean, and GitHub attacks have highlighted their potential to cause widespread disruption and significant economic losses [1–3]. A number of high-profile sites have been taken down by large-scale DDoS attacks, leaving users without service. According to a recent report from Cloudflare, DNS-amplified DRDoS

attacks remain a persistent threat, with an 80% year-over-year increase [4]. Developing effective detection mechanisms that can operate in real time to stop these DRDoS attacks has become urgent. Deep learning is also a highly desirable approach to DRDoS traffic classification because it can be trained to autonomously select features.

The existing methods for detecting DRDoS attacks using deep learning have mainly relied on learning by extracting packet metadata [5,6], which can have limitations in detection due to the sophisticated evasion techniques used by attackers. They also have the disadvantage of requiring a significant amount of data preprocessing and additional processing time.

This research uses CNNs to convert packet data into image format for fast and efficient analysis. There are several ways to image packets, including extracting metadata values from packet headers [7], extracting metadata values from packet flows [6,8], and imaging the entire packet payload, such as the one proposed in this work [9–11]. While the full-payload imaging approach significantly reduces the preprocessing time required by the detection system and reduces the computational overhead associated with traditional metadata extraction, we speculate that it will be inferior to learning from metadata extraction in terms of accuracy. However, in this paper, we study a simple CNN structure that is suitable for detecting DNS amplification DRDoS attacks based on packet payload images and tuning the hyperparameters to achieve good speed and performance of DL.

The main objectives of this research are threefold: First, to build a simplified preprocessing mechanism through packet imaging [9,11] to achieve faster response time in DRDoS attack detection. Second, to design and optimize a lightweight and effective CNN model by tuning various hyperparameters [12]. Third, although deep learning models such as CNNs are promising for image-based data, their effectiveness may vary depending on the specific characteristics of the data. Therefore, we compare our proposed CNN-based model with existing machine learning and deep learning techniques to verify its effectiveness in maintaining high accuracy and execution speed on even low-dimensional image data (32×32 pixels). We use SVM, Random Forest, K-Nearest Neighbor (KNN), Logistic Regulation, and Gradient Boosting as the machine learning techniques we compare.

The deep learning techniques we compare with the CNN models implemented include Visual Geometry Group (VGG16), DenseNet121 (Densely CNNs), AlexNet, and Residual Networks50 (ResNet50), which are proficient in extracting image features, and Vision Transformer (ViT) and Shift Windows (SWIN) Transformer which are transformer models.

By achieving these objectives, this research aims to improve the responsiveness and efficiency of DRDoS detection systems, contributing to building a more resilient cybersecurity infrastructure capable of responding to the evolving DNS amplification DRDoS threat landscape.

2. Related Research

2.1. DNS Amplification DRDoS

Distributed Reflection DoS (DRDoS) is an attack technique in which multiple systems spoof the source IP to the IP of the system to be attacked so that the response packets are concentrated on the target system. It is characterized by the fact that the attacker spoofs the origin, making it difficult to recognize the attacker, and the existing legitimate servers become attack agents, or attack routes. The servers send response packets to the target system, and the transit servers designated by the attacker generate multiple response packets, causing a denial of service to the target system. On the other hand, a technique that increases the size of traffic by using a technique in which response packets are amplified compared to outgoing packets during a DRDoS attack is also called a DRDoS amplification attack.

DRDoS amplification attacks include Internet Control Message Protocol (ICMP) Flooding, TCP Flooding, DNS Amplification Flooding, Network Time Protocol (NTP) Amplification Flooding, Simple Network Management Protocol (SNMP) Amplification Flooding, Simple Service Discovery Protocol (SSDP) Amplification Flooding, and Memcached Amplification Flooding. In the case of DNS amplification DRDoS, as shown in Figure 1, the attacker impersonates the victim host and sends request packets to multiple Recursive DNS servers. The DNS servers then send response packets to the victim host, which can be larger than the request packets, causing a denial of service to the victim host. If you specify the record value as ANY type when querying the DNS server, the DNS server will send all the record information related to the query domain, which has the effect of amplification. According to HJ Kim et al. [13], an amplification of 264–499% was observed.

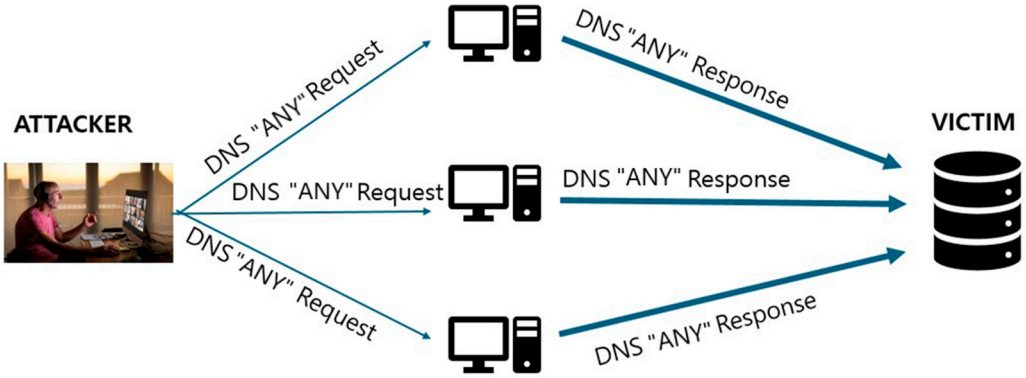

Figure 1. DNS amplification DRDoS operation process.

2.2. Deep Learning-Based Research of Network and DDoS Traffic Analysis

Mauro Conti et al. [14] categorized related work on network traffic analysis by three criteria: (1) the purpose of the analysis, (2) the point in the network where traffic is monitored, and (3) the ML algorithms that were reviewed, including Naïve Bayes, C4.5 Decision Trees, Random Forests, and K-means. The study focused on mobile devices and compared the analysis methods, validation techniques, and results achieved.

M. Yeo et al. [15] preprocessed data by extracting 35 features captured from packet flows as metadata and showed that they could classify 10 types of malware using deep learning on network traffic. They used CNN, Multi-Layer Perceptron (MLP), Support Vector Machine (SVM), and Random Forest (RF) for classification, with CNN and RF achieving over 85% accuracy, precision, and recall for all classes.

Aydın [16] proposed a Lightweight long Short-Term Memory (LSTM) model for real-time DDoS response in cloud environments, achieving 98.2% accuracy and a 97.5% detection rate, and showed a better detection rate and F1 score compared to existing ML models based on SVM and Decision Tree.

Diaba and Elmusrati [17] proposed a lightweight CNN and LSTM combined model for DDoS detection in smart grid environments, achieving 98.4% accuracy and showing a performance improvement of about 5% or more compared to Random Forest and KNN.

Gadze et al. [18] developed a deep learning model for DDoS detection, showing accuracy and recall of 99.4% and 98.7% with a CNN-LSTM hybrid model, respectively, and achieving more than 2% higher accuracy compared to SVM, Logistic Regression, and a single CNN model.

Aswad et al. [19] developed a lightweight CNN-LSTM model for DDoS detection in IoT networks, achieving 97.6% accuracy and 96.8% detection rate, with more than 6% performance improvement over SVM and Decision Tree models.

Ahmed et al. [20] used a hybrid model combining a CNN and Recurrent Neural Network (RNN) to learn network patterns and detect DDoS attacks, with 98.3% accuracy and a 97.6% detection rate, outperforming single CNN or RNN models by more than 3%.

While research on deep learning in the field of DDoS detection is actively being conducted, as research is focused on accuracy, the number of layers is increasing, and there is a lot of research on combining multiple deep learning models rather than a single model. To do this, it takes time to extract metadata in advance, and it also takes a lot of time and consumes a lot of resources to perform deep learning.

2.3. Imaging-Based Deep Learning Research of Network Traffic Analysis

K. M. Jeong [8] proposed a preprocessing technique suitable for deep learning by imaging network traffic, which is to read it in sessions (the termination condition is the receipt of a FIN flag or RST flag or no additional packets within 30 s of the last packet), save it as a txt file, and convert it to an N × N image file Portable Network Graphics (PNG) format.

As described by Yang, J et al. [10], the basic method of 2D-CNN is to convert byte data of network traffic into a byte-level greyscale image, and that of 3D-CNN is to convert traffic packet head data into a two-dimensional image and add a time based features which trained in the form of video.

For each session in the PCAP file, R. E. Davis et al. [7] generated a 50 × 50-pixel RGB image, with each row of the image representing exactly one packet. So, the first 50 packets of the session are used to generate the image. For each packet, 50 pixels are then used to display the fields in the IP header. Applying this approach to the Malware Capture Facility Project (MCFP) dataset, they were able to achieve a macro F1 score of 97% when classifying 142 malware classes with a CNN.

G. Bendiab et al. [11] used BinVis (a Binary Visualization tool) to generate an image from the pcap file and then classified traffic into eight classes, which are also associated with flows and sessions. Using this image with Resnet50, they obtained a 94.5% F1 score.

Kim. T et al. [21] improved the performance of intrusion detection using deep learning by converting network traffic into three-channel Red, Green, and Blue (RGB) color images instead of converting it to a typical grayscale image.

Y. He et al. [22] classified encrypted network traffic by converting encrypted traffic into grayscale images and classifying the converted grayscale images with a CNN, achieving an F1 score of 97.73% for encrypted traffic classification and 99.55% for virtual private network (VPN) classification.

In a study comparing the performance of deep learning models using color and monochrome images, J. Kim [23] generated CSE-CIC-IDS 2018, a dataset containing advanced DoS attacks such as DoS-Hulk, DoS-Slow HTTPTest, DoS-GoldenEye, DoS-Slowloris, DDoS-LOIC-HTTP, and DDoS-HOIC, in both RGB and monochrome images. They experimented with CNN models and found that the RGB images were more accurate than the grayscale images in both binary and multi-class classification. The CNN model achieved an average accuracy of 91.5%, while the RNN model achieved an average accuracy of 65%.

Hattak A et al. [9] analyzed 10 IoT network packet datasets, classified them with deep learning, and preprocessed PCAP files by cutting them to a certain size to create image files with RGB channels of 224 × 224 pixels. Experiments with CNN, MobileNet, Resnet50, and VGG16 showed that the CNN showed 99.1% accuracy in binary classification and 94.8% in multiple classification.

Wei Wang et al. [24] considered the bytes of network packet dataas natural language characters and used Natural Language Processing (NLP) text classification algorithms to

preprocess packet data and convert them into 28 × 28-pixel grayscale images with a size of 784 bytes. It was reported that 20 traffic classes, including malware, could be classified using a CNN on network traffic with an average accuracy of 99.41%.

A lot of research has been conducted on classifying network traffic through machine learning and deep learning by imaging network traffic. However, there is a problem in that it is not the network traffic itself that is imaged, but the metadata in the traffic is extracted, and then the image is processed, which consumes a lot of time and resources in the conversion of metadata to images.

2.4. Deep Learning Research of DNS Amplification DRDoS Traffic Analysis

I. H. Seo [5] extracted and compressed metadata from about 400 network packets for DNS amplification DRDoS detection, represented it as a 21 × 21 two-dimensional image, trained it with a CNN, and achieved 98.55% accuracy.

In Y. Gao [6], DNS amplification DRDoS attack traffic was analyzed, and the following five characteristics were extracted: the number of packets per time unit that contains only the IP header without the TCP or UDP header in the packet header, the size of UDP packets sent to the target per time unit, the total number of all packets sent to the target per time unit, the difference between the number of packets sent from the target and the number of packets sent to the target per time unit, and the maximum number of packets per time unit sent to the target out of all ports. They proposed a method of detecting attacks by training with SVM.

Machine learning research on DNS amplification DRDoS traffic also finds attack characteristics of DNS amplification DRDoS and extracts metadata to apply to machine learning and deep learning, adding a process to extract metadata. This makes it possible to perform postprocessing after an attack, but it is difficult to adapt and detect attacks in real time.

In this study, in order to solve the problems identified in the above study, such as the complexity of the process of extracting metadata, the increase in execution time, and the increase in accuracy, first, the process of extracting metadata is omitted, and the network packet itself is directly imaged and used as data for deep learning. Second, a model suitable for real-time DNS amplification DRDoS detection is introduced by implementing simple deep learning to show high accuracy while minimizing execution time.

