mathematics

MDPI

Article

Convolutional Neural Network-Based Cryptography Ransomware Detection for Low-End Embedded Processors

Hyunji Kim, Jaehoon Park, Hyeokdong Kwon, Kyoungbae Jang and Hwajeong Seo *

Division of IT Convergence Engineering, Hansung University, Seoul 02876, Korea; 1594012@hansung.ac.kr (H.K.); 20213201@hansung.ac.kr (J.P.); hyeok@hansung.ac.kr (H.K.); starj1234@hansung.ac.kr (K.J.)
* Correspondence: hwajeong@hansung.ac.kr; Tel.: +82-760-8033

Abstract: A crypto-ransomware has the process to encrypt victim's files. Afterward, the crypto-ransomware requests a ransom for the password of encrypted files to victims. In this paper, we present a novel approach to prevent crypto-ransomware by detecting block cipher algorithms for Internet of Things (IoT) platforms. We extract the sequence and frequency characteristics from the opcode of binary files for the 8-bit Alf and Vegard's RISC (AVR) processor microcontroller. In other words, the late fusion method is used to extract two features from one source data, learn through each network, and integrate them. We classify the crypto-ransomware virus or harmless software through the proposed method. The general software from AVR packages and block cipher implementations written in C language from lightweight block cipher library (i.e., Fair Evaluation of Lightweight Cryptographic Systems (`FELICS`)) are trained through the deep learning network and evaluated. The general software and block cipher algorithms are successfully classified by training functions in binary files. Furthermore, we detect binary codes that encrypt a file using block ciphers. The detection rate is evaluated in terms of F-measure, which is the harmonic mean of precision and recall. The proposed method not only achieved 97% detection success rate for crypto-ransomware but also achieved 80% success rate in classification for each lightweight cryptographic algorithm and benign firmware. In addition, the success rate in classification for Substitution-Permutation-Network (SPN) structure, Addition-Rotation-eXclusive-or structures (ARX) structure, and benign firmware is 95%.

Keywords: deep learning; cryptography; ransomware; internet of things

Citation: Kim, H.; Park, J.; Kwon, H.; Jang, K.; Seo, H. Convolutional Neural Network-Based Cryptography Ransomware Detection for Low-End Embedded Processors. *Mathematics* **2021**, *9*, 705. https://doi.org/10.3390/math9070705

Academic Editor: Eckhard Pfluegel

Received: 22 February 2021
Accepted: 22 March 2021
Published: 24 March 2021

1. Introduction

In 2017, users of Microsoft Windows were infected by Wannacry ransomware virus, since the file sharing protocol has the the vulnerability [1]. It is the biggest attack in the history of ransomware, with more than 200,000 computers affected.

There are two categories of ransomware, including cryptography type or locker type. Since the locker type locks the target machine, it can no longer be accessible by users [2]. However, the data can be copied to other devices, and the data can be recovered because the data is not fully encrypted.

On the other hand, files of devices are encrypted by the cryptography ransomware. Since cryptography algorithms used to encrypt the victim's files are designed to be secure in mathematical assumptions, it cannot be recovered without the valid secret key. The victim has to pay the ransom to the hacker to recover files. Then, the victim can recover original files using the secret key. Since most users are moving to digitization, the ransomware is a large threat to digital devices. In order to prevent ransomware attacks, many works devoted to detect the ransomware virus and recover damages.

The four most common crypto ransomware programs are analyzed in Reference [3]. All ransomware viruses that rely on the target system's available system tools are identified. The file can be recovered by shadow copies generated while the tool is running.

The ransomware detection is divided into the analysis of data traffic and the function call. In order to improve the performance, the machine learning is actively studied.

In Reference [4], the ransomware virus is identified by analyzing ransomware network behavior and packet selection. In Reference [5], the light and deep networks to detect the ransomware virus were evaluated. The dominance of application programming interfaces (APIs) is analyzed to characterize and differentiate ransomware. In Reference [6], a framework for multi-level big data mining is utilized. The ransomware is analyzed at different levels, such as the function call, dynamic link library (DLL), and machine code level, through supervised machine learning. Many research studies focused on the cryptography function because of the nature of crypto ransomware.

In Reference [7], the collected data through the dynamic binary analysis is utilized to characterize a specific aspect of cryptographic codes. In Reference [8], an approach is presented to automatically identify parameters and block cipher algorithms in the binary code, and it is based on static. In Reference [9], asymmetric key cryptographic (AKC) algorithms are targeted since the ransomware performs the public key algorithms to encrypt files. The encryption process performed by public-key cryptographic algorithms can be detected by monitoring integer multiplication instructions. However, the architecture of block cipher was paid little attention in previous works. In addition, there are not many works to defend against ransomware in the Internet of Things (IoT) environment.

In this paper, we propose a new approach to prevent ransomware viruses by classifying the cryptography process of block cipher on low-end embedded processors. Codes of target embedded processors are analyzed and classified as ransomware or innocent software. Using neural networks, we trained and evaluated the binary code of functions in the block cipher algorithm in the Fair Evaluation of Lightweight Cryptographic Systems (FELICS) and the benign program in the Alf and Vegard's RISC (AVR) processor (AVR) package. First, the binary files of each block cipher and benign firmware are pre-processed in two ways. One is the time series data representing the flow of opcode, and the other is the frequency of the opcode. That is, our neural network trains those features by fusing two features from one source data. Finally, a binary file of file encryption process or benign firmware is entered into the trained model and then classified. The proposed method successfully classified the encryption process and detected ransomware virus. In addition, we evaluated the result in terms of precision, the harmonic mean of recall, and F-measure. Following are our contributions.

1.1. Contribution

1.1.1. Detecting Crypto Ransomware Using Deep Learning for IoT

The potential ransomware is successfully detected by classifying the binary code. The binary file is transformed to two arrays representing sequence and frequency. Then, these two type of data are applied the neural network. After training, the trained model pruned and converted to tensorflow lite model to inference in IoT environments.

1.1.2. Experiments with Several Methods to Achieve High Performance

We evaluated the proposed method with several options to achieve high performance in the ransomware detection, such as using two features about sequence and frequency from binary file and only one feature about sequence are compared. In addition, both methods for extracting instructions and opcodes from binary code are compared and evaluated.

1.1.3. Detailed Analysis of the Instruction Set Used in Block Ciphers Implemented in Microcontrollers

Symmetric Key Cryptography (SKC) algorithms are classified into two categories, Substitution-Permutation-Network (SPN) or Addition-Rotation-eXclusive-or structures (ARX). This approach observes distinguished features between them. Based on this feature, the performance is significantly improved.

1.2. Extended Version of World Conference on Information Security Applications (WISA)'20

This paper is an extension of our previous work published in WISA'20 [10]. A method of detecting crypto-ransomware by converting the binary code of an encryption algorithm into an image was proposed in the previous work. To perform this method, first, we extracted the instructions or opcodes from the binary file generated by compiling in the AVR environment and converted it into an image. Converted images entered into the convolutional neural network (CNN) to classify these image data. Since the generated image contains features of a specific cryptographic operation, the encryption process can be detected by the CNN. Therefore, it is possible to classify ransomware and general firmware because the crypto-ransomware encrypts some files through an encryption algorithm. In addition, the classification by each cryptographic algorithm and by SPN and ARX structure is also possible. However, in the case of the previous work, the detection of the entire process of ransomware encrypting files was not conducted. In other words, an encryption process is detected, but it is not about the detection of an entire ransomware file. This work further extended to detect crypto-ransomware through binary files by neural networks.

The organization of this paper is as follows. The background of cryptography ransomware detection methods for embedded processors is provided in Section 2. Then, we propose and evaluate a new method to prevent cryptography ransomware in Section 3 and Section 4. In Section 5, we conclude this paper.

