Payload-based Packet Classification using Deep Learning

Michael Shell[1], Homer Simpson[2], James Kirk[3], Montgomery Scott[3], and Eldon Tyrell[4], *Fellow, IEEE*
[1]School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA 30332 USA
[2]Twentieth Century Fox, Springfield, USA
[3]Starfleet Academy, San Francisco, CA 96678 USA
[4]Tyrell Inc., 123 Replicant Street, Los Angeles, CA 90210 USA

Recently, with the advent of various applications, a huge amount of network traffic is generated on the network. Because of this, it has a big influence on the IoT network which has to provide fast service. Accordingly, the network operator must provide the Quality of Service according to the service provided by each application. To do this, research should be done on how to classify various types of application network traffic accurately and how to handle high-capacity traffic generated on high-speed links. Also, in order to automatically respond to real-time applications, it is necessary to classify traffic using deep learning without network operator intervention. In this paper, we generate training data set of packet unit and flow unit through our own network traffic preprocessing and we have 5 Deep Learning Models (Convolution Neural Network, Residual Network, Recurrent Neural Network, Long Short-Term Memory CNN+RNN) to perform network traffic classification. Finally, we analyze the network traffic classification performance of packet unit and flow unit through f1-score using five deep learning models and demonstrate its effectiveness.

***Index Terms*—Packet Classification, Deep Learning, CNN, RNN, ResNet, LSTM, Model Tuning**

I. Introduction

Recently, the importance of network operation and management has been emphasized due to the occurrence of various services and applications. Especially due to the rapid growth of IoT, various related applications and services are being provided through the network. Therefore, the network operator needs differentiated services according to applications provided through the network. In particular, video and voice services require fast network services. On the other hand, services such as text can provide enough services without having a fast transmission speed. In addition, Peer-to-Peer (P2P) services such as BitTorrent take up a significant portion of the world's Internet traffic, which has a significant impact on overall network speed. Thus, according to each service, the network operator tries to provide smooth Quality of Service (QoS) by placing different priority.

Research on how to classify network traffic in order to provide smooth services according to application programs is actively being conducted. There are various ways to classify network traffic, such as rule-based, port-based, flow-correlation-based, payload-based and using deep-learning. In general, rule-based and port-based network traffic classification methods are widely used. However, rule-based and port-based network traffic classification methods are classified by well-known application port numbers or specific rules but are not well classified for unknown applications because they do not know the port number or specific rules. Also, a network operator has the inconvenience of manually adding a rule or port number in order to provide a new network service.

The payload-based network classification method classifies only the pure application layer payload information excluding the packet header information of the entire network traffic. There is a problem with locality dependency when packet classification is performed by adding header information. Also, if the header information is changed, classification will not be performed properly. But, the payload-based network classification method solves the locality dependency problem because it uses only the payload data, and it is not affected even if the header information is changed. Additionally, current payload-based methods provide the best classification accuracy. However, there is a practical problem due to the difficulty of accessing the payload of the raw packet and the encryption due to the user privacy policy.

Recently, researches on the field of deep learning have progressed actively and have become applicable to various fields. So, there is much research on network traffic classification using deep learning. In the network traffic classification study using the existing deep learning model, classification is performed using the header information of the packet as a feature. Therefore, there is a limitation because the classification is performed within a local network. In a real network that is outside the local limits, it is difficult to classify through the previously learned model.

In this paper, the datasets of the deep learning model create training datasets from the network traffic through the self-developed data preprocessing. Through preprocessing, one packet in network traffic is imaged and generated as training data. The imaged packets are gathered for each application and are made up of training datasets of packet unit or flow unit. Convolution Neural Network(CNN), Residual Network (ResNet), Recurrent Neural Network (RNN), Long Short-Term Memory (LSTM), and CNN + RNN are learned to classify network traffic using the generated packet unit and flow unit training datasets. CNN and ResNet models are suitable for classification of image data, they are used for learning imaged datasets of packet unit. The RNN, LSTM, and CNN + RNN learning models are suitable for learning sequential datasets. Therefore, They are used to learn flow unit datasets that contain time sequential of network traffic.

Manuscript received December 1, 2012; revised August 26, 2015. Corresponding author: M. Shell (email: http://www.michaelshell.org/contact.html).