3. DNS Amplification DDoS Detection System Based on Image Deep Learning

3.1. Create a Packet Image

3.1.1. Data Distribution

As we have seen in the previous related work, most deep learning requires extracting malware visualization and packet visualization metadata and preprocessing them into data suitable for deep learning. However, in this study, we applied the method of learning by imaging the malware binary itself to classify malware with deep learning [25] and studied whether it is possible to image packets without preprocessing and learn them with deep learning for meaningful detection.

First, we used the data visualization technique t-distributed Stochastic Neighbor Embedding (t-SNE), as shown in Figure 2, to explore the clustering, distances, and distribution of the dataset.

The normal DNS packet file (normal) shows several clear clusters, each with a different pattern, while the DNS amplification DRDoS packet file (attack) is more widely distributed than the normal DNS packets, showing a variety of shapes and independent features with few clusters.

Through this visualization analysis, we were able to identify differences such as non-linear similarities and clustering characteristics of the two-packet data and identify the potential for deep learning based on packet visualization.

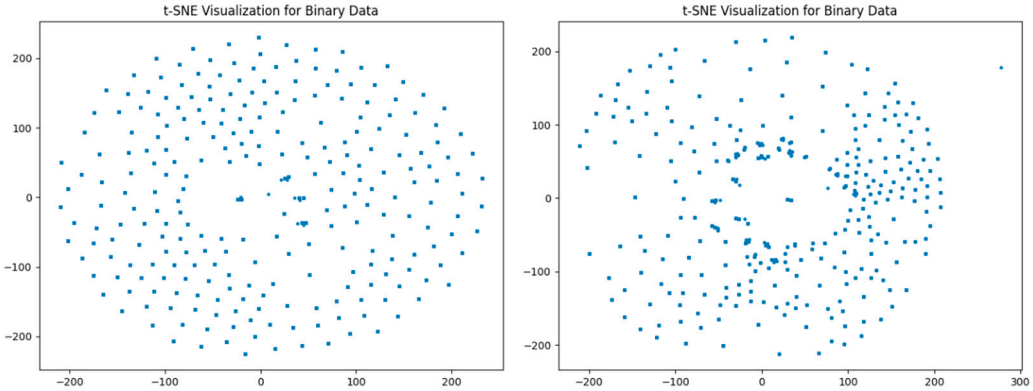

Figure 2. t-SNE visualization results (L: normal, R: DNS amplification DRDoS).

3.1.2. Data Preprocessing

The PCAP (packet capture) file, as shown in Figure 3, is a lossless file without compression and has a repeated structure of a packet header (16 bytes) and packet data; if you image the entire packet, the packet header will act as noise. Therefore, the file header and packet header are removed from the packet so that the actual packet data can be continuous.

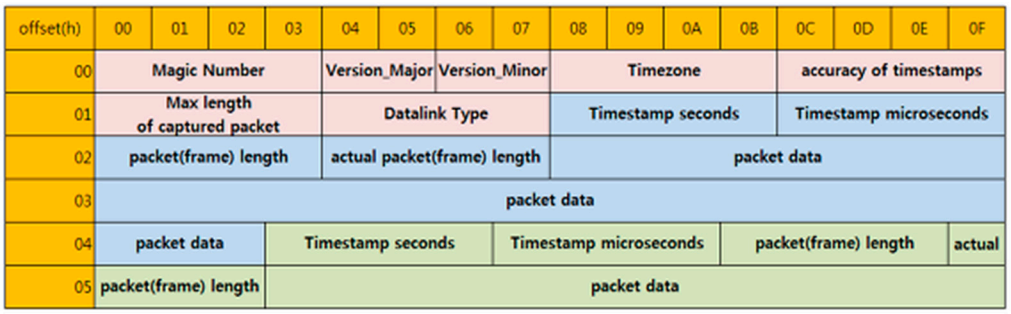

Figure 3. PCAP file structure (packet header and packet data).

Next, in order to classify the collected packets using image-based deep learning, we need to organize the packet data into images. Since packets come in different sizes, the larger the size, the larger the vertical length of the image. The image is converted into a square-shaped image for image-based deep learning training, and when visualizing traffic, R Kumar et al. [26] suggested that one pixel uses one byte; therefore, it can represent 256 levels of grayscale in the range of 0 to 255. R Kumar used this technique for malware preprocessing, but in this paper, the technique is applied to network packet preprocessing for image-based deep learning.

SY Lee et al. [27] converted malware files into images of 256 × 256 (65,536 bytes) size for malware file classification with a CNN, but for large image files, deep learning training time increases exponentially with image size. In our experiments, the generated images can extract enough visual features even at small sizes, so we made packet files as images of 32 × 32 horizontal and vertical lengths and used them in our experiments (Figures 4 and 5).

Figure 4. Example 32 × 32 DNS amplification DRDoS packet image.

Figure 5. Example 32 × 32 normal DNS packet image.

3.2. Selecting a Deep Learning Model

In the previous study, it was found that most of the deep learning analysis papers on network packet images mainly used a CNN, which has excellent performance in image processing, and there are many models that combine with other deep learning techniques, such as the LSTM model and comparison with RNN, to add temporal characteristics. Therefore, in this study, we selected a CNN as the basic deep learning model and experimented with it.

A basic CNN model, as shown in Figure 6, comprises the following layers: an input layer, which receives image input; a convolution layer, which employs filters to discern local patterns in the input image and generate a feature map; a pooling layer, which reduces the dimensionality of the data to prevent overfitting; a dense layer, which links all neurons in the input data to learn high-level global features; and an output layer, which represents the final output.

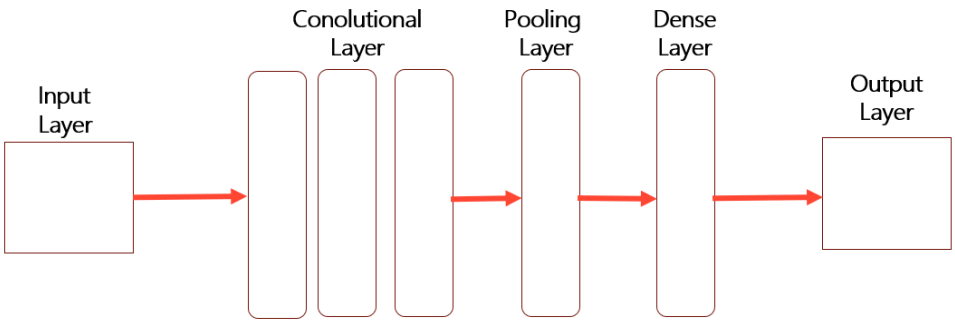

Figure 6. The basic CNN model structure.

3.2.1. Input Layer

The input layer, the first step of a CNN, is responsible for receiving the data, i.e., images, that the model will learn from. In this step, the pixel values of the image are passed as they are, and no additional calculations are performed. The input data are generally represented as a tensor in the form of height, width, and channel, and RGB images are processed with three channels and monochrome images with one channel. The important point at this stage is that the data must be normalized. This helps stabilize the model training and speed up the training process.

3.2.2. Convolution Layer

The convolution layer is a core element of a CNN, which learns various local patterns (e.g., edges, textures, shapes) that make up an image. In this layer, a filter (or kernel) slides over the input image and performs a convolution operation to generate a feature map. The output of the convolution layer may be smaller than the input image, and the generated feature map is passed to the next step. The initial convolution layer learns low-level features, while the later layers learn high-level features.

3.2.3. Pooling Layer

The pooling layer reduces the spatial size of the input feature map, which helps reduce the computational load of the model and prevent overfitting. This is a process of removing unnecessary details while retaining important information in the input data. The pooling layer reduces the size of the input data, which reduces the amount of computation and allows the model to learn more efficiently. It also increases the robustness of the model to distortions in spatial positions.

3.2.4. Dense Layer (Fully Connected Layer)

The dense layer connects all the neurons of the input data in the last stage of a CNN to learn high-level global features. This layer serves to map the input data to a final class or value using the features extracted in the previous layer. The activation function (ReLU, sigmoid) is used to apply nonlinearity to dense layer.

3.2.5. Output Layer

The output layer of a CNN provides the final prediction based on the output results of the dense layer. The structure and activation function of this layer vary depending on the type of problem.

A CNN mainly uses the input layer, convolution layer, and pooling layer, which extract features from each image, and the pooling layer simplifies the image by deleting the parts that are not the main features.

The more convolution layers, the more sophisticated the model, but the longer the training time. Since the dense layer, which is a Fully Connected Layer, can only learn one dimension, the multidimensionality is reduced to one dimension through flattening, and then the data are extracted through the dense layer, leaving only important features.

3.3. System Configuration

This section details the configuration of the CNN-based detection system designed to detect DNS amplification DRDoS attacks as shown in Figure 7.

The packet data are converted to greyscale images by cropping them to the appropriate size, normalizing the pixel values [0, 255] to the range [0, 1], and then assigning an appropriate class (category) to each image by labeling in image classification. This labeling constitutes an important preprocessing step for the model, as it enables the learning of input images.

Data augmentation is employed to generate novel data by transforming existing data, thereby enhancing the model's generalization performance. Subsequently, we devise a CNN classifier, which constitutes the core of the CNN model.

The convolution layer is responsible for the extraction of features and the addition of Max Pooling, which serves the dual purpose of reducing the dimensionality of the dataset and preventing overfitting.

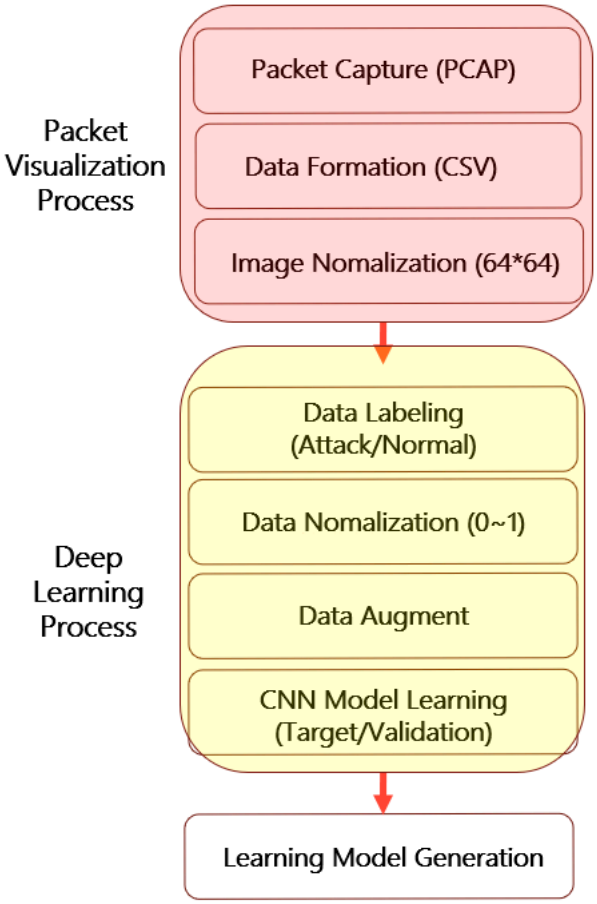

Figure 7. DNS amplification DRDoS detection system configuration diagram.

3.3.1. Packet Image Processing and Normalization

The packets are collected, cropped to a suitable size, and imaged in grayscale. For a detailed description of data preprocessing, please refer to Section 3.1.2. For normalization, each image is normalized to convert pixel values [0 to 255] to the range [0, 1].

The activation function, the sigmoid function, is optimized for input and output values in the range [0, 1], which is numerically more stable than large values between 0 and 255. The activation function, normally the Rectified Linear Unit (ReLU) function, adds nonlinearity to the neural network and works better with normalized inputs.

3.3.2. Data Labeling

In image classification, labeling refers to the process of assigning an appropriate class (category) to each image data point. Labeling is an important preprocessing step that helps the model learn the input image and output class. Here, the dataset is labeled as Attack and Benign.

3.3.3. Data Augmentation

A technique that generates new data by transforming existing data to improve the generalization performance of a model in deep learning. It is mainly used in various types of data, such as images, text, and audio, and is useful for solving the problem of insufficient data and preventing overfitting.

3.3.4. CNN Classifiers

The CNN model used in this paper consists of two convolutional layers and two dense layers. The convolution layer extracts features from the entire image and adds a Max

Pooling layer to extract only important features to achieve the benefits of dimensionality reduction, overfitting prevention, and transition invariance.