2. Related Works

2.1. Ransomware on IoT Environments

Due to the rapid development of the IoT, ransomware virus protection and security enhancement have been established as fundamental components for IoT-based services [11]. Many works have been carried out for a safe IoT environment.

In Reference [12], a method based on machine learning to detect ransomware is presented. They monitor the power consumption patterns of some processes, and then the malicious ransomware is detected from benign applications. In Reference [13], a method based on deep learning is presented to detect malware by using the opcode sequence of machine. Opcodes are converted to a vector space. Afterward, they used a deep Eigenspace learning. Finally, malware and harmless application are classified. In Reference [14], the proposed method utilized the behavior. Afterward, Transmission Control Protocol/Internet Protocol (TCP/IP) header is extracted and the ransomware attacks are detected by command and control (C&C) server blacklisting. In Reference [15], the sequence of instructions is converted to a grayscale image. Then, multi-classes are separated using the statistical method along with the reduction of dimension.

However, these previous approaches focused on high-end IoT platforms. To collect the data in distance, low-end microcontrollers are used in IoT services. For the success of IoT-based services, the ransomware detection mechanism should be considered for low-end microcontrollers. In this work, a novel mechanism to detect ransomware for low-end environment is presented. To detect malicious ransomware, we use the sequence and frequency of binary code as a feature of the encryption process. Afterward, the data representing these features are classified through the neural network.

2.2. Ransomware Detection Methods Based on Cryptographic Function Call

To encrypt victim's files, ransomware use the cryptographic function. Therefore, detecting the cryptographic function should be highly considered to classify the malicious software. We compared the cryptographic function call-based ransomware detection methods (see Table 1). In Reference [7], they detected symmetric key and public key cryptography by the features of encryption functions. This approach uses a heuristics method that is based on the target architecture. In Reference [8], the data flow graph extracted from the binary file was utilized. The cryptographic function call is identified by using the sub-graph isomorphism. In Reference [9], the instructions of multiplication heavily called in Rivest–Shamir–Adleman (RSA) algorithm are monitored, then the public

key cryptographic algorithms are detected. However, the deep learning-based ransomware detection method is not explored in previous works. Recently, Reference [16] showed that malware detection can be improved through deep learning algorithms. However, this method does not target ransomware, and the high-end desktop is target platform. In this paper, a cryptography ransomware detection mechanism targeting microcontroller is presented.

Table 1. Comparison of detection methods based on cryptographic function call. SKC and AKC represent Symmetric Key Cryptography and Asymmetric Key Cryptography, respectively.

Category	[7]	[8]	[9]	This Work
Cryptography	SKC and AKC	SKC	AKC	SKC
Apporach	Dynamic	Static	Dynamic	Static
Algorithm	Heuristics	Data graph flow	System monitor	Deep learning
Architecture	Desktop	Desktop	Desktop	Embedded Processor

2.3. Symmetric Key Cryptography on Low-End Embedded Processors

In 2015, the cryptography benchmarking framework (FELICS) was presented by the University of Luxembourg [17]. The cryptographic engineers around the world submitted a number of block cipher implementations on embedded processors to FELICS. We used the block cipher implementations of FELICS in this work. The ATmega128, an 8-bit AVR widely used in low-end IoT environments, is a target low-end microcontroller. The microcontroller is the modified Harvard architecture-based 8-bit single-chip. It has 8-bit unit registers and instructions. Block ciphers can be classified into two categories. There are Addition, Rotation, and bitwise eXclusive-or (ARX) and Substitution-Permutation-Network (SPN). Each architecture can be characterized by its structure and the operations performed. In this paper, we classified the binary code by distinguishing features of both architectures.

3. Proposed Method

The proposed method detects ransomware with binary code for low-end IoT devices. In general, the sensor network has a tree structure [18]. Low-end IoT used as leaf nodes collect sensor data. And the leaf nodes of the tree are managed by the root node (i.e., base station). To ensure better service than before, the firmware of leaf nodes is regularly updated by the base station. When the packets between base stations (or firmware servers) are intercepted by a hacker, and the crypto ransomware is inserted into the firmware, the base station or leaf node has to detect the ransomware virus. In our scenario, the crypto-ransomware is detected by classifying the firmware's binary file using a neural network. The proposed approach classifies ransomware and harmless firmware depending on whether the encryption process is running or not. This approach is able to self-defense for the middle-end IoT with 64-bit Advanced RISC Machines (ARM) processors, such as raspberry pi. The convolutional and fully connected neural network can be performed on the device by TensorFlow Lite model.

The proposed method is designed as shown in Figure 1. We targeted block cipher algorithms. After compiling a source code of block cipher algorithms, we can get the binary data of source code. There are many functions to perform encryption, decryption, and key schedule operations. We extract some functions to perform the encryption. Afterward, opcodes for these functions are extracted. Since opcodes are a string type, we convert hexadecimal opcodes to decimal numbers to encode. Converted decimal numbers become an array for each function. Since the converted array has an opcode for each element. we change the converted array to express the frequency of opcode. Two arrays are inserted to the neural network for the training. After the training, they are classified each cryptographic function or general firmware. Then, we can detect the ransomware. The neural network we used is consist of two models. One model trains the opcode sequence and the other

trains frequency of each opcode in a binary file. Then, two models are concatenated and entered to a fully connected layer.

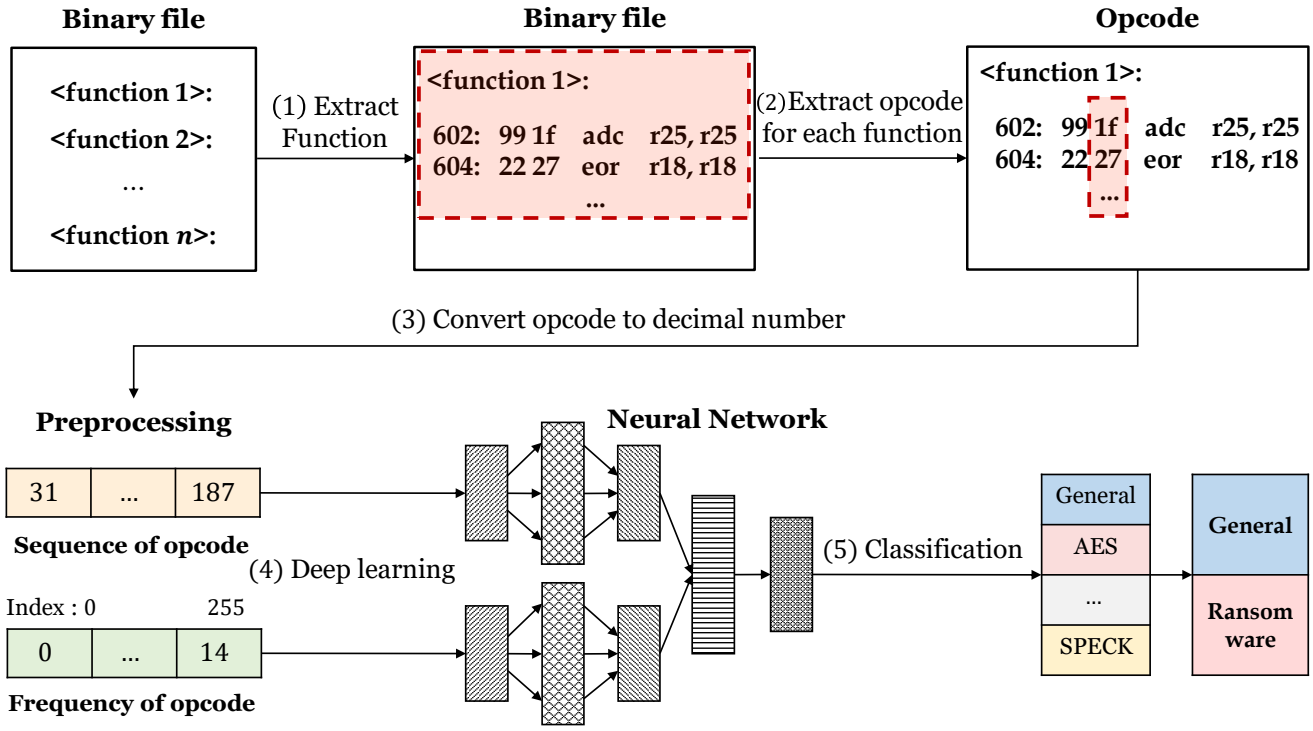

Figure 1. System configuration for proposed method.