To enhance the performance of network traffic classification, we added a model tuning procedure to find optimal hyper-parameters of each deep learning model. Then, we compare the packet unit and the flow unit datasets by using five different deep learning models.

The remainder of this paper is organized as follows. Section II describes related work and introduces motivation for our work. Section III describes the deep learning models architecture for network traffic classification. Section IV presents the model tuning method for optimal hyper-parameter. Section V shows experiment results and analysis. Section VI provides concluding remarks.

II. Related Work

Recently, there are many researches and technologies for packet classification in networks. In the existing research, there is a rule-based packet classification method. Recently, research on deep learning has been developed and research on packet classification using deep learning has been actively carried out. Packet classification using deep learning is a method of automatically classifying packets without human intervention. In this chapter, we study the rule-based packet classification research and the packet classification studies that utilized deep learning.

A. Rule-based Packet Classification

Rule-based packet classification is a method of classifying packets entering the network according to predefined rules. The rule-based packet classification method is classified by using the header information of the network packet. Therefore, rule-based packet classification is performed based on the source and destination IP and port of the packet header.

So, there are limitations to the rule-based packet classification method. Since the packet is classified using the information of the packet header, if the information of the packet that doesn't match the packet is received, the packet can't be classified or classified differently. In addition, because of rule-based packet classification with IP and port information of packet header, there are local limitations. Therefore, when a new network accesses or packets of a new network occur, there is a problem that a new rule must be redefined.

fangfan Li et al. [] At least it was used to lower the dependency of packet header information. Using this approach, they found that our packets having device uses HTTP and TLS-handshake fields in their matching rules, and only for the first packet in each direction. If there is similar information in the header information of the incoming packet using the header information found in the first packet, it is classified as the packet of the same type. Although the IP and port number of the packet is used less, the method of classifying the subsequent packets by using the header information of the first packet also depends on the header information.

B. Classify Packets Using Deep Learning

Several studies using deep learning have been conducted recently. Also, researches to utilize machine learning in networks are actively being conducted. Accordingly, studies are being actively carried out to perform packet classification using deep learning of network.

Wei Wang et al. uses CNN model to classify malware traffic and general traffic. First, if 5-tuple (source IP/port, destination IP/port, protocol) are the same among the packets, one flow is defined as one dataset. In addition to the flow dataset, packets are defined as a session set as a dataset. The session dataset is a case where 5-Tuple is paired with the same flow and the same source IP/port and Destination IP/port cross each other. When data is divided into actual flow and session, datasets is constructed by removing information of IP and MAC address. The reason is that IP and MAC address can show certain characteristics. The flow and session data sets constructed above are composed of 28 * 28 data similar to the MNIST dataset. The constructed dataset is used to learn CNN model to classify malware traffic and general traffic. In this paper, the result of classification of malware traffic and general traffic is 100

M. Lopez-Martin et al. used packet classification using CNN and RNN combination of deep learning model. The packet data is extracted from the header information and the payload data in the packet using the DPI Tool and used as learning data. The extracted learning data was used as input data to the combined model of CNN and RNN. They showed that CNN and RNN combined better than CNN and RNN models. In both of the above papers, learning was performed on the deep learning model by adding the header information of the packet to the dataset. In such a case, the classification accuracy may be high because the header information can be certain information that characterizes the data to some extent. Therefore, in this paper, we will perform learning only with payload data of application layer except for header information of the packet.

C. Payload-based Traffic Classification

Payload-based traffic classification is a replacement for rule-based and port-based traffic classification. However, because it retrieves the payload for known application features, higher computation and resource costs are incurred. It can also be difficult to maintain and manage the growing number of application features. In addition, as security and privacy issues increase, payloads are often encrypted and prohibited by privacy laws. Therefore, it is not easy to deduce the signature for an application class using a payload.

Haffner et al. [] reduced the amount of computation that occurs when generating payload-based datasets. To reduce the amount of computation, they used only the first few bytes of unidirectional traffic data and unencrypted TCP data. They use NB, AdaBoost, and MaxEnt for traffic classification. AdaBoost outperforms NB and MaxEnt and yields an overall precision of 99% with an error rate within 0.5%.