Meanwhile, in this convolution layer, the activation function is based on ReLU, which imposes nonlinearity to learn complex patterns and alleviates the Vanishing Gradient Problem, where the output of a traditional sigmoid function is limited to between 0 and 1, and the gradient approaches zero as the input value increases. However, one of the drawbacks of ReLU is that for negative inputs, neurons can be completely deactivated and not contribute to learning. To solve this problem, the Leaky Rectified Linear Unit (Leaky ReLU) was invented. It is a modified version of ReLU, where ReLU is an activation function that makes the output zero when the input is less than zero. Instead of making the output zero for negative values, Leaky ReLU leaves a small gradient for negative values, which allows the neurons to learn without dying when the input is negative.

A 32×32 image is filtered with 32 3×3 kernels in the first convolution layer, and then padding is added so that the image size is not lost, resulting in 32 32×32 images. In this model, Leaky ReLU, which showed the best performance through multiple experiments, is applied as an activation function to the generated image, and then a 2×2 Max Pooling layer is used to generate 32 16×16 images.

The second convolution layer takes 32 16×16 images as input and filters them into 64 3×3 kernels to generate 64 16×16 images. After applying Leaky ReLU to the generated images, they are passed through a 2×2 Max Pooling layer to generate 64 8×8 images.

The 4096 values of the generated image as one feature data point are processed through the Flatten layer to convert the three-dimensional data into one-dimensional data, and 64 values are output through the dense layer.

After dropout regularization, which randomly deactivates some neurons to prevent overfitting, the CNN model is a binary classification model. Therefore, the number of output neurons in the last output layer, the dense layer, is 1, and the neurons are subjected to a sigmoid activation function to output a probability between 0 and 1 (if the output value is more than 0.5, the model is classified as class 1, and if it is less than 0.5, it is classified as class 0). The optimizer is Adaptive Moment estimation (Adam).

4. Experiment

The detection system experiment was implemented in python, and the deep learning model was written and executed in Tensorflow with the Google Colaboratory (Colab) environment with the Tensor Processing Unit (TPU).

4.1. Datasets

The DNS amplification DRDoS data are from the "DDoS Packet Capture Collection" provided by L.F. Haaijer for DDoS research [28], which categorizes DNS amplification DRDoS pcap files as DRDoS packets.

The "DDoS Packet Capture Collection" is an anonymized collection of DDoS packet captures provided by him, as it is difficult to obtain actual DDoS packet data. These data can be freely used to build and improve DDoS defense systems. The data are anonymized with the victim's address and port. This project is still receiving real-time DDoS traffic packet sources, and the normal DNS service data are from DNS protocol packets (54,863) from a small router of a university (Table 1).

We experimented with a total of 7216 training images and 3646 benign images, as shown in Table 2, with 70% of the data used for training and 30% for validation. We divided the dataset into training and validation data to train deep learning and derive the detection accuracy.

However, to address the lack of data and ensure data diversity, data augmentation was used to transform or expand the original data in various ways to solve data scarcity, prevent overfitting, and improve the model's generalization ability. Image data augmentation methods in this experiment include rotation, flipping, cropping, resizing, and translation.

Table 1. Structure of Benign and Attack datasets.

	Attack	**Benign**
Total Packets Count	12,000	54,863
Average Packet Size (bytes)	512	122
UDP Configuration Ratio (%)	74.9	98.4

Table 2. Amount of Benign and Attack of the training and validation data.

Training (70%)		**Validation (30%)**	
Attack	Benign	Attack	Benign
4075	3141	2057	1589

4.2. Performance Evaluation Metrics

4.2.1. Confusion Matrix

To evaluate the performance of a deep learning model, we used a confusion matrix, which is a tool used to evaluate the performance of a classification model. It is a visual representation of the predicted categories versus the actual data, and it is used to evaluate the performance of a model to see how accurately it is classified.

Table 3 shows the confusion matrix, denoting True Negative (TN) when the model predicts normal DNS traffic and the actual DNS traffic is normal, False Positive (FP) when the model predicts DNS attack traffic but the actual traffic is normal, False Negative (FN) when the model predicts normal DNS traffic but the actual traffic is an attack, and, finally, True Positive (TP) when the model predicts DNS attack traffic and the actual file is also DNS attack traffic.

Table 3. Confusion matrix.

	Predicted Positive	**Predicted Negative**
Actual Positive	True Positive	False Negative
Actual Negative	False Positive	True Negative

These four TN, FP, FN, and TP can be utilized to calculate accuracy, precision, recall, F1 score, and more. Accuracy is skewed in unbalanced datasets. For this reason, we used the following secondary metrics. Precision and recall are interdependent metrics, and arbitrarily increasing one can cause the other to drop.

- Accuracy = (TP + TN)/(TP + TN + FP + FN), which is the percentage of how accurately it predicts the input data as the number of correctly predicted data out of the total data.
- Precision = TP/(TP + FP), which is the percentage of samples predicted to be positive that are actually positive.
- Recall = TP/(TP + FN), which is the percentage of true positives that the model correctly predicts as positive.
- F1 Score = 2 × (Precision × Recall/Precision + Recall). The F1 score is an index created by combining precision and recall, which is the harmonized average of precision and recall. It is used for accurate evaluation between precision and recall in unbalanced classes.

4.2.2. Receiver Operating Characteristic Curve

The Receiver Operating Characteristic (ROC) curve, as shown in Figure 8, is often used as a performance evaluation method in binary classification, and a good model of classification will have a curve close to the left corner. A quantitative way to evaluate this is the Area Under Curve (AUC), which represents the area under the RoC curve and is referred to as AUROC and has a maximum value of 1.

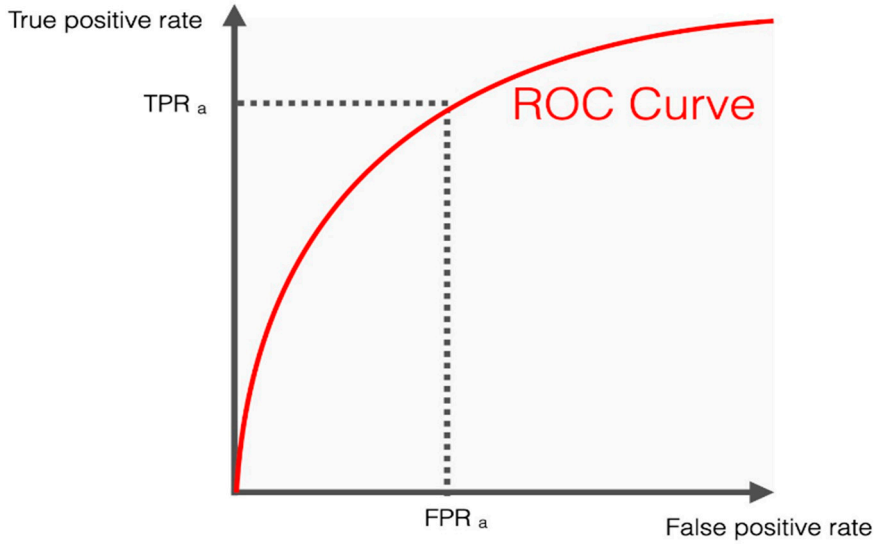

Figure 8. ROC curve.

4.3. Experimental Results

To realize the optimal CNN model, we adjusted the kernel size for each convolutional layer to find the optimal combination of kernel sizes and experimented with the CNN model by changing the output size of the dense layer, learning rate, dropout, and epoch values to compare the performance. Finally, we evaluated the performance of the proposed model by comparing its performance with machine learning methods used for image classification.

4.3.1. Numbers of CNN Convolution Layers and Numbers of Kernels

In a CNN model, each kernel is responsible for extracting a specific pattern or feature from the input data. The more kernels there are, the more different features the network can learn. For example, the first layer might detect basic patterns like edges or corners, while subsequent layers recognize more complex shapes or objects. As the number of kernels increases, the expressive power of the model increases. This increases the ability to model complex patterns in the data. Therefore, deeper and more complex CNN architectures typically use more kernels.

However, an excessively large number of kernels can increase the risk of overfitting. This means that the performance on the training data will be higher, but the ability to generalize to new data may be lower. Setting the right number of kernels is therefore very important. A higher number of kernels increases the amount of computation and memory consumption required, which may require more efficient hardware and more time.

The number of kernels is an important hyperparameter of a CNN. It controls the depth and width of the CNN network and plays an important role in tuning it for optimal performance.

On the other hand, the higher the number of CNN layers, the deeper and more complex features the network can extract from the input data. Early layers recognize simple patterns (e.g., edges, textures, etc.), while deeper layers recognize higher-dimensional structures (e.g., objects, faces, etc.). Therefore, stacking multiple convolutional layers contributes to improving the performance of the model. However, more layers means

higher computational cost and memory usage, longer training time, and possibly more powerful hardware. Therefore, when designing a CNN, you need to consider the tradeoff between performance and resource consumption. Depending on the purpose of the CNN model (image classification, object detection, etc.), the optimal number of convolutional layers can vary. Choosing the best number of layers for a particular problem is important to maximize performance: too many layers can overfit the training data, resulting in poor generalization performance, so choosing the right number of layers and using regularization techniques (e.g., dropout, batch normalization, etc.) is important.

We experimented with two convolutional layers and three convolutional layers, and we found that only two convolutional layers were sufficient for the 32 × 32 image size dataset. We first experimented with three layers and found that the accuracy was above 0.998, but two layers also showed an accuracy between 0.996 and 0.998 in a convolutional neural network (CNN), so we chose two convolutional layers. In this study, we used two convolutional layers to find the optimal kernel size for each convolutional layer by varying the kernel size to 8, 16, 32, 64, 128, and 256 for each convolutional layer. We aimed to find a stable model that has better performance than the best performance but does not cause overfitting (hyperparameter value: epoch 40, learning rate 0.001, dropout 0.5, dense layer output 128, kernel size 3 × 3). The hyperparameter value was continuously modified to a value that showed stable and excellent performance as the experiment progressed.

The performance was better when more kernels were applied per layer, and more detailed features could be extracted as the next layer was added. The best accuracy (0.9997) was achieved with 64 and 128 kernels in each of the two layers. However, the performance was good enough with a smaller number of kernels, so we considered a smaller size.

Theoretically, the more the number of kernels, the better the performance, but in the case of the CNN7, CNN8, and CNN10 models in Table 4, even though they are a combination of a large number of kernels, their performance was slightly lower than that of CNN6, which had a small number of kernels. Therefore, the number of kernels did not always mean better performance.

Table 4. Accuracy comparison at different kernel sizes at 2 convolution layers (CLs).

CNN	CL1	CL2	Accuracy
CNN1	8	8	0.9970
CNN2	8	16	0.9929
CNN3	16	16	0.9984
CNN4	16	32	0.9984
CNN5	32	32	0.9986
CNN6	32	64	0.9995
CNN7	64	64	0.9984
CNN8	32	128	0.9984
CNN9	64	128	0.9997
CNN10	128	128	0.9989
CNN11	128	256	0.9995

In this study, we used the CNN6, CNN9, and CNN11 models, which showed an accuracy of 0.9995 or higher, as shown in Figure 9, to find the optimal number of kernels. Among them, we selected the CNN6 model (32, 64) with the smallest number of kernels as the main experimental model.

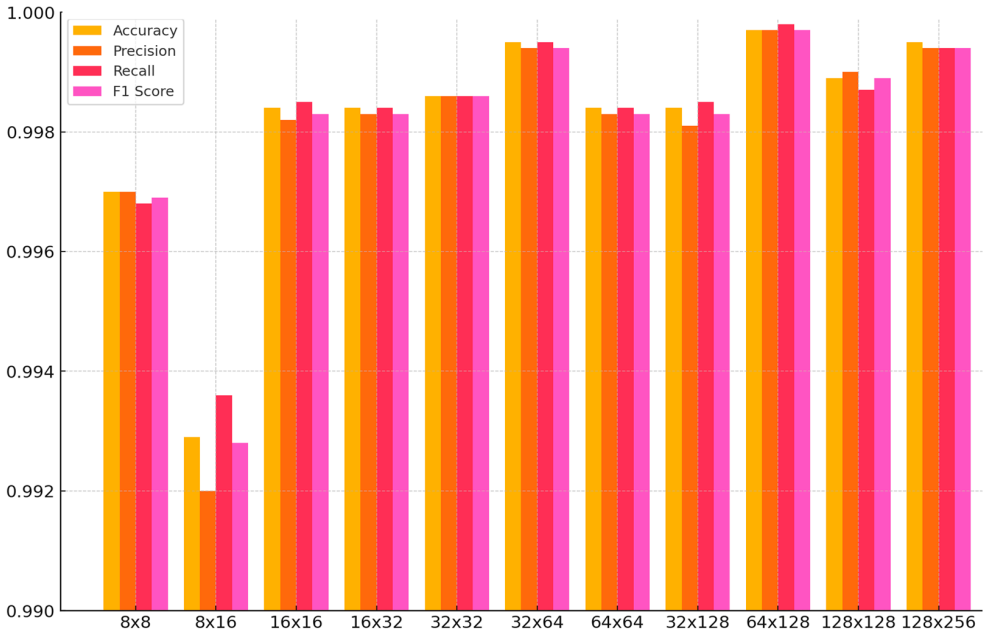

Figure 9. Performance comparison at different kernel sizes at 2 convolution layers (CLs).