3.1. Data Generation Based on Binary Code

After compiling on source codes, we can get binary files. Since the encryption process is a distinguished feature of ransomware, we extract functions to perform the encryption in the binary file. There are several functions in the cryptographic algorithm. We set operations used in each algorithm as a feature because operations used in other algorithms are different. The instruction pattern for encryption operations is trained, and we perform the classification of the binary file for the algorithm. The section that transfers from one function to another is not trained because it is not a sequence for a specific operation. We made the extraction tool. Since extracted opcode patterns are performed in order, they are time series data for each function. As shown in Figure 2, extracted opcodes become 1-dimensional (1-D) array. The max length of the sequence array is set to 1000. When we calculated the length of function, the minimum is 14, and the maximum is 13,859. The maximum value is the case that all operations are implemented without using functions. The number of such cases is only 2, and most binary files have functions. As a result of finding the percentile, if the length of a function is 878, among them, 90% of the number of data is covered. We set the max length to 1000, and only two cases were excluded. If the length of opcode sequence for each function is less than 1000 or more than 1000, they are padded with 0 or cut up to 1000. Finally, each opcode sequence is set to the name of its source function.

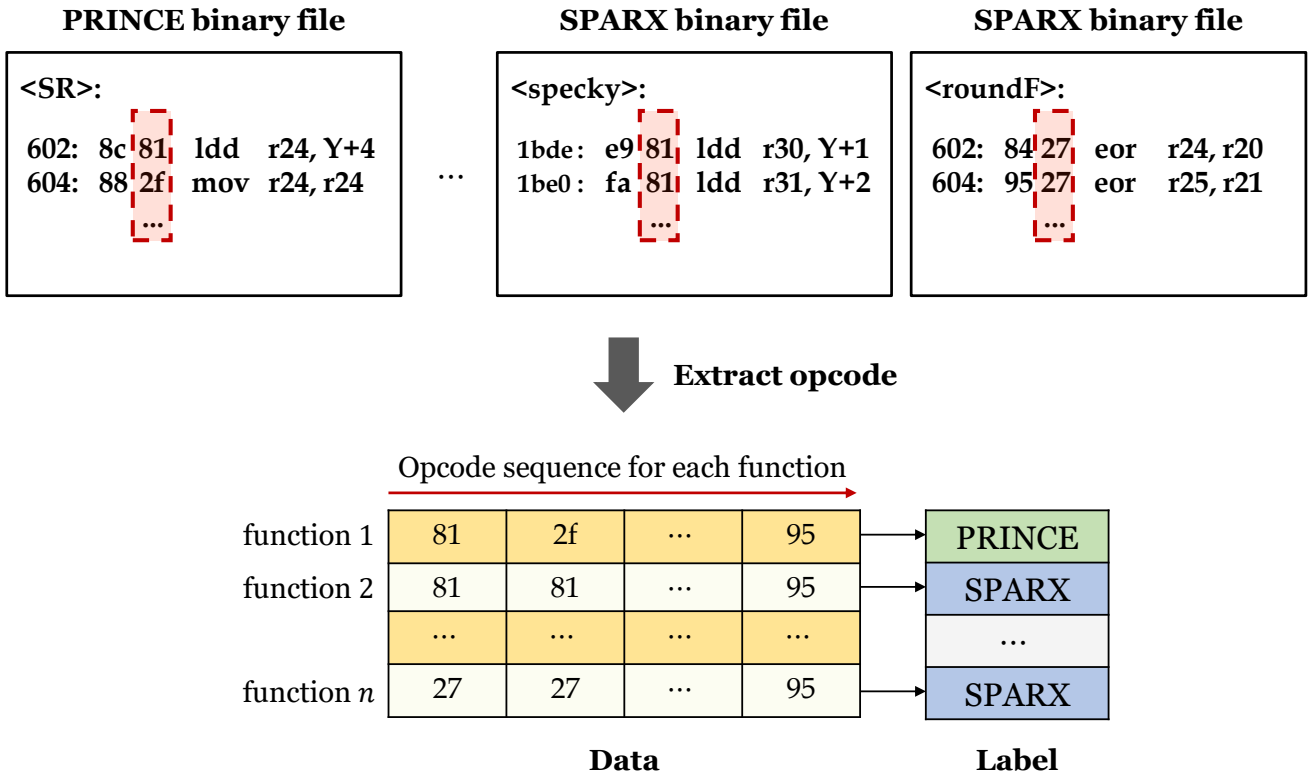

Figure 2. Opcodes sequence extraction for each function.

Extracted opcode sequences are formatted in a string type, which is hexadecimal and 1-dimensional array. So, we need to convert input data type and format as shown in Figure 3. The pre-processing phase is divided into two ways. First, the elements of the opcode array must be transformed to be each row in order to be input into the convolution 1D layer. Afterward, it becomes time series data having one feature. In addition, each data is extracted from 1 source data, so both sequence and frequency have the same label.

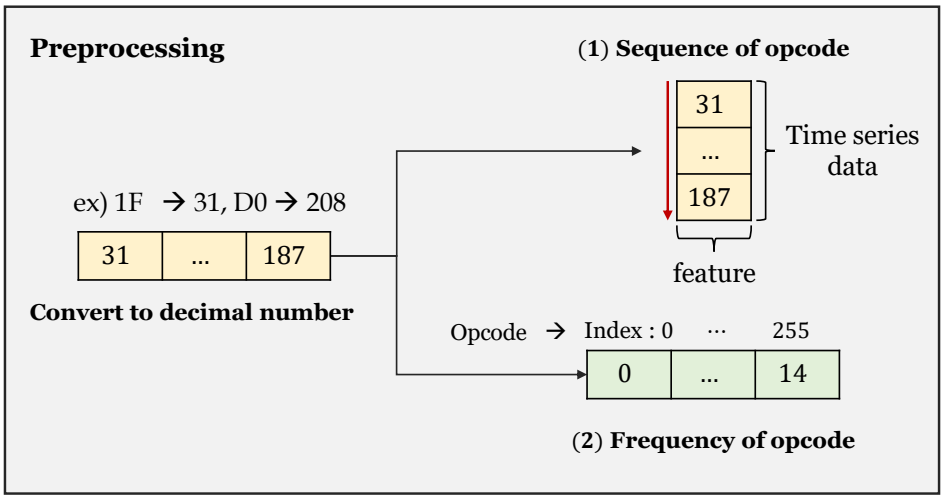

Figure 3. Data pre-processing.

3.2. Deep Learning Phase

We used the concatenated neural network to train two types of pre-processed data. In other words, the two inputs and one output are required. Each input is entered to the neural network and trained, properly. Then, the feature map of each model is concatenated.

Afterward, we can get the result of classification with feature maps. This architecture is a late fusion, where errors are propagated for each model. This considers the responsibility and the plausibility of each input. If one of the inputs goes the wrong result, the late fusion model can solve this problem because the models are independent of each other.