Despite, payload-based classification methods are used to provide high accuracy and establish ground truths.

III. Data Preprocessing

In this section, the dataset used to perform the traffic classification is introduced to preprocess the PCAP file provided

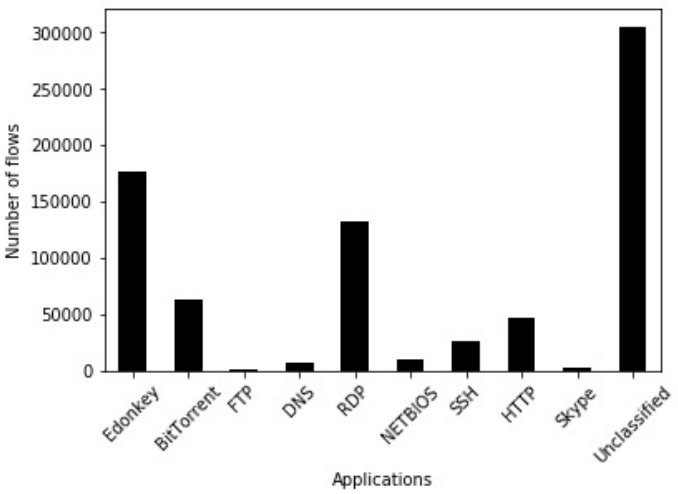

Fig. 1. BroadBand Communication Research Group PCAP Data Flow

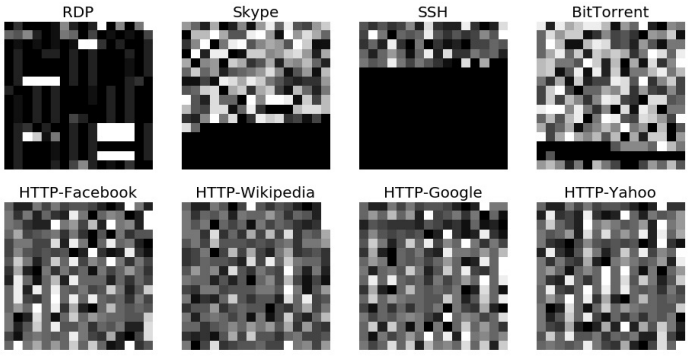

Fig. 2. BroadBand Communication Research Group PCAP Data Flow

by Broadband Communication Research Group [] for the deep learning. A PCAP (Packet Capture) file captures and stores network packets using programs such as Wireshark and TCPdump. Therefore, in this paper, traffic dataset provided by Broadband Communication Research Group is divided into flow unit and packet unit, and the preprocessing packet is used as learning data.

A. Data Split

he size of the original PCAP file provided is about 59 GB, and there are a total of 769,507 flows in the file. Fig. 1 is a graph showing the number of flows per application. The info file provided with the PCAP file, there is a labeling of the traffic data. Labeling is divided into three categories: application type, protocol, and application name for the corresponding traffic flow. Because of the labeling file, accurate label information corresponding to the group truth in the flow-based traffic classification can be obtained by using the deep learning model proposed in this paper.

For the preprocessing of the supplied data, 8 kinds of application were selected based on the number of flows. The 8 applications selected the application that has the traffic of the actual network and the number of flow is more than 2000. The 8 most common Label names are the Remote Desktop Protocol (RDP), Skype, SSH, BitTorrent, HTTP-Facebook, HTTP-Google, HTTP-Wikipedia, and HTTP-Yahoo. The application layer payload of the selected flow internal packet is filtered and extracted, and 8 application layer payload data files are generated using the extracted payload data.

B. Learning Data Generation

he process of converting the application layer payload data file created in the above section into input data suitable for deep learning will be described. The application payload data file is divided into per packet unit and per flow unit, and each learning data is generated. Flow is the same as the 5-tuple of the packet's header information, and packets generated within 3600 seconds of the previous packet are bundled into the same flow.

Each packet is extracted from each application layer payload data, and the elements of the payload data of the packet are grouped by 4 bits into one unit of learning data. Therefore, one pixel of the learning data represents the number of 0 to 15. One packet data collects pixelized data and image data as shown in Fig.2. One image data is converted into one packet learning data.