4.3.2. Dense Layer Output Adjustment

In a CNN, the dense layer brings together multiple features learned by the neural network in previous stages to create more detailed and precise information. For example, a neural network that classifies images initially recognizes basic features like color, shape, and size. Dense layers help it combine these basic features to come to a conclusion about "what this is". The neurons in the dense layer (Fully Connected Layer) are connected to all the inputs from the previous layer and add nonlinearity through their activation functions.

For example, a dense layer neuron count of 64 indicates that there are 64 neurons in this layer. Each neuron receives an input vector, applies a weight and bias, passes it through an activation function, and produces an output. The experimental results show that 4096 neurons perform best, but this is considered to be an excessive number of neurons for a model for processing 32 $\times$ 32 images. To find the most appropriate dense layer output for the CNN6 model, we compared the accuracy by doubling the output from 64 to 4096 (hyperparameter value: epoch 30, learning rate 0.001, dropout 0.5, kernel size 3 $\times$ 3) and found that the accuracy is between 0.997 and 0.999 and that the execution time increases logarithmically when the output doubles. We compared the performance through experiments to find a value that represents a small but effective performance tradeoff. Since the number of 4096 neurons is inefficient due to increased computational cost and execution time, we chose 64 neurons, which is the smallest number of neurons with relatively good performance for the purpose of fast detection rather than a perfectly high-performance model, as shown in Table 5.

Table 5. Performance by dense layer (DL) output (neuron count).

DL Output	Accuracy	Precision	Recall	F1 Score
64	0.9986	0.9986	0.9986	0.9986
128	0.9986	0.9986	0.9986	0.9986
256	0.9986	0.9985	0.9987	0.9986
512	0.9978	0.9976	0.9980	0.9978
1054	0.9989	0.9990	0.9988	0.9989
2048	0.9986	0.9986	0.9986	0.9986
4096	0.9997	0.9997	0.9998	0.9997

Except for the dense layer value of 512, most of the models have accuracy above 0.998, so we compared the training accuracy and validation accuracy graphs of the models. When comparing Figures 10–12, which have dense layer outputs with an accuracy of 0.998 or higher, we can see that the training and validation accuracy is stable when the output is 64, and the rest of the models are unstable at the beginning of epochs 1~20.

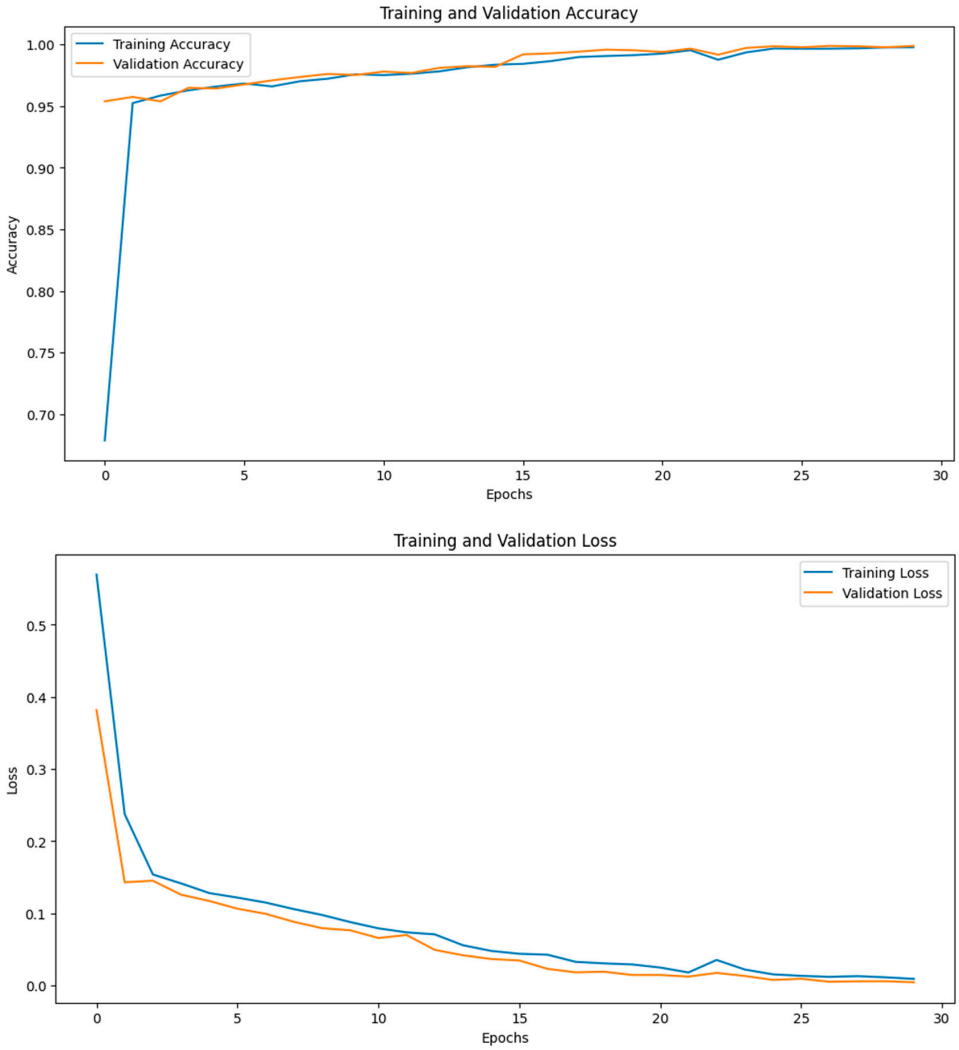

Figure 10. Performance graph of dense layer output 64.

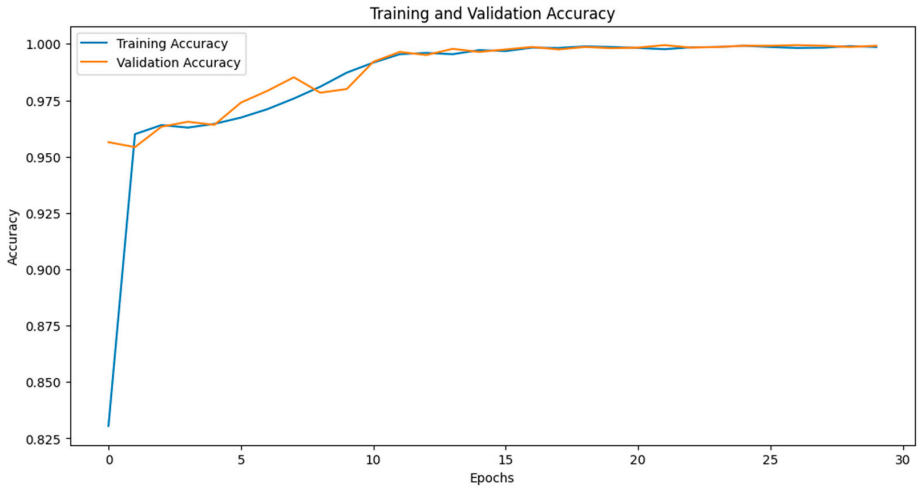

Figure 11. *Cont*.

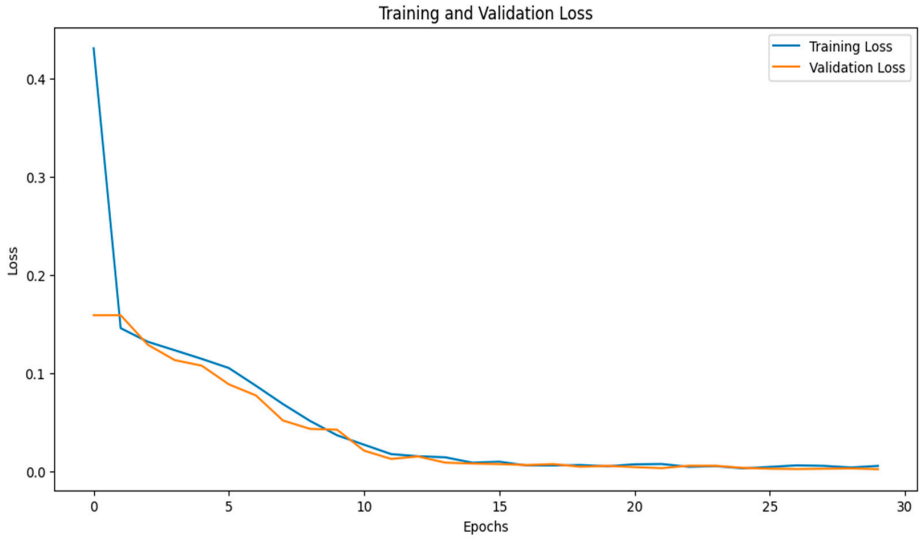

Figure 11. Performance graph of dense layer output 1024.

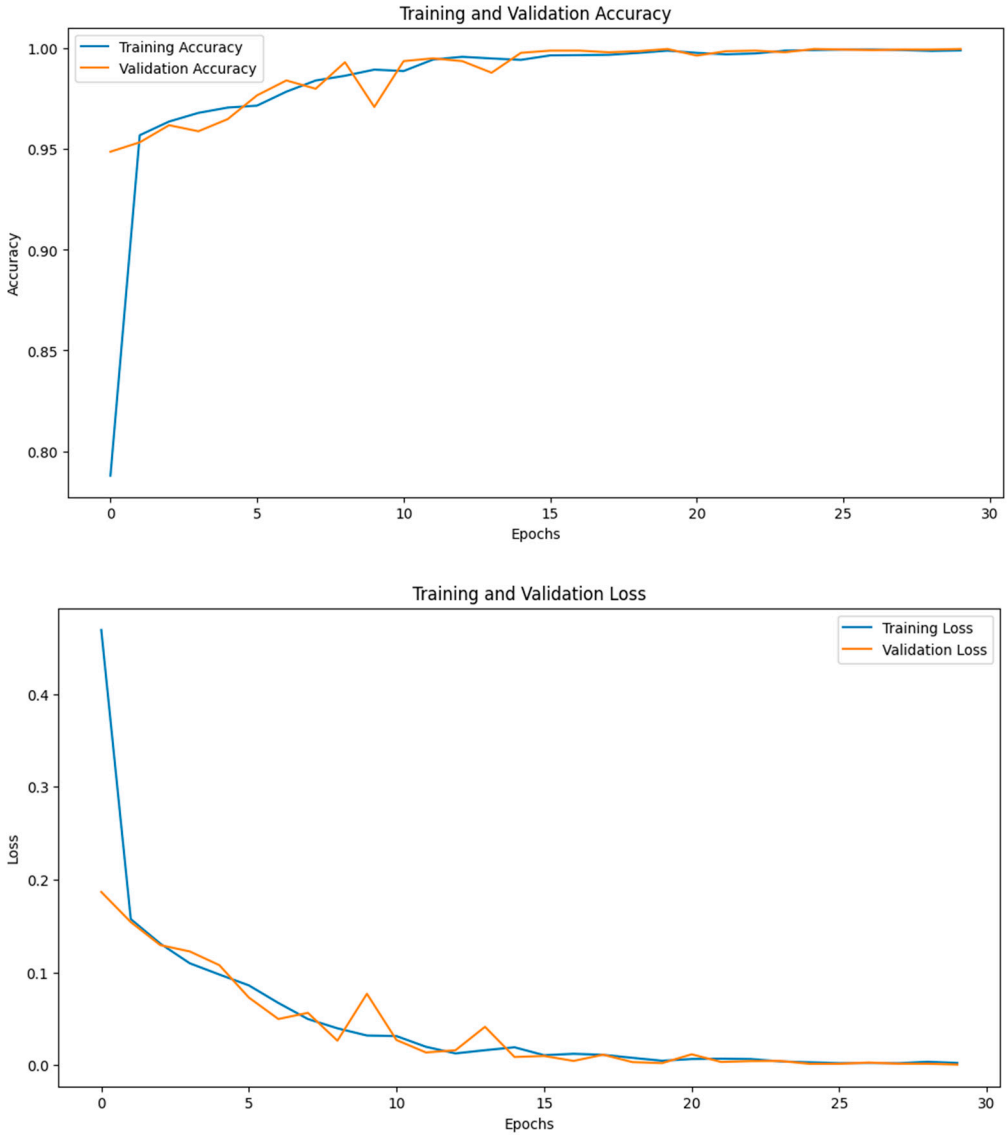

Figure 12. Performance graph of dense layer output 4096.