3.2.1. Training Phase

In the training phase, the pre-processed data about the function of each algorithm is entered to the neural network as shown in Figure 4. In other words, the sequence of the opcodes and the frequency of the opcodes used in function, are trained. That is, it is to train by extracting two kinds of features from one source data. The each data go through all the layers in each model, then they are concatenated and classified 11 cryptographic algorithms and 1 general firmware. A CNN model consisting mainly of convolution 1D layers is used to train opcode sequences, which are time-series data. A fully connected model composed mainly of dense layers is used to train frequency of opcodes. In fact, the Recurrent Neural Network (RNN) is good for training the sequential data. However, the proposed system detects the ransomware on IoT devices. For this reason, we need to convert our model to the tensorflow lite model to inference on edge tensor processing unit (TPU). However, the Google Coral Edge TPU does not support the RNN layer yet. In addition, the sequence is not long enough because we train the opcode sequence for each function, not all binary files. For this reason, we selected the convolution 1D layer. Detailed hyper parameter values of neural network are given in Table 2. The convolution 1D layer has filters, and filters move at specific intervals by the value of stride. This layer is not to train the spatial feature, such as convolution 2D layer. The filter of convolution 1D layer moves from the start to the end of the sequence as shown Figure 5. Therefore, training the sequence data is possible. We set the filter size to 16 and 4 for each convolution layer through the hyper parameter optimization. The filter can extract (or detect) the feature from the space covered by filter. If the filter size is bigger, features can be extracted from the bigger space. However, this makes features obscure because the value of the space covered by the filter is multiplied with the filter and then all values are added to form one feature. The average length of opcode sequence in one operation, such as multiplication for one index, is about from 30 to 40. If we set the filter size to 30 or 40, one operation becomes one value. Then, it may obscure the feature of that operation. The filter size is selected. The stride of convolution layer is 1; then, the filter can operate densely. The proposed system is a multi-class classification, and then we use the categorical cross-entropy loss function and softmax activation function for output layer. The range of data is 0 to 255 because the data is 1-byte opcodes. The max length value is set to 1000 aforementioned.

Table 2. The hyperparameters for training.

Hyperparameters	Descriptions	Hyperparameters	Descriptions
Number of label	12	Loss	Categorical crossentropy
Range of data	0∼255 integer (1-byte opcode)	Activation	ReLu (hidden), Softmax (output)
Sequence max length	1000	Optimizer	Adam (lr = 0.0002)
Conv1D filters	16, 4	Epochs	50
Conv1D strides	1	Batch size	8

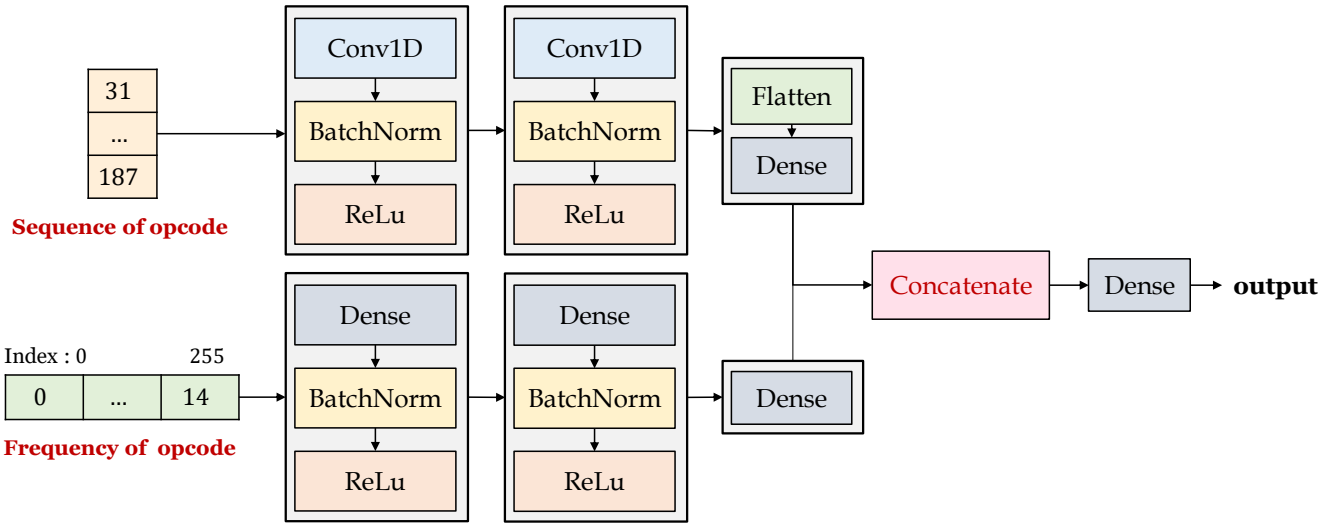

Figure 4. Neural network architecture for proposed method.

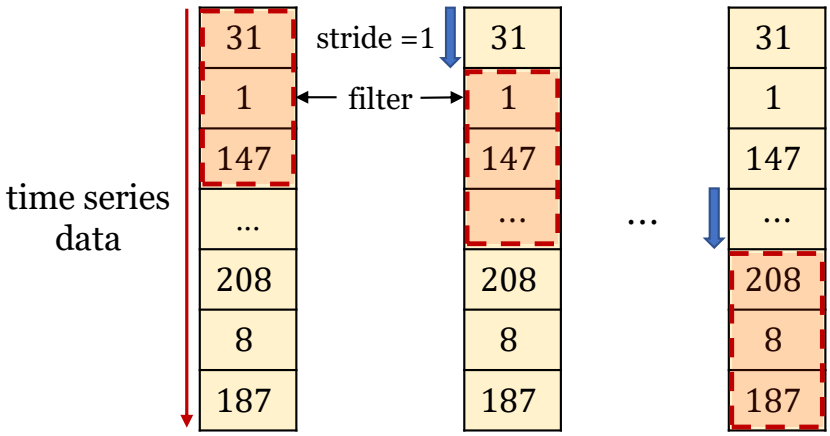

Figure 5. Convolution 1D layer for time series data training.

3.2.2. Ransomware Detection Phase

Figure 6 shows the detection phase. In the detection phase, all functions of a binary files are entered to the neural network. However, the input shape must be the same with the input shape of training data because the inference is performed through the trained model. The test data is truncated to a length of 1000. Except the process, the pre-process step for detection is identical. The truncated data is entered to the trained neural network. Then, the input data is classified as one of 12 labels, such as general firmware, Advanced Encryption Standard (AES), and PRESENT.