The learning data is generated by the packet unit and the flow unit for each application. The learning data for each application per packet generates learning data by arbitrarily extracting packets of 8 applications (RDP, SSH, Skype, BitTorrent, Facebook, Wikipedia, Google, Yahoo) from the application layer payload data file. In each application, 10000 random packets were extracted to extract a total of 80,000 learning data. One packet of each randomly extracted application is resized according to the size of the payload. Therefore, the payload size of each application packet is extracted as 36 (6 * 6), 64 (8 * 8), 256 (16 * 16), and 1024 (32 * 32). Figure 2 shows 16 * 16 of the payload data of an arbitrary packet for each extracted application. If the extracted payload size is smaller than the set size, the size is adjusted by zero padding to match the set size.

The learning data of the flow unit is arbitrarily selected for each application in the application layer payload data file. The selected flow fetches the packets from the first N packets as the number of packets (N) per predetermined flow. They are packets that have undergone a preprocessing process as in the learning data of each packet unit. The learning data of the flow unit extracts 2000 flows for each application from the application layer payload data file and has a total of 16000 flows. The number of N is set to 30, 60, and 100, and packets of each flow are fetched by the corresponding number to generate learning data.

Then, it is expressed as a one-hot vector with 8 lengths so that each of 8 applications can have label data. A one-hot vector label is a vector with a value of only one element. A one-value index can be defined as a label representing an application. Therefore, two sets of learning data are used as the learning data packet or flow data and label data indicating the application data.

IV. Deep Learning Models

This chapter describes learning models that are used to classify network traffic using deep learning. The Deep Learning

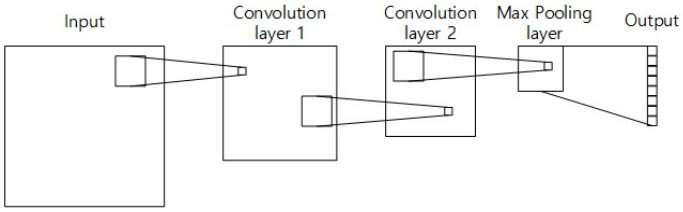

Fig. 3. 2-Layer Convolution Neural Network Learning Model

model used is the Convolution Neural Network (CNN), the Residual Network (ResNet), the Recurrent Neural Network (RNN), the Long Short-Term Memory LSTM) and the Convolution and Recurrent Neural Network (CNN + RNN). The Deep Learning model was supported by Keras, and the ResNet and CNN + RNN models were generated using Keras' CNN and RNN models simultaneously.

CNN and ResNet models are commonly used for information extraction, sentence classification, face recognition, and image classification. CNN is a structure that extracts characteristics of data and grasps patterns of features. In the case of RNN and LSTM, it is a model specialized for repetitive and sequential data learning. Therefore, the previous learning data is reflected in the current learning using the circulation structure. It is generally used for the composition of speech, wave, and text. Therefore, in the case of CNN and ResNet, it is used to classify using imaged packet unit data generated through preprocessing. RNN, LSTM, and CNN + RNN models are used to classify sequential data, so they are used to classify learning data in flow units that contain sequential information of network traffic.

A. Convolution Neural Network Architecture

CNN model among the deep learning models for classifying network traffic will be described. The model architecture of CNN is composed of the input layer, Convolution layer, and Pooling layer and Fully connected layer. The input layer uses the payload and label of the packet converted into learning data. Packets are used as input data in the input layer in the form of $N \times N$ (N = 6, 8, 16, 32) like images. Then, the feature of each packet data is convolved through the kernel of two Convolution layer, and output is generated through filter and activation process. In the pooling layer, it is the process of reducing the size of the output through the convolution process. It simply reduces the size of the data, cancels noise, and provides consistent features in fine detail. Finally, the Fully connected layer extracts the prediction value according to the last 8 classes by activation.

Fig. 3 is CNN architecture

B. Residual Network Architecture

nlike traditional CNN, ResNet [] has a unique concept called shortcut connection. A shortcut connection is added to the existing CNN model structure, and this shortcut is directly connected without any other parameters. A shortcut connection is a type of adding a new type of network to an input value so that learning can be performed. Therefore, the newly added network can achieve better performance while maintaining the performance of the existing learned network as much as possible.