4.3.3. Adjusting the Learning Rate

If the learning rate is large, the speed is fast, but it does not reduce the error in the training process, and overfitting occurs, and if the learning rate is too low, the training process takes a long time, and the error value in the validation is too large. In a CNN, the learning rate is an important hyperparameter used to update the weights of the model. This value determines how much to adjust the weights along the gradient of the loss function. The learning rate is the scaling factor used when adjusting the weights in each training iteration, meaning it determines how much to move in the direction calculated by the gradient.

A learning rate that is too large can result in too large weight updates, which can cause the model to overfit or diverge from the optimal value, while a learning rate that is too small can result in smaller weight updates, slowing convergence and increasing training time. An appropriate learning rate helps the model converge to the optimal value quickly and reliably and improves the model's ability to generalize.

To find the right learning rate for the CNN6 model, we varied the learning rate from 0.01 to 0.00001 and measured its performance, as shown in Table 6 (hyperparameter value: epoch 40, dense layer output 64, dropout 0.5, kernel size 3 × 3).

Table 6. Performance comparison of the learning rate.

Learning Rate	Accuracy	Precision	Recall	F1 Score
0.01	0.9997	0.9998	0.9997	0.9997
0.001	0.9986	0.9986	0.9986	0.9986
0.0001	0.9789	0.9769	0.9812	0.9786
0.00001	0.9515	0.9497	0.9567	0.9511

In the experiments, 0.01 had the best performance, but it was unstable because it often overshot or diverged from the optimal value, and 0.0001 showed increasingly better performance when running more than 200 epochs, but it took too long to run. For the CNN6 model, we decided to use a learning rate of 0.001, which consistently performed better than 0.9980 after 50 epochs, in line with our goal of fast learning and detection rather than perfect performance., It is very stable when normally the high learning rate is 0.0001 and 0.00001, but performance can drop significantly when the number of epochs is low, requiring hundreds of iterations to obtain adequate performance.

4.3.4. Adjusting Kernel Size

In a CNN, varying the size of the kernel affects performance in many ways. An odd-sized kernel allows for symmetrical processing and smooth padding for specific locations in the input data. Odd-sized kernels also have an advantage when applying padding to input data. Padding allows certain areas to be processed while maintaining the size of the input data, and odd-sized kernels add padding more evenly to handle pixels on the edges more effectively.

Small kernels, such as 3 × 3, are good for learning complex patterns and can stack multiple layers to preserve fine image details. For example, architectures like VGG16 primarily use 3 × 3 kernels to perform image classification with high accuracy. On the other hand, larger kernels, such as 7 × 7, can capture large features or patterns in a single computation, which is advantageous for recognizing the overall structure, but accuracy may suffer because large kernels significantly reduce the spatial dimension in a single computation. Large kernels can also effectively mitigate overfitting when there is sometimes not enough training data. As shown in Table 7, we experimented with kernel sizes of 3 × 3, 5 × 5, and 7 × 7 and found that the 3 × 3 kernel performance value, as

shown in Figures 13 and 14, and the 3 × 3 kernel stability of the performance graph is the best, so we applied it to the CNN6 model (hyperparameter value: epoch 40, dense layer output 64, learning rate 0.001, dropout 0.5).

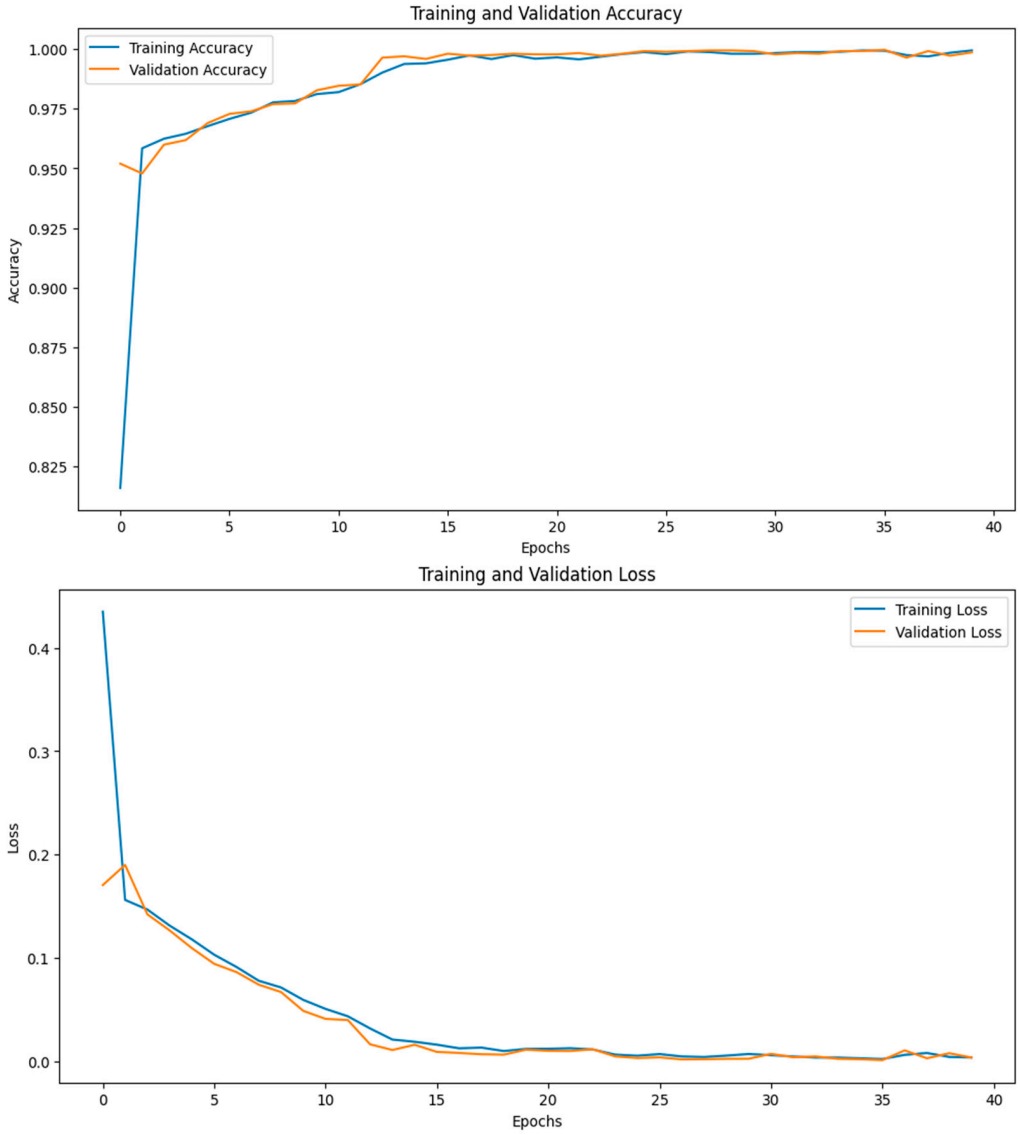

Figure 13. Performance graph of kernel size 3 × 3.

Table 7. Comparison of the kernel size.

Kernel Size	Accuracy	Precision	Recall	F1 Score
3 × 3	0.9995	0.9994	0.9995	0.9994
5 × 5	0.9981	0.9981	0.9980	0.9980
7 × 7	0.9953	0.9949	0.9957	0.9957

4.3.5. Epoch Changes

In deep learning, an epoch refers to the entire training dataset being passed through the model one time. During each epoch, the model looks at the training data and updates its weights, and by training over multiple epochs, the model gradually learns the patterns in the data. If the number of epochs is too large, the model may overfit the training data and generalize poorly to new data. We measured the performance of the CNN6 model as

the epoch value varied from 10 to 100, as shown in Tables 8 and 9 (hyperparameter: dense layer output 64, learning rates 0.001/0.0001, dropout 0.5, kernel size 3 × 3).

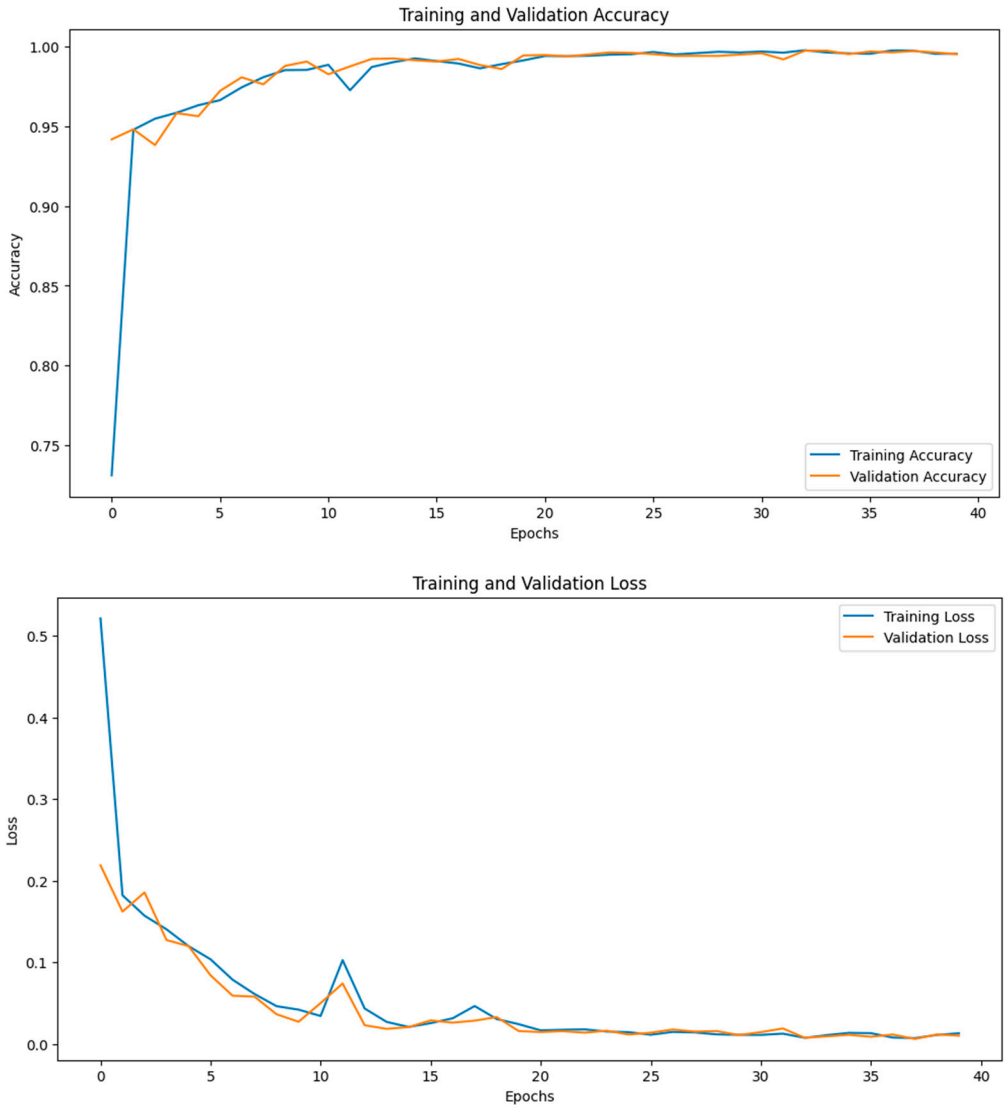

Figure 14. Performance graph of kernel size 7 × 7.