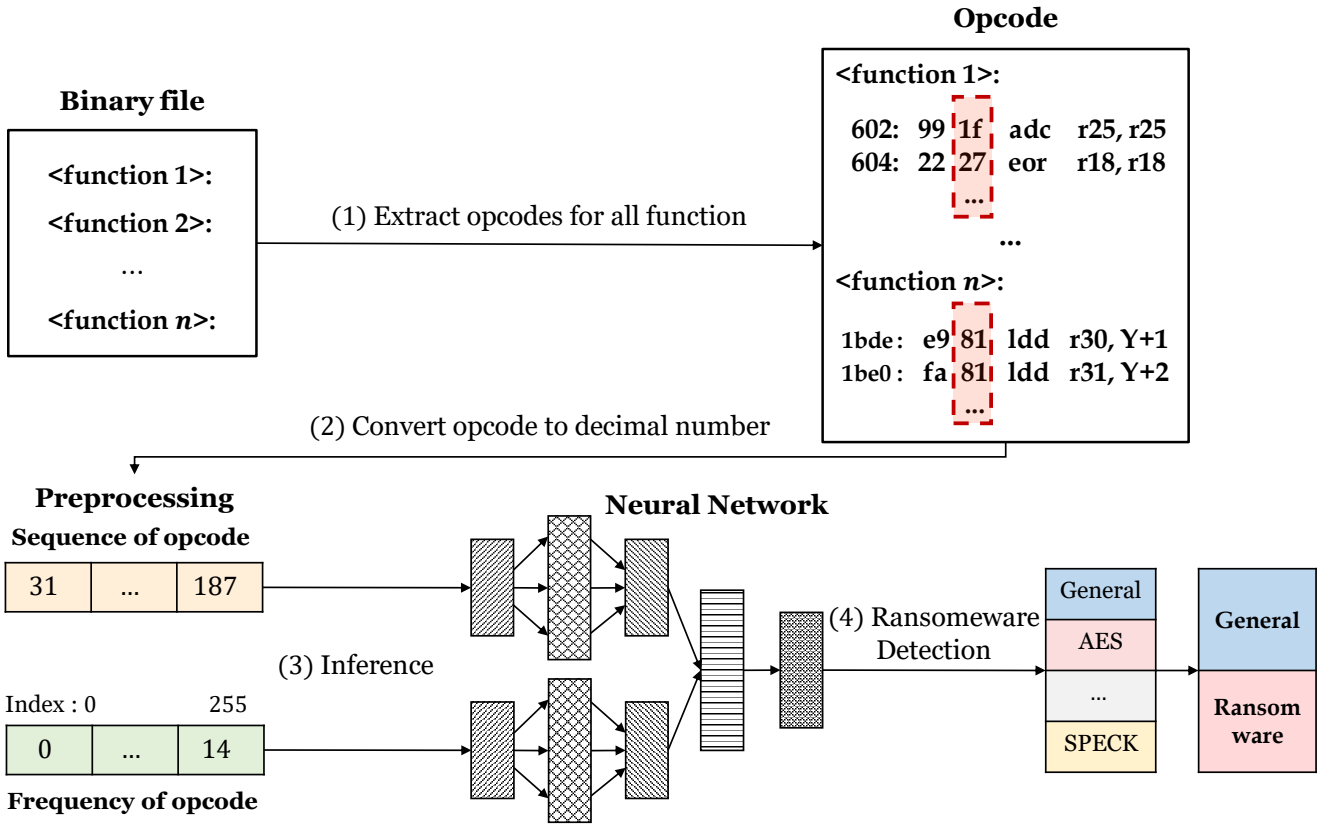

Figure 6. Ransomware detection phase.

3.2.3. Pruning for Ransomware Detection on the IoT Devices

The model of the proposed system is pruned for the lightweight model to detect the ransomware on IoT devices. Therefore, we apply the weight pruning for the trained model. In the pruning process, small weights not exceeding the threshold are removed. However, the pruned model has to achieve a similar level of performance to the original model. Since unimportant weights are removed from original weights to the extent, the performance degradation does not happen. The pruned model is converted to the tflite model to inference on edge devices. The pruned tflite model is deployed to IoT devices, and then we can detect the ransomware on IoT environments.

4. Evaluation

In this experiment, we used a cloud-based service, Google Colaboratory. This supports Intel Xeon CPU (13 GB RAM), Nvidia GPU (12 GB RAM), and Ubuntu 18.04.5 LTS. Python 3.6.9, TensorFlow 2.4.1, and Keras 2.4.0 were utilized for the programming environment.

Since the encryption is performed unlike benign firmware when the crypto ransomware runs, the encryption process is set as a characteristic of ransomware. Table 3 shows the detailed dataset of general firmware and symmetric key cryptography (e.g., substitution-permutation-network and addition-rotation-exclusive-or architectures) in IoT environments. Among the FELICS implementations, we used cryptographic modules written in the C language. General programs, including Radio-Frequency Identification (RFID), WiFi, xBee, and Bluetooth, are used as general firmware. In addition, values in parentheses are the number of data, and the training set, validation set, and test set are divided into approximately 7:2:1. In the detection phase, only binary files of block ciphers belonging to the test set are used. However, they are not divided by function, and their entire binary files are used.

Table 3. Detailed programs for the dataset.

Architecture	Examples
Substitution-Permutation-Network (88)	RECTANGLE(5), PRIDE(25), PRINCE(33), PRESENT(10), and AES(15)
Addition-Rotation-eXclusive-or (191)	SPECK(37), RC5(12), LEA(9), SIMON(19), HIGHT(58), and SPARX(56)
Benign (66)	XBee, GPS, WiFi, Bluetooth, and RFID

The experiment is conducted in three ways. The first experiment is conducted about opcodes. This way will construct a dataset with the proposed system and then detect the ransomware. The second experiment is carried out in the same way. However, the dataset consists of instructions containing operand, and opcode, not opcode. The third experiment is done only with opcode sequences, without considering the frequency characteristics. The last one is an experiment on the pruned model. Because an unbalanced dataset is used in this experiment, we use an F-measure. It is the harmonic mean of precision and recall rather than accuracy. There are micro and macro averaged F-measures. The micro approach takes into account the data belonging to each class. The macro approach takes into account the all classes with the same weight. The dataset used for these experiments has a different number of data for each class. Therefore, we evaluated through micro F-measure for each experiment. In addition, all results are the average value of 10 experiments. In addition, the result table consists of validation F-measure, test F-measure, and detection F-measure. Since validation and test results are about the training phase, these are results performed with each function in the binary file. Since the detection is about the detection phase, the detection is done with the entire binary file, not a function.

4.1. Instruction-Based vs. Opcode-Based

This experiment is to compare the performance of the proposed method (opcode-based) and instruction-based (opcode and operand). In case of instruction-based, the pre-process method is same, but operand and opcode are inputted into the neural network as feature values for one time series data. The sequence data of opcode-based method has 1 feature (i.e., opcode). However, the sequence data of instruction-based method has 2 features (operand and opcode).That is, if the proposed method has one column for n time-series data, the instruction-based method has two columns for n time series data. If the same operation is performed, it has same opcodes. However, operands are different in high possibility.

The experiment results of training and detecting on opcode-based and instruction-based methods are in Tables 4 and 5. These values mean the success rate. In case of classifying 11 cryptographic algorithms and benign firmware into 12 labels, the opcode-based method achieved 4%, 10%, and 10% higher detection performance in validation, testing, and detection, respectively, than the instruction-based method. For this reason, we proposed the opcode-based method.

Table 4. Evaluation of opcode-based model.

Category	Training		Detection
	Validation F-Measure	Test F-Measure	F-Measure
Each algorithm vs. General firmware	0.93	0.85	0.80
SPN vs. ARX vs. General firmware	0.99	0.98	0.97
Ransomware vs. General firmware	0.99	0.98	0.97

In the opcode-based method, most of the misclassifications were for (SPECK and LEA) and (SPECK and SPARX) in experiment for each algorithm. In the instruction-based method, most of the misclassifications were for (SIMON and SPECK), (SPARX and SPECK), and (LEA and SPECK) in the same experiment.

Table 5. Evaluation of instruction-based model.

Category	Training		Detection
	Validation F-Measure	Test F-Measure	F-Measure
Each algorithm vs. General firmware	0.89	0.75	0.70
SPN vs. ARX vs. General firmware	0.95	0.93	0.95
Ransomware vs. General firmware	0.99	0.98	0.97

In SPN versus ARX versus General firmware, the opcode-based method misclassified SPECK and general firmware in validation and test. In addition, in detection, the general firmware was incorrectly detected as SPARX. That is, there were no cases of misclassifying SPN and ARX structures. However, the instruction-based method misclassified between RECTANGLE, PRIDE, HIGHT, and general firmware in validation and test. In addition, in the detection, there were misclassification between RECTANGLE, SPARX, SPECK, and generic firmware. SPN and ARX structures are sometimes misclassified. However, both opcode-based and instruction-based methods show the same performance for the ransomware detection. Results of detecting ransomware is the same percentage. However, as shown in Figure 7, the instruction-based method has about 2 times higher loss value than the opcode-based method. Due to the nature of the training process that trains to minimize the loss, a model with a smaller loss value can classify in the accurate manner. Therefore, the opcode-based method is a better way to detect ransomware.