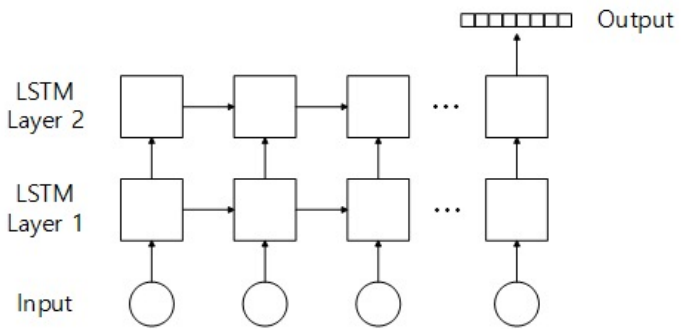

Fig. 4. 2-Layer Convolution Neural Network Learning Model

In terms of computation, there is nothing more than adding an operation. Deep networks can be easily optimized through shortcut connections and can improve accuracy as depth increases.

C. Recurrent Neural Network Architecture

NN is a network architecture that can accept inputs and outputs regardless of input data length and can be implemented variously and flexible as needed. Therefore, the architecture of RNN used in this paper is composed of multi-layer as shown in figure 4. In the RNN, the number of packets per flow (30, 60 and 100) is received at the input layer in order to learn flow unit data. The number of units to be set is then output to the number of applications learned in the output layer through the RNN cell.

D. Long Short-Term Memory Architecture

n addition to the existing RNN model, LSTM determines whether to keep the weight value by adding another feature layer called a cell state. Through this, we solve the phenomenon that the weight value is not maintained as the distance between information and information of one input data in the existing RNN becomes longer, and the learning ability decreases. LSTM is more persistent than existing RNN because it keeps updating the past data. The cell state is responsible for adding or deleting information. The structure of the LSTM model is configured as shown in Fig.4.

The advantage of LSTM is that each memory control is possible and the result can be controlled. However, there is a possibility that the memory may be overwritten, and the operation speed is slower than that of the conventional RNN. Therefore, it is composed of two layers different from existing RNN model.

V. Model Tuning

The CNN, RNN and LSTM models provided by Keras each have various hyper-parameters. Therefore, the performance of the model can be greatly changed by using certain hyper-parameters. In addition, the performance of the model may vary depending on the characteristics of the training datasets.

TABLE I
CNN HYPER-PARAMETER VALUES

	$6 \times 6(36)$	$8 \times 8(64)$	$16 \times 16(256)$	$32 \times 32(1024)$
filter	18	32	256	512
kernel size	3×3	5×5	5×5	3×3
kernel initializer	glorot uniform	uniform	uniform	uniform
padding	same	same	same	same
activation	softmax	softmax	softmax	softmax
optimizer	adam	adam	adam	adam
batch size	100	100	10	10

TABLE II
RESNET HYPER-PARAMETER VALUES

	$6 \times 6(36)$	$8 \times 8(64)$	$16 \times 16(256)$	$32 \times 32(1024)$
filter	18	32	256	512
kernel size	3×3	7×7	5×5	7×7
kernel initializer	glorot uniform	glorot uniform	uniform	glorot uniform
padding	same	same	same	same
activation	softmax	softmax	softmax	softmax
optimizer	adam	adam	adam	rmsprop
batch size	100	100	100	100

To do this, we use GridSearchCV, which is provided by Scikit-Learn, as a method of finding hyper-parameters optimized for each deep learning model according to the datasets. The GridSearch method looks for the best hyper-parameter for a dataset by trying every possible combination of hyper-parameters based on the dataset. GridsearchCV verifies the validity of the model by performing cross-validation in addition to finding the optimal hyper-parameters. Cross-validation is a method of dividing specific data into training-only data and test-specific data, then using training data to learn, and testing with test data to verify the validity of the learning. In this model tuning, we set the value of CV to 5 to find the optimal hyper-parameters and verify the validity of the model and hyper-parameters at the same time.