For a learning rate of 0.001, the accuracy is better than 0.9995 after epoch 50, as shown in Figure 15. For a smaller learning rate, such as 0.0001, like in Table 9, it learns quite finely and stably, but it does not reach 0.999 until more than 200 epochs, showing that it learns slowly compared to a learning rate of 0.001.

Table 8. Comparison of epoch changes (LR = 0.001).

Epoch	Accuracy	Precision	Recall	F1 Score
10	0.9748	0.9727	0.9774	0.9745
20	0.9964	0.9962	0.9966	0.9964
30	0.9975	0.9972	0.9977	0.9975
40	0.9992	0.9993	0.9991	0.9992
50	0.9995	0.9994	0.9994	0.9994
60	0.9997	0.9997	0.9998	0.9997
80	0.9995	0.9994	0.9994	0.9994
100	0.9995	0.9995	0.9994	0.9994

Table 9. Comparison of epoch changes (LR = 0.0001).

Epoch	Accuracy	Precision	Recall	F1 Score
50	0.9737	0.9716	0.9760	0.9734
100	0.9904	0.9894	0.9913	0.9903
200	0.9989	0.9990	0.9988	0.9889

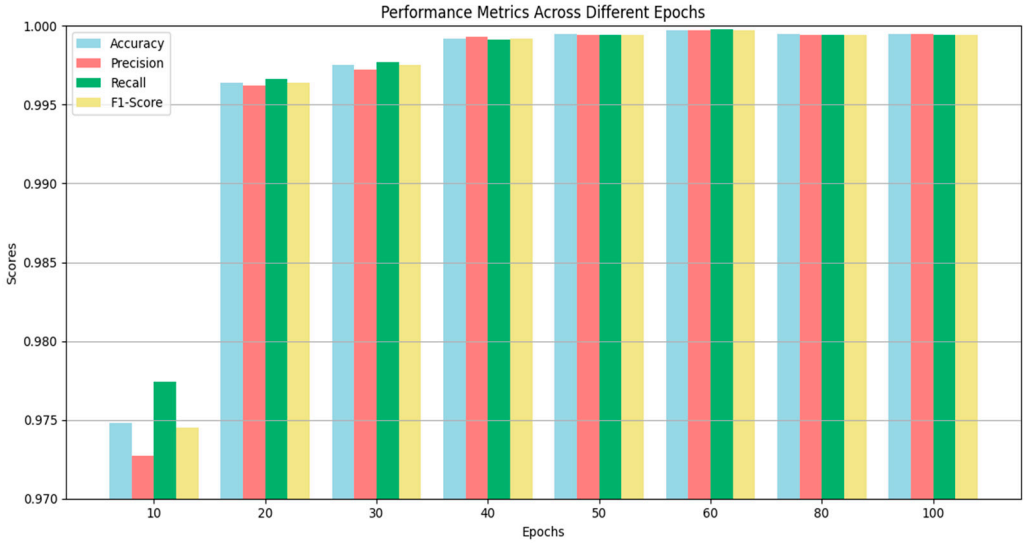

Figure 15. Performance comparison by the epoch value (LR = 0.001).

4.3.6. Adjusting Dropouts

Dropout is a regularization technique used in CNNs to prevent overfitting. It is a generalization technique that randomly disables some neurons during the training process to prevent the model from becoming overly dependent on certain neurons. Dropout is a method of randomly "turning off" some neurons during the learning process so that only a small portion of the network is used. This process ensures that the neural network does not rely on specific neurons or pathways.

Dropout prevents the model from becoming overly dependent on a particular pattern or noise, and dropout has the effect of training multiple different neural networks. This is because a randomly selected neuron is activated in each training iteration, resulting in different combinations of neural networks each time the training is repeated.

If the dropout rate is too small, overfitting can occur, and if it is too high, many neurons in the neural network will randomly turn off during training, causing the neural network to lose important information that it needs to learn. As a result, the model will not be able to learn useful patterns properly, which can lead to poor performance. This is why you need to choose an appropriate dropout value. For example, a dropout of 0.1 randomly deactivates only 10% of the neurons during training. However, in this case, the deactivation rate is small, and the generalization performance is poor. Here, we varied the dropout from 0.1 to 0.9 in the CNN6 model to find the optimal value. In the experimental results, as shown in Table 10, we found that the performance is excellent at 0.3 dropout, and the learning pattern is stable in the training and validation graphs (hyperparameters: epoch 50, dense layer output 64, learning rate 0.001, kernel size 3×3).

Table 10. Performance comparison by dropout value.

Dropout	Accuracy	Precision	Recall	F1 Score
0.1	0.9989	0.9990	0.9988	0.9989
0.3	0.9995	0.9994	0.9994	0.9994
0.5	0.9986	0.9988	0.9984	0.9986
0.7	0.9989	0.9989	0.9989	0.9989
0.9	0.9978	0.9975	0.9981	0.9978

Comparing the training and validation accuracy and loss graphs, as shown in Figures 16 and 17, at dropout 0.9, it can be seen that the difference between the training and validation graphs is large, indicating underfitting. It can be seen that 0.3 is more stable, so 0.3 was selected for the CNN6 model lastly.

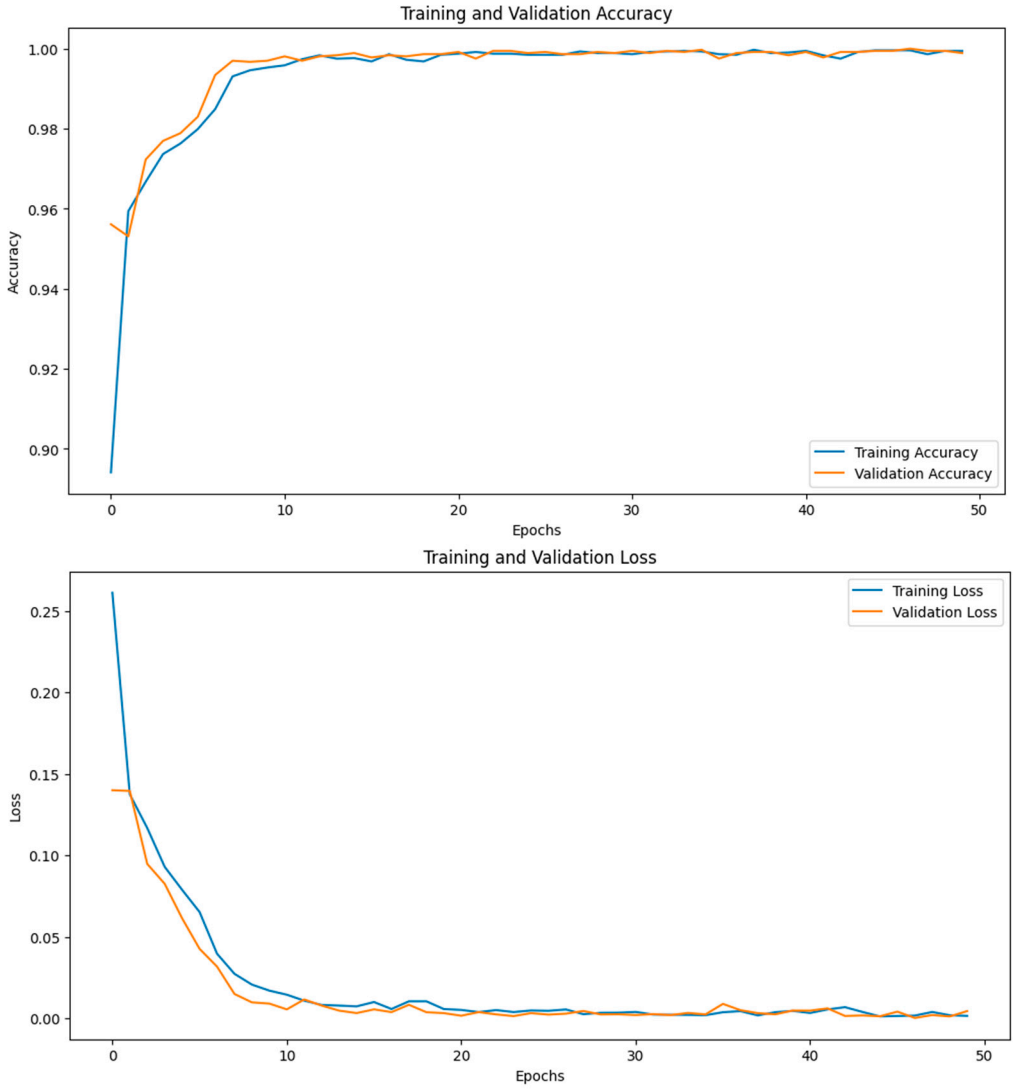

Figure 16. Learning performance with a dropout of 0.3.

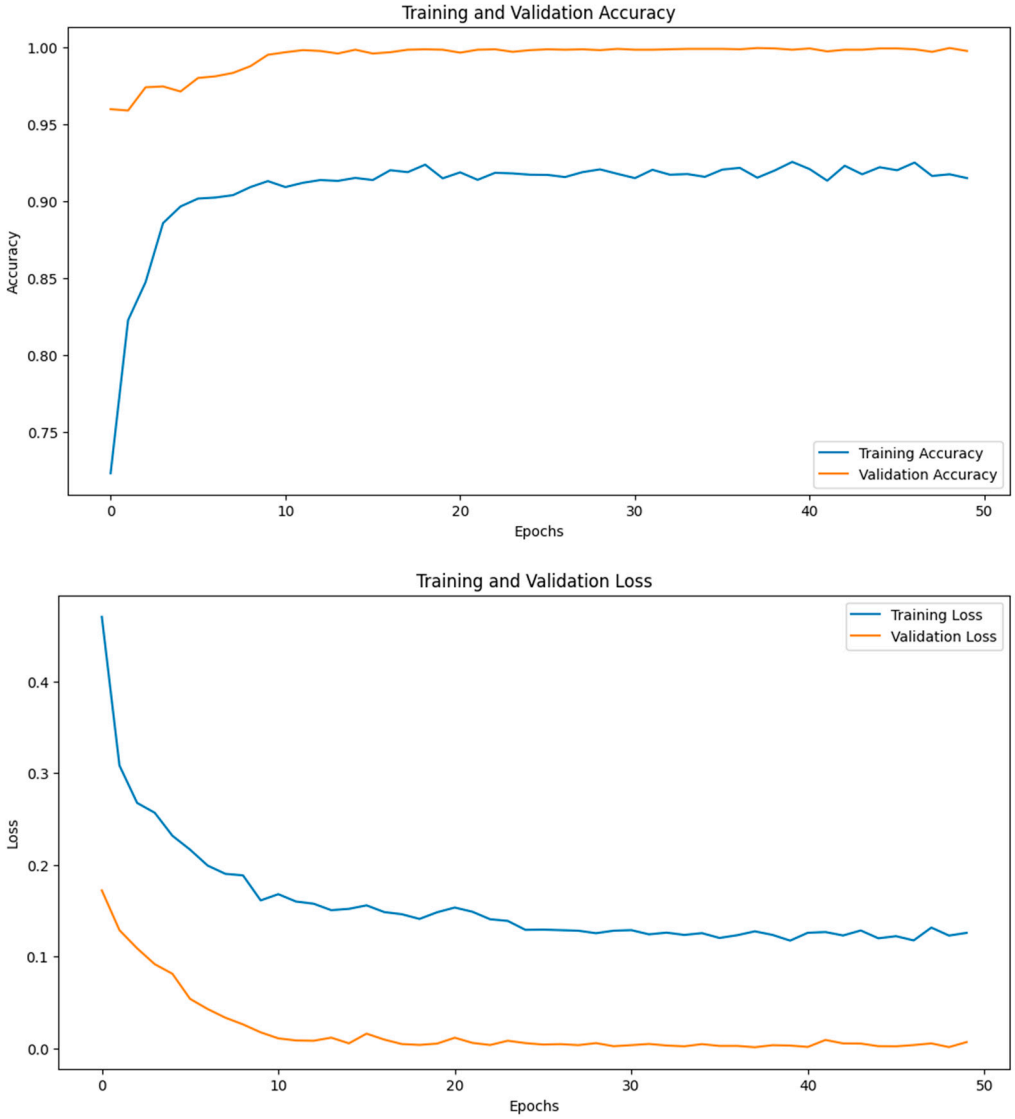

Figure 17. Learning performance when the dropout is 0.9.