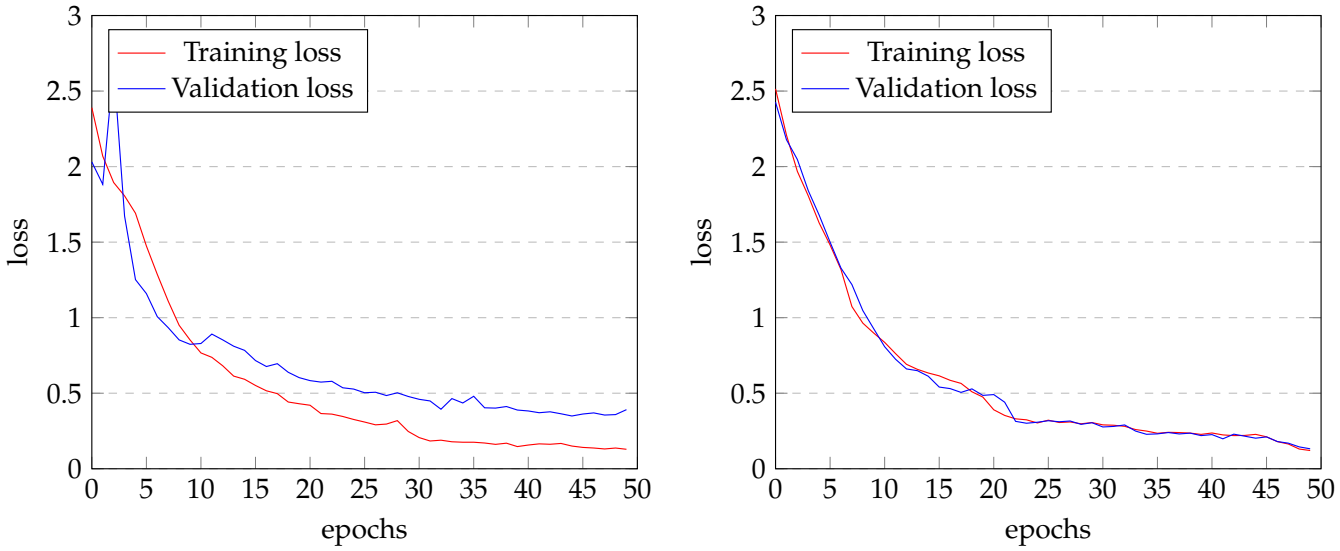

Figure 7. Training and validation losses for instruction-based (**left**) and opcode-based (**right**) models.

Figure 8 is the visualization of opcodes sequence for each function. First, functions that perform the same operation show almost similar patterns, and different functions have different patterns. We can see that rotation functions of SIMON and SPARX have a very similar sequence. In addition, the same pattern is repeated in the function. Figure 9 shows the frequency of opcodes for each algorithm. These images can be represented in an array with 256 columns. Therefore, each index represents an opcode, and elements of this array represent the number of times the opcode is used. The opcode used is represented by the vertical line in these Figures. Therefore, it is visually revealed that the distribution of opcodes is different for each structure through the location and contrast of the lines. In the gray scale, 0 means black, and 255 means white. The frequency was normalized to be within the range. The line of color closer to white is the more used opcode. The comparison is made between Figure 9 and Table 6. The most commonly used operations, such as `LD`, `ST`, and `MOVW`, are 129, 131, and 1, respectively. The middle of the images is expressed close

to white, and there is a common line in the leftmost part. The cryptographic algorithms which have ARX (Addition-Rotation-Xor) structure frequently performed arithmetic and logical operations, such as `ADIW, XOR, SBCI,` and `ADC`. The third most frequently used `ADIW` is 150. `XOR` and `ADC` are 30 and 40, respectively. There are lines close to the white color in the left part, which represent those operations. The cryptographic algorithms of SPN structure have operations to access the memory in the part of S-box operation. It is common and frequently performed. Similar operation to ARX was used, but the part corresponding to `XOR` or `ADC` was expressed in gray compared to ARX. In the general firmware, there are many branch instructions, and instructions that access I/O registers, which are rarely found in cryptographic code, such as `RJMP`, `BRNE`, and `OUT`. The operation like `NOP` is not used in cryptographic algorithms. Since operations of the collected general firmware are different, it is difficult to have a common pattern like encryption algorithms. There are no particular emphasized parts. Compared to this, cryptographic algorithms with the same architecture tend to share similar operation patterns. In addition, certain patterns are repeated because block cipher algorithms repeat rounds.

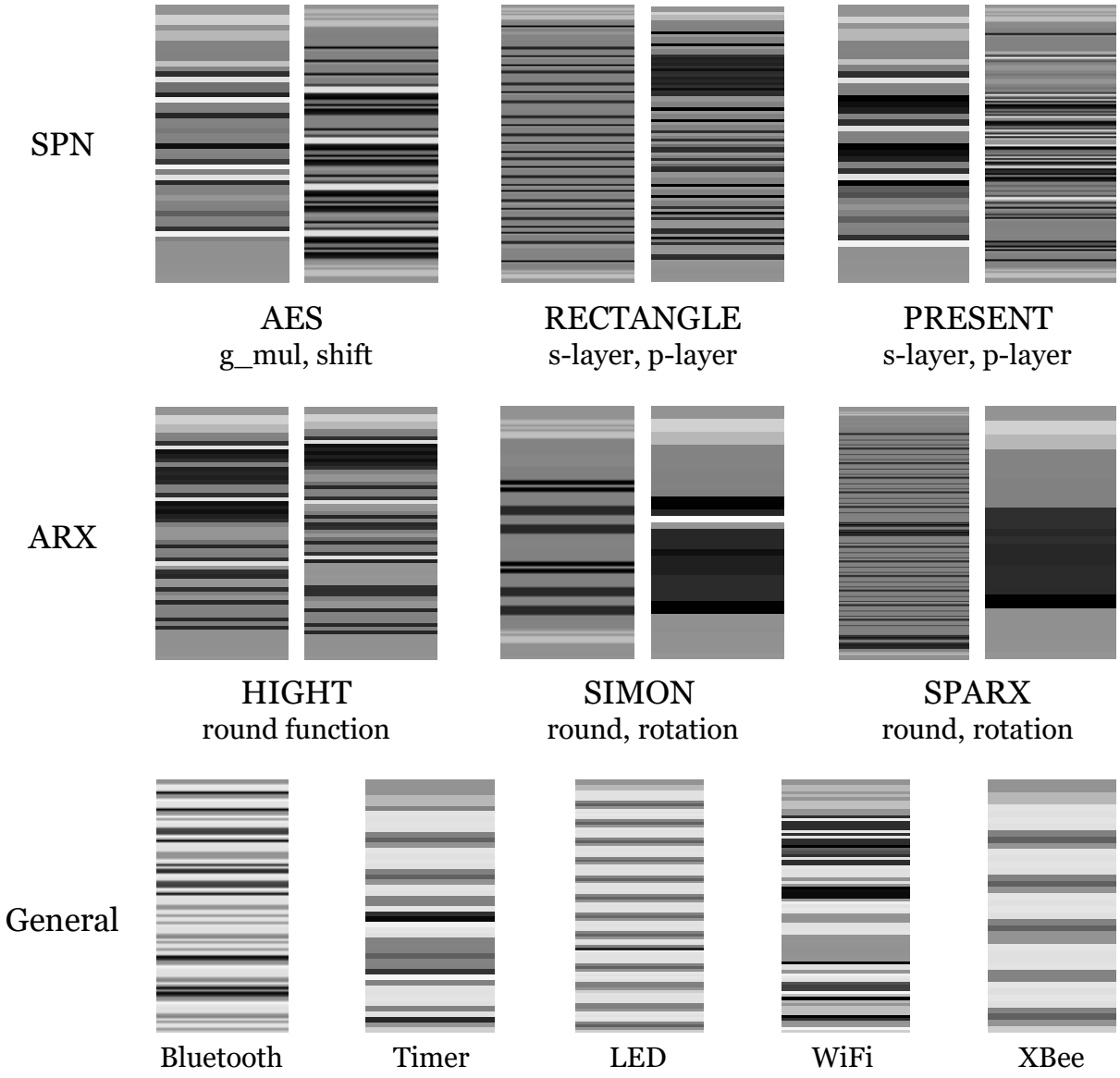

Figure 8. Visualization of binary images for each function.

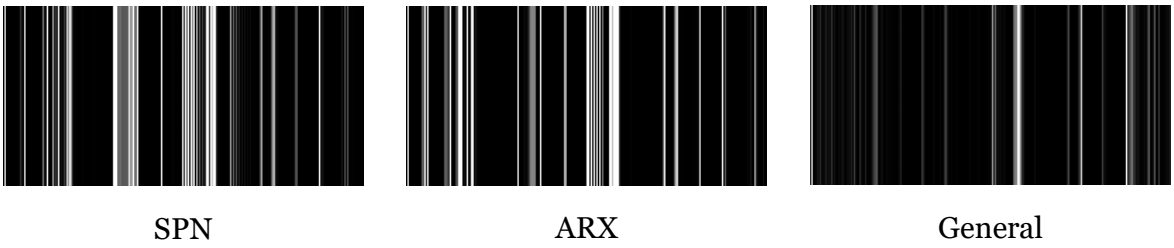

Figure 9. Visualization of opcode frequency for each algorithm.

Table 6. Comparison of each architecture depending on instruction frequency.

Architecture	Sorted by Frequency							
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
SPN	LD	ST	MOVW	XOR	ADIW	ANDI	MOV	STD
ARX	LD	MOVW	ADIW	STD	XOR	SBCI	ADC	LDD
General	LD	MOVW	RJMP	BRNE	BREQ	NOP	OUT	SBCI

4.2. Frequency and Opcode Sequence Versus Sequence-Only

In this experiment, we evaluated the model reflecting frequency and opcode (proposed model) and the model reflecting only the sequence. We applied the late fusion in proposed model. As mentioned in Section 3.2, the late fusion model can cope with a problem that the one of inputs is wrong, and it has independent error propagation. The post-concatenation process focuses on learned strengths (features) for each input. Therefore, better performance is achieved in general.

Table 7 shows the result of experiment with the model reflecting only the sequence. Compared to Table 4, in ransomware versus general firmware, the classification success rate decreased by 5% and 4% for the test data and data for ransomware detection, respectively. For validation and test data, there is no misclassification between SPN and ARX. In the detection phase, the classification success rate decreased by 10% between two cryptographic algorithms. Besides, in the case of classification by each algorithm, the performance was significantly degraded. The loss of the sequence-only model is shown in Figure 10. The loss value is sufficiently decreased, but there is slight gap between training loss and validation loss. In the case of this model, the detection success rate tends to be decreased for the full binary file in the detection.

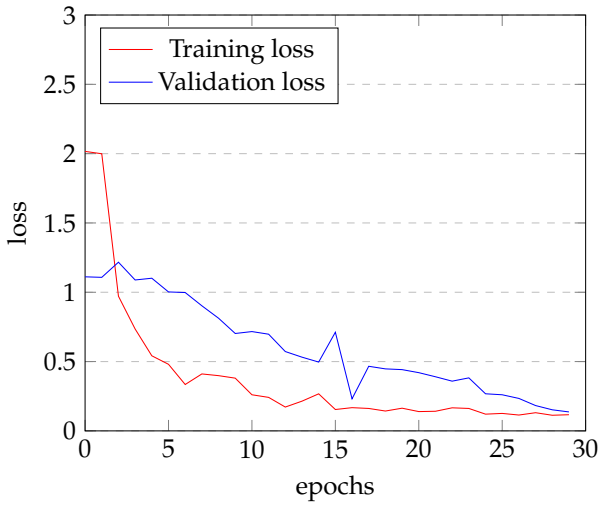

Figure 10. Training and validation losses for sequence-only model.

Table 7. Evaluation of sequence-only model.

Category	Training		Detection
	Validation F-Measure	Test F-Measure	F-Measure
Each algorithm vs. General firmware	0.86	0.76	0.67
SPN vs. ARX vs. General firmware	0.99	0.93	0.87
Ransomware vs. General firmware	0.99	0.93	0.93

4.3. Pruned Model for Detection on IoT Devices

We carried out the pruning to reduce the file size of model. The pruning is performed with the trained model. The loss value decreased rapidly as shown in Figure 11, and 20 epochs were enough. Table 8 shows the evaluation of the pruned model. The classification by algorithm and structure slightly decreased, but the performance was maintained for the classification of ransomware and general firmware. However, the file size is reduced only 7% because performance should not be influenced. Table 9 shows the file size and the F-measure of two models. The F-measure is calculated in detection of 'Ransomware versus General firmware' case.

Table 8. Evaluation of the pruned model.

Category	Training		Detection
	Validation F-Measure	Test F-Measure	F-Measure
Each algorithm vs. General firmware	0.93	0.88	0.83
SPN vs. ARX vs. General firmware	0.99	0.95	0.94
Ransomware vs. General firmware	0.99	0.95	0.97

Table 9. Evaluation of the lightweight model.

Model	File Size	F-Measure of Detection
Original tflite model	1.74 MB	0.97
Pruned tflite model	1.63 MB	0.97

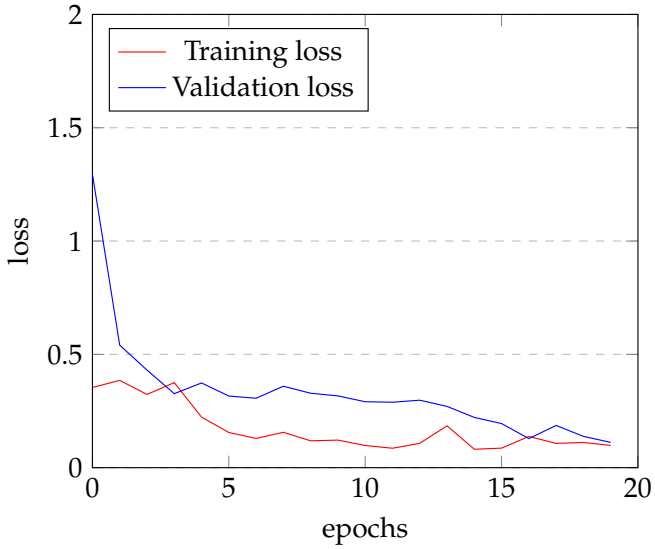

Figure 11. Training and validation losses for the pruned model.

4.4. Comparison with Other Methods

Table 10 shows the comparison with other methods. There are implementations of cryptographic algorithms, such as the OpenSSL and Cryptopp library. Since this work is an AVR environment, the implementation written in C in `FELICS` was used. Crypto-ransomware mainly consists of AES and RSA algorithms. Therefore, these two algorithms should be detected. In Reference [7], there are three approaches: chains, mnemonic-const, and verifier. Among them, the verifier identification method checks the existence and parameters of the symmetric encryption process, and it has the best performance. The verifier method achieved a detection success rate of 0.946 in AES. However, it was not possible to detect Message-Digest (MD5) algorithm and RSA, and if the chains method is used, both AES and RSA can be detected. In Reference [19], another factor that enables identification of the existence of the encryption process, the key scheduling process, can also be detected. It can detect AES, Data Encryption Standard (DES), RC4, MD5, and RSA with simple mechanisms. Proposed methods can cover a wider range of cryptographic algorithms. It is possible to detect lightweight block ciphers for low-power embedded processor environments. Therefore, it can be used to detect the ransomware for IoT environments.