4.3.7. Comparison of Activation Functions

In a CNN, the activation function is responsible for nonlinearly transforming the model's learned output. Without activation functions, a CNN would only perform simple linear transformations between inputs and outputs, preventing it from learning complex patterns. The use of activation functions transforms the output of each layer nonlinearly, allowing a CNN to learn different data patterns and model complex relationships. A CNN is composed of multiple layers, and the activation function adds nonlinearity to allow each layer to go beyond simple linear transformations to learn complex features of the image data, emphasizing or suppressing certain features at each layer so that the next layer can learn more meaningful patterns.

For example, an activation function such as ReLU, which converts negative numbers to zero and removes certain activations, helps to alleviate the problem of gradient vanishing during the learning process so that the learning signal can be effectively transmitted to deeper layers during backpropagation. Activation functions such as ReLU, Leaky ReLU, Exponential Linear Unit (ELU), sigmoid, and Hyperbolic Tangent (Tanh) are used in CNNs, with ReLU being the most used, and recently, variations of ReLU, such as Leaky ReLU and ELU, have been utilized to improve performance.

ReLU outputs 0 if the input is less than 0, and it outputs the input value as it is if it is greater than 0. This is simple to compute, mitigates the problem of gradient vanishing, and provides fast learning because the gradient is always 1 for positive inputs. However, if the input is negative, the gradient can go to zero, causing a "dead neuron" (the dead ReLU problem).

The Tanh function scales an input value between -1 and 1. It outputs 0 when the input is 0, and the smaller the input value, the smaller the output. Since the output is zero-centered, it can average the data to zero, which helps it converge faster. However, it is still susceptible to gradient vanishing, especially in deep networks.

Leaky ReLU is a variant of ReLU that maintains a small slope for negative inputs, meaning that it outputs a value with a slight slope if the input is below zero. Leaky ReLU can solve the problem of dead ReLU, and it reduces information loss by maintaining the slope on negative inputs. However, it still has a very small slope for inputs below zero, which can limit the flow of information.

ELU takes the form of an exponential function when the input is less than or equal to zero and outputs the input value as it is when it is greater than zero. ELU can adjust the average output closer to zero, which helps it converge faster. It can also solve the dead ReLU problem, but its computational cost is higher than ReLU, which can affect performance in large models. In this model, we selected Leaky ReLU as the best-performing activation function based on our experiments, as shown in Table 11 and Figure 18 (hyperparameters: epoch 50, dense layer output 64, learning rate 0.001, dropout 0.3, kernel size 3×3).

Table 11. Performance comparison by activation function.

Activation Function	Accuracy	Precision	Recall	F1 Score
ReLU	0.9995	0.9994	0.9994	0.9994
Tanh	0.9986	0.9984	0.9988	0.9986
Leaky ReLU	0.9997	0.9996	0.9997	0.9997
ELU	0.9975	0.9972	0.9978	0.9975

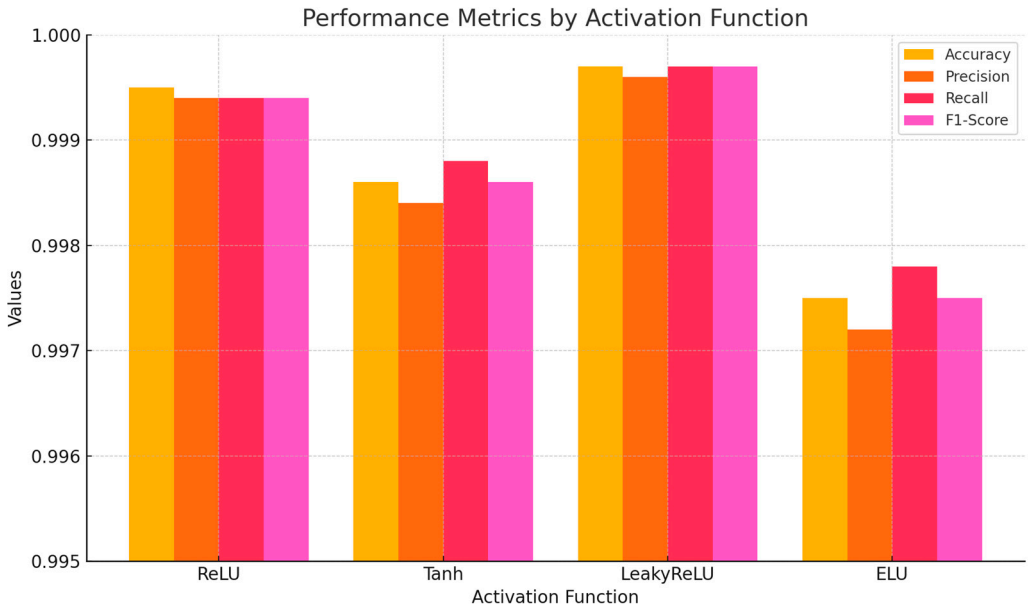

Figure 18. Performance comparison by activation function.

4.3.8. Comparing Loss Functions

In the training of a machine learning model, a loss function serves to measure the difference between the output value predicted by the model and the actual correct answer value to evaluate how badly the model predicted during the training process. Without a loss function, the model would have no basis for judging the accuracy of its predictions, making it impossible to learn, so the loss function quantifies the value of these wrong predictions and gives the model a goal to minimize this value. By quantifying the difference between the predictions and the actual values, the loss function ensures that the model's weights are updated during the optimization process, thus guiding the CNN to learn to make increasingly accurate predictions.

A large value for the loss function indicates that the model makes a lot of errors, so an optimization algorithm (e.g., gradient descent) will adjust the model's weights in a way that minimizes the loss function, which is key to helping the CNN gradually learn the correct pattern. The loss function is essential for calculating the gradient during backpropagation, and the gradient determines the direction and size of the weight updates, making it essential information for the CNN to learn. As such, the loss function plays an important role in allowing a CNN model to quantitatively evaluate the difference between its predictions and the true value and adjust the parameters of the model to reduce this error, ultimately improving its prediction performance.

- Binary cross entropy;
- Hinge loss;
- Squared hinge loss.

Binary cross entropy is a loss function used primarily for binary classification problems. It measures the difference between the model's prediction and the actual label. This loss function is based on a probability value to evaluate how well the model predicts the true label. The lower the value, the closer the model's prediction is to the true label.

Hinge loss is primarily used in classification algorithms such as SVM. This loss function trains the model by considering the margin at the classification boundary. Hinge loss penalizes misclassified samples by forcing the model to take a margin when making predictions. This loss function aims to maximize the margin.

Squared hinge loss is a variant of hinge loss that uses the square of the error to calculate the loss. This loss function is designed to have a stronger impact on the model's predictions by imposing a larger penalty than hinge loss. Like hinge loss, this loss function tries to maximize the margin, but the penalty is stronger, which helps the model converge faster. In general, it is better for learning more complex boundaries.

In our experiments, as shown in Table 12, binary cross entropy and squared hinge loss performed similarly, but we chose binary cross entropy because it is commonly used for binary classification in CNN6 models (hyperparameters: epoch 40, dense layer output 64, learning rate 0.001, dropout 0.3, kernel size 3×3).

Table 12. Comparison of loss functions.

Loss Function	Accuracy	Precision	Recall	F1 Score
Binary Cross entropy	0.9986	0.9986	0.9986	0.9986
Hinge Loss	0.9627	0.9606	0.9669	0.9624
Squared Hinge Loss	0.9986	0.9986	0.9986	0.9986

4.3.9. CNN6 Model Performance Evaluation

Through the above experiments, we were able to summarize the best initial parameters for the proposed CNN6 model, as shown in Table 13.

Table 13. Hyperparameters for the CNN6 model.

Hyperparameter Settings	
Input vector	32 × 32
Convolution Layer	2
Number of kernels (filters)	32, 64
Kernel size	3 × 3
Pooling layer	2 × 2
Activation function	Leaky ReLU, sigmoid
Epoch	50
Dropout	0.3
Dense layer	64
Optimizer	Adam
Learning rate	0.001

The experimental results of the CNN6 model with the above parameters showed that when the CNN6 model was run 10 times, the accuracy ranged from 0.9991 to 0.9997, with an average accuracy of 0.9995.

Since image classification is also possible with machine learning, we compared the performance of SVM, Random Forest, K-Nearest Neighbor (KNN), Logistic Regulation, Gradient Boosting, etc., among common machine learning models.

Decision Tree and Naive Bayes are also machine learning models, but due to the nature of images, they consider each pixel as an independent characteristic and segment data based on it. However, for images, it is important to learn the associations (e.g., shapes, patterns, boundaries) between neighboring pixels, so we excluded them from the comparison models here.

Among the machine learning models, as shown in Table 14 and Figures 19 and 20, SVM performed the best. Most of the other image classification methods using machine learning showed an accuracy of 0.95 or higher, but it was difficult to extract key features from 32 × 32 images, and the performance was much lower than the CNN6 model, which showed an accuracy of 0.9995. The machine learning model did not produce consistent results because the performance varied greatly from run to run.

Table 14. Performance comparison of the CNN6 model with other ML models.

Model	Accuracy	Precision	Recall	F1 Score
Gradient Boosting	0.9778	0.9764	0.9788	0.9775
Radom Forest	0.9737	0.9719	0.9764	0.9735
SVM	0.9820	0.9806	0.9836	0.9818
KNN	0.9580	0.9122	1.0000	0.9541
Logistic Regulation	0.9578	0.9550	0.9599	0.9570
CNN (epochs = 50)	0.9995	0.9994	0.9994	0.9994

On the other hand, the proposed CNN6 model showed stable and superior performance compared to machine learning, and it was confirmed that even in the case of low image resolution, deep learning using CNN can show good enough performance if the optimal hyperparameter combination is applied.

We compared the performance of the CNN6 model with that of major deep learning models, as shown in Table 15. We also compared the performance of the CNN-based models—VGG16, AlexNet of Alex Krizhevsky, DenseNet121 (Densely CNNs), MobileNet V2, and Residual Networks50 (ResNet50)—and Transformer-based models—Vision Transformer (ViT) and Shift Windows (SWIN) Transformer—as well as models that combine CNN and LSTM, as shown in Table 15.

Figure 19. Machine learning vs. CNN performance comparison.

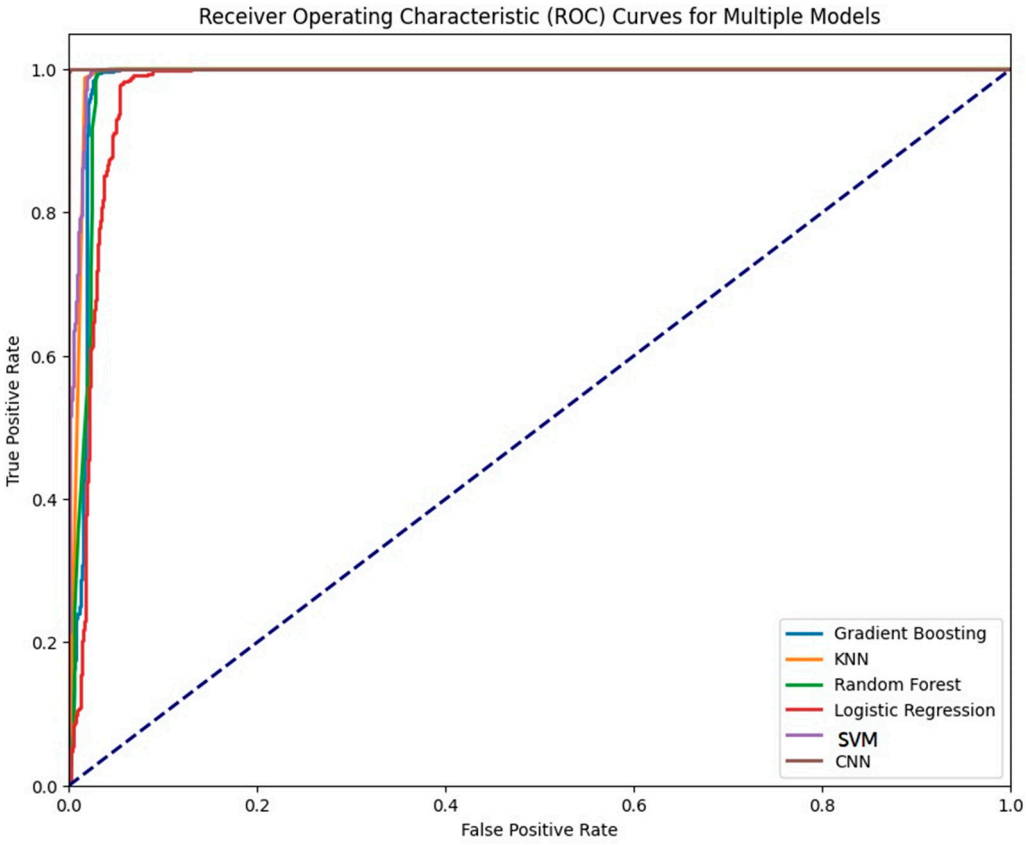

Figure 20. Machine learning vs. CNN performance comparison (ROC).

Most of the deep learning models were experimented with the keras API, and AlexNet was implemented directly. With the exception of MobileNet V2, the accuracy was between 0.996 and 0.999, and the performance of other deep learning models was similar to or slightly lower than that of the CNN6 model.

In theory, it should be better than a simple CNN model. However, the CNN6 model is a model that was applied with the best hyperparameters after repeated experiments and tuning. In contrast, most of the deep learning models based on a CNN are designed based on RGB images of 224 × 224 in size and modified for grayscale images of 32 × 32 in size.

Therefore, there was no significant difference in performance, and the Transformer-based model did not show better performance than the CNN6 model.

Finally, the model combining the CNN6 model and LSTM also showed low performance, contrary to expectations, and did not perform as well as the CNN6 model.

Table 15. Performance comparison of CNN6 with other deep learning models.

Model	Accuracy	Precision	Recall	F1 Score	Time (s)
CNN6	0.9995	0.9994	0.9994	0.9994	2653.55
CNN6 and LSTM	0.9991	0.9991	0.9991	0.9991	3294.83
DenseNet121	0.9995	0.9995	0.9995	0.9995	6786.47
VGG16	0.9992	0.9991	0.9993	0.9992	6226.52
AlexNet	0.9992	0.9992	0.9991	0.9992	9391.88
MobileNet V2	0.7348	0.7391	0.7176	0.7205	3110.65
ResNet50	0.9674	0.9652	0.9704	0.9670	3460.64
ViT	0.9781	0.9764	0.9796	0.9778	5761.84
SWIN Transformer	0.9893	0.9881	0.9904	0.9892	6662.54

In terms of the performance of execution time, the CNN6 model recorded the fastest execution time because it was designed with a compacted structure with well-adjusted hyperparameters, while the general-purpose deep learning models, which mostly use a large number of layers, took up to three times longer than the CNN6 model.

Therefore, we were able to confirm once again that the CNN6 model, which is properly tuned for DNS amplification DRDoS attacks, is considerably superior in terms of performance and execution times.

5. Conclusions

Machine learning and deep learning detection methods for DNS amplification DRDoS have already been studied in various ways. However, normal AI-based detection methods are not fit for this purpose. They have to extract DNS amplification DRDoS attack traffic characteristics and preprocess the dataset before training them for high performance. They also have to execute deep learning models, which takes a lot of time and consumes a lot of resources.

To overcome this limitation, the CNN6 model reduced the overhead of preprocessing by using an image processing method that converts the entire packet into a small image of 32×32 pixels. Furthermore, the proposed simple CNN model was continuously modified through micro-adjustment of the hyperparameter values. The CNN6 model delivered excellent classification accuracy for DNS amplification DRDoS attack traffic. The experimental results demonstrated that the proposed CNN6 model consistently outperformed general machine learning models, maintaining exceptional accuracy and performance. It also reduced data preprocessing, resulting in the fastest execution times. In the field, real-time DDoS response is more important than an accurate detection of 100%. Therefore, detection speed is more important than detection accuracy.

This study proposes an effective deep learning model that can detect and respond to DNS amplification DDoS attacks early in a real-time network environment using minimal time and resources.

In future research, we will explore the possibility of detecting and classifying various types of DDoS attacks by applying image-based learning techniques to other types of DDoS attacks. We will also improve the generalization of the detection model by using DDoS datasets from various sources.

In this study, we conducted experiments with the image minimized and fixed. In future studies, we will conduct experiments with varying image sizes to identify the

optimal image size for the deep learning model in terms of speed and performance. We will also conduct research on hybrid models based on simple CNNs. These will include models that combine image feature extraction techniques, such as Principal Component Analysis (PCA), Local Binary Pattern (LBP), Visualization tools like BinVis with CNNs, models that combine machine learning with CNNs, and models that combine CNN with other deep learning methods.

Author Contributions: Conceptualization, H.S. and J.J.; methodology, H.S. and O.K.; software, H.S. and J.L.; validation, J.J., K.C. and J.L.; formal analysis, O.K.; investigation, H.S. and K.C.; resources, J.L.; writing—original draft preparation, H.S. and J.J.; writing—review and editing, H.S. and D.S.; visualization, K.C.; supervision, D.S.; project administration, D.S. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by a National Research Foundation of Korea (NRF) grant funded by the Korean government (MSIT) (No. 2022R1F1A1074773).

Data Availability Statement: The data presented in this study are available on request from the corresponding author.

Conflicts of Interest: Author Jaeil Lee was employed by the company SmileGate. The remaining authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

References

1. Available online: https://www.isssource.com/huge-ddos-attack-a-new-approach/ (accessed on 9 November 2024).
2. Available online: http://techcrunch.com/2016/10/21/many-sites-including-twitter-and-spotify-suffering-outage/ (accessed on 9 November 2024).
3. Available online: https://www.eweek.com/security/github-hit-by-largest-ddos-attack-ever-recorded-at-1.35-tbps (accessed on 9 November 2024).
4. Available online: https://blog.cloudflare.com/ddos-threat-report-for-2024-q1/ (accessed on 9 November 2024).
5. Seo, I.H.; Lee, K.T.; Yu, J.; Kim, S. CNN based real-time DNS DDoS attack detection system. *KIPS Trans. Comput. Commun. Syst.* **2017**, *6*, 135–142. [CrossRef]
6. Gao, Y.; Feng, Y.; Sakurai, K. A Machine Learning Based Approach for Detecting DRDoS Attacks and Its Performance Evaluation. In Proceedings of the 2016 11th Asia Joint Conference on Information Security (AsiaJCIS), Fukuoka, Japan, 15 December 2016; pp. 80–86. [CrossRef]
7. Davis, R.E.; Xu, J.; Roy, K. Classifying Malware Traffic Using Images and Deep Convolutional Neural Network. *IEEE Access* **2024**, *12*, 58031–58038. [CrossRef]
8. Jeong, K. A Study of Data Preprocessing for Network Intrusion Detection based on Deep Learning. In Proceedings of the Korean Society of Computer Information Conference, Jeju, Republic of Korea, 16–18 July 2018; Available online: https://koreascience.kr/article/CFKO201831342440410.page (accessed on 9 November 2024).
9. Hattak, A.; Martinelli, F.; Mercaldo, F.; Santone, A. On the Adoption of Explainable Deep Learning for Image-Based Network Traffic Classification. In Proceedings of the 14th International Conference on Simulation and Modeling Methodologies, Technologies and Applications—Volume 1: SIMULTECH, Dijon, France, 10–12 July 2024; pp. 370–377. [CrossRef]
10. Yang, J. The Application of Deep Learning for Network Traffic Classification. *Highlights Sci. Eng. Technol.* **2023**, *39*, 979–984. [CrossRef]
11. Bendiab, G.; Shiaeles, S.; Alruban, A.; Kolokotronis, N. IoT malware network traffic classification using visual representation and deep learning. In Proceedings of the 6th IEEE Conference on Network Softwarization (NetSoft), Ghent, Belgium, 29 June–3 July 2020. [CrossRef]
12. Xiong, Y.; Dong, S.; Liu, R.; Shi, F.; Jing, X. IoT network traffic classification: A deep learning method with Fourier transform-assisted hyperparameter optimization. *Front. Phys. Sec. Soc. Phys.* **2023**, *11*, 1273862. [CrossRef]
13. Hyojong, K.; Kunhee, H.; Seungsoo, S. Response System for DRDoS Amplification Attacks. *J. Converg. Inf. Technol.* **2020**, *10*, 22–30. [CrossRef]
14. Conti, M.; Li, Q.Q.; Maragno, A.; Spolaor, R. The Dark Side(-Channel) of Mobile Devices: A Survey on Network Traffic Analysis. *Commun. Surv. Tuts.* **2018**, *20*, 2658–2713. [CrossRef]

15. Yeo, M.; Koo, Y.; Yoon, Y.; Hwang, T.; Ryu, J.; Song, J.; Park, C. Flow-based malware detection using convolutional neural network. In Proceedings of the 2018 International Conference on Information Networking (ICOIN), Chiang Mai, Thailand, 10–12 January 2018; pp. 910–913. [CrossRef]
16. Aydın, H.; Orman, Z.; Aydın, M.A. A long short-term memory (LSTM)-based distributed denial of service (DDoS) detection and defense system design in public cloud network environment. *Comput. Secur.* **2022**, *118*, 102725. [CrossRef]
17. Diaba, S.Y.; Elmusrati, M. Proposed algorithm for smart grid DDoS detection based on deep learning. *Neural Netw.* **2023**, *159*, 175–184. [CrossRef] [PubMed]
18. Gadze, J.D.; Bamfo-Asante, A.A.; Agyemang, J.O.; Nunoo-Mensah, H.; Opare, K.A.B. An investigation into the application of deep learning in the detection and mitigation of DDOS attack on SDN controllers. *Technologies* **2021**, *9*, 14. [CrossRef]
19. Aswad, F.M.; Ahmed, A.M.S.; Alhammadi, N.A.M.; Khalaf, B.A.; Mostafa, S.A. Deep learning in distributed denial-of-service attacks detection method for Internet of Things networks. *J. Intell. Syst.* **2023**, *32*, 20220155. [CrossRef]
20. Ahmed, T.U.; Hossain, M.S.; Alam, M.J.; Andersson, K. An integrated CNN-RNN framework to assess road crack. In Proceedings of the 2019 22nd International Conference on Computer and Information Technology (ICCIT), Dhaka, Bangladesh, 18–20 December 2019. [CrossRef]
21. Kim, T.; Pak, W. Deep Learning-Based Network Intrusion Detection Using Multiple Image Transformers. *Appl. Sci.* **2023**, *13*, 2754. [CrossRef]
22. He, Y.; Li, W. Image-based encrypted traffic classification with convolution neural networks. In Proceedings of the 2020 IEEE Fifth International Conference on Data Science in Cyberspace (DSC), Hong Kong, China, 27–30 July 2020. [CrossRef]
23. Kim, J.; Kim, J.; Kim, H.; Shim, M.; Choi, E. CNN-Based Network Intrusion Detection against Denial-of-Service Attacks. *Electronics* **2020**, *9*, 916. [CrossRef]
24. Wang, W.; Zhu, M.; Zeng, X.; Ye, X.; Sheng, Y. Malware traffic classification using convolutional neural network for representation learning. In Proceedings of the 2017 International Conference on Information Networking (ICOIN), Da Nang, Vietnam, 11–13 January 2017; pp. 712–717. [CrossRef]
25. Nataraj, L.; Karthikeyan, S.; Jacob, G.; Manjunath, B. S Malware images: Visualization and automatic classification. In Proceedings of the 8th International Symposium on Visualization for Cyber Security, Pittsburgh, PA, USA, 20 July 2011. [CrossRef]
26. Kumar, R.; Xiaosong, Z.; Khan, R.U.; Ahad, I.; Kumar, J. Malicious Code Detection based on Image Processing Using Deep Learning. In Proceedings of the 2018 International Conference on Computing and Artificial Intelligence (ICCAI '18), Association for Computing Machinery, New York, NY, USA, 12–14 March 2018; pp. 81–85. [CrossRef]
27. Lee, S.Y.; Moon, B.; Kim, J. Malware Classification Schemes Based on CNN Using Images and Metadata. In Proceedings of the Korea Information Processing Society Conference, Seoul, Republic of Korea, 14–15 May 2021. [CrossRef]
28. Haaijer, L.F. DDoS Packet Capture Collection. 2022. Available online: https://github.com/StopDDoS/packet-captures (accessed on 9 November 2024).