Table 10. Comparison with other methods.

	Gröbert et al. [7]	**Caballero et al. [19]**	**This Work**
Algorithm	Heuristic	Field semantics inference	Deep learning
Implementation	OpenSSL, Cryptopp, Beecrypt	OpenSSL, Cryptopp, Beecrypt	FELICS
Description	AES, DES, RC4, and RSA	AES, DES, RC4, MD5, and RSA	See Table 3

5. Conclusions

In this paper, new methods to identify the potential malware by classifying the block cipher modules and benign firmware for embedded processors is presented. Opcodes in binary file are converted to decimal, then the converted data is pre-processed into 2 types of data which represent features of the functions. One is the sequence of opcodes, and the other one is the frequency of opcodes. Then, the neural network is applied to the deep learning. In order to classify operations performed by each algorithm, we divided them into function units and used them as features of each algorithm. In addition, the block cipher classification has 2 classes by utilizing unique features of block ciphers. Types of opcodes and frequency of opcodes mainly used in SPN and ARX are different. Through an experiment, it was confirmed that the frequency is a feature that affects performance. These approaches significantly improved the classification and the detection performance.

Furthermore, we suggested the proper ransomware detection framework by considering the resource-constrained environment. RNN layers that are not supported by deep learning accelerators for inference in edge devices were not used. Therefore, a convolution 1D layer that can be used for learning time series data and has less weight to be trained was used. In addition, pruning was performed to reduce the weight of the model. The file size was reduced by 7%, and the F-measure achieved 97% as before.

As a future work, we will explore classification method for the other cryptography modules, including public key cryptography and hash function.

Author Contributions: Investigation, H.K. (Hyeokdong Kwon) and J.P.; Software, H.K. (Hyunji Kim), K.J., and H.S.; Writing-original draft, H.K. (Hyunji Kim) and H.S.; Writing-review and editing, H.K. (Hyunji Kim), J.P., H.K. (Hyeokdong Kwon), K.J., and H.S. All authors have read and agreed to the published version of the manuscript.

Funding: This work was partly supported by Institute for Information & communications Technology Promotion(IITP) grant funded by the Korea government(MSIT) (No.2018-0-00264, Research on Blockchain Security Technology for IoT Services) and this work was partly supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (No.

NRF-2020R1F1A1048478). This research was financially supported by Hansung University for Hwajeong Seo.

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

References

1. Mohurle, S.; Patil, M. A brief study of Wannacry threat: Ransomware attack 2017. *Int. J. Adv. Res. Comput. Sci.* **2017**, *8*.
2. Kharaz, A.; Arshad, S.; Mulliner, C.; Robertson, W.; Kirda, E. UNVEIL: A large-scale, automated approach to detecting ransomware. In Proceedings of the 25th USENIX Security Symposium (USENIX Security 16), Austin, TX, USA, 10–12 August 2016; pp. 757–772.
3. Weckstén, M.; Frick, J.; Sjöström, A.; Järpe, E. A novel method for recovery from Crypto Ransomware infections. In Proceedings of the 2016 2nd IEEE International Conference on Computer and Communications (ICCC), Chengdu, China, 14–17 October 2016; pp. 1354–1358.
4. Tseng, A.; Chen, Y.; Kao, Y.; Lin, T. *Deep Learning for Ransomware Detection*; IEICE Technical Report; IEICE: Tokyo, Japan, 2016; Volume 116, pp. 87–92.
5. Vinayakumar, R.; Soman, K.; Velan, K.S.; Ganorkar, S. Evaluating shallow and deep networks for ransomware detection and classification. In Proceedings of the 2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI), Udupi, India, 13–16 September 2017; pp. 259–265.
6. Poudyal, S.; Dasgupta, D.; Akhtar, Z.; Gupta, K. A multi-level ransomware detection framework using natural language processing and machine learning. In Proceedings of the 14th International Conference on Malicious and Unwanted Software MALCON, Nantucket Island, MA, USA, 1–4 October 2019.
7. Gröbert, F.; Willems, C.; Holz, T. Automated identification of cryptographic primitives in binary programs. In Proceedings of the International Workshop on Recent Advances in Intrusion Detection, Menlo Park, CA, USA, 20–21 September 2011; Springer: Berlin/Heidelberg, Germany, 2011; pp. 41–60.
8. Lestringant, P.; Guihéry, F.; Fouque, P.A. Automated identification of cryptographic primitives in binary code with data flow graph isomorphism. In Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security, Singapore, 14–17 April 2015; pp. 203–214.
9. Kiraz, M.S.; Genç, Z.A.; Öztürk, E. Detecting Large Integer Arithmetic for Defense Against Crypto Ransomware. 2017. Available online: https://eprint.iacr.org/2017/558/20170608:200345 (accessed on 7 March 2021).
10. Kim, H.; Park, J.; Kwon, H.; Jang, K.; Choi, S.J.; Seo, H. Detecting Block Cipher Encryption for Defense Against Crypto Ransomware on Low-End Internet of Things. In Proceedings of the International Conference on Information Security Applications, Jeju Island, Korea, 26–28 August 2020; Springer: Berlin/Heidelberg, Germany, 2020; pp. 16–30.
11. Yaqoob, I.; Ahmed, E.; ur Rehman, M.H.; Ahmed, A.I.A.; Al-garadi, M.A.; Imran, M.; Guizani, M. The rise of ransomware and emerging security challenges in the Internet of Things. *Comput. Netw.* **2017**, *129*, 444–458. [CrossRef]
12. Azmoodeh, A.; Dehghantanha, A.; Conti, M.; Choo, K.K.R. Detecting crypto-ransomware in IoT networks based on energy consumption footprint. *J. Ambient Intell. Humaniz. Comput.* **2018**, *9*, 1141–1152. [CrossRef]
13. Azmoodeh, A.; Dehghantanha, A.; Choo, K.K.R. Robust malware detection for internet of (battlefield) things devices using deep eigenspace learning. *IEEE Trans. Sustain. Comput.* **2018**, *4*, 88–95. [CrossRef]
14. Zahra, A.; Shah, M.A. IoT based ransomware growth rate evaluation and detection using command and control blacklisting. In Proceedings of the 2017 23rd International Conference on Automation and Computing (ICAC), Huddersfield, UK, 7–8 September 2017; pp. 1–6.
15. Karimi, A.; Moattar, M.H. Android ransomware detection using reduced opcode sequence and image similarity. In Proceedings of the 2017 7th International Conference on Computer and Knowledge Engineering (ICCKE), Mashhad, Iran, 26–27 October 2017; pp. 229–234.
16. 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, Chengdu, China, 21–23 March 2018; pp. 81–85.
17. Dinu, D.; Biryukov, A.; Großschädl, J.; Khovratovich, D.; Le Corre, Y.; Perrin, L. FELICS—Fair evaluation of lightweight cryptographic systems. In Proceedings of the NIST Workshop on Lightweight Cryptography, Gaithersburg, MD, USA, 20–21 July 2015; Volume 128.
18. Williams, J.L.; Fisher, J.W.; Willsky, A.S. Approximate dynamic programming for communication-constrained sensor network management. *IEEE Trans. Signal Process.* **2007**, *55*, 4300–4311. [CrossRef]
19. Caballero, J.; Poosankam, P.; Kreibich, C.; Song, D. Dispatcher: Enabling Active Botnet Infiltration Using Automatic Protocol Reverse-Engineering. In Proceedings of the 16th ACM Conference on Computer and Communications Security CCS '09, Chicago IL, USA, 9–13 November 2009; Association for Computing Machinery: New York, NY, USA, 2009; pp. 621–634. [CrossRef]