A. CNN and ResNet Model Tuning

he learning dataset of CNN and ResNet consists of packet unit data. For each of the 8 applications, 10,000 packets were randomly organized into a single learning data set and the payload sizes of each packet were matched. The payload size was divided by 36 (6 * 6), 64 (8 * 8), 256 (16 * 16), and 1024 (32 * 32) to increase the size of the dataset. Therefore, the shape of the total training datasets is (80000, 6, 6, 1), (80000, 8, 8, 1), (80000, 16, 16, 1), (80000, 32, 32, 1).

There are a total of 15 hyper-parameters used in the CNN model provided by Keras. Four of CNN hyper-parameters were selected. *filter* represents the size of the output through one convolution layer, *kernel size* represents the size of the kernel used in the *filter*, *kernel initializer* represents the parameters for initializing the weight vectors of each layer, and *padding* represents the output of the Convolution layer. It is necessary to fill in the pixel value specified by the outer angle of the input data with a specific value. We perform GridSearch by adding *activation*, *optimizer*, and *batch size* which affects the whole learning in addition to the hyper-parameters to CNN.

Table 1 and 2 shows the values of the gridsearch performed for the selected hyper-parameter.

B. RNN and LSTM Model Tuning

he training datasets of SimpleRNN and LSTM is data of flow unit. The training dataset was constructed by arbitrarily fetching 2000 flows from 8 applications. In addition, the payload size of each flow unit packets has 36, 64, 256, and 1024, as packet unit. The training data set of the flow unit is a data set that collects sequentially generated packets within the same 5-tuple and within 3600 hours. The input data of the RNN must match the number of packets included in the flow. Therefore, the number of packets per flow is set to 30, 60, and 100. The final shape of the training data set is (16000, 30, 36), (16000, 30, 64), (16000, 30, 256) 16, 100, 256, 16000, 100, 36, 16000, 100, 64, 16000, 1024).

Keras The hyper-parameters used in RNN and LSTM models are 20 and the hyper-parameters of RNN and LSTM are the same. We selected dual *units*, *kernel initializer*, *recurrent initializer*, and *dropout*. *Units* represents the space of the output dimension, *kernel initializer* initializes the weight vector values of RNN and LSTM, *recurrent initializer* initializes the weight vector of recurrent state, *dropout* is a number between 0 and 1 It is a variable that deletes by the percentage of the number arbitrarily set in the number of units. Like CNN, we perform GridSearch by adding *activation*, *optimizer*, and *batch size*.

Table 3 and 4 shows the values of variables that perform gridsearch for the selected hyper-parameter units, *kernel initailizer*, *recurrent initializer*, *dropout*, *activation*, *optimizer*, *batch size*.

VI. EXPERIMENTS EVALUATION

In this section, we compare packet-based application prediction and flow unit application prediction. CNN and ResNet are used for packet unit prediction. RNN and LSTM are used for flow unit prediction. In addition, we compare the performance of the model with the flow unit and the packet unit through the overall comparison of CNN, ResNet, RNN, and LSTM.

A. Experiments Environment

The learning model is based on CNN, RNN, and LSTM supported by Keras 2.2.0. In case of ResNet, network model was constructed using CNN model supported by Keras. Model learning epochs were set to 200.

The experiments environment was run on Ubuntu 16.04 LTS, using 32GB of RAM and two NVIDIA GTX 1080Ti 11GB. Experiments implementation uses version 1.8 of Tensorflow-gpu and version 2.2.0 of Keras in Python 3.6. We were also able to utilize the GPU with CUDA 9.0 and CuDNN version 7.1.2.

The experiment performs Scikit-learn's GridSearchCV for one dataset and finds the optimal hyper-parameters through each model tuning. The model learning was performed using optimal hyper-parameters for each dataset in Tables 1, 2, 3 and 4 of the Section.V.

B. Performance Metrics

In this paper, we use accuracy, precision, recall, and f1-score for performance comparison of CNN, ResNet, RNN and LSTM models. Accuracy, recall, precision, and f1-score provide performance metrics that take into account the unbalanced distribution of each application. f1-score is the most important metric in all metrics. f1-score represents the harmonic mean of precision and recall and represents the classification performance of unbalanced datasets. The f1-score is expressed as 0 to 1 and is the best value at 1.

The definition of accuracy, precision, recall, and f1-score follows four previous definitions as follow: First, False Positive (FP) indicates that the prediction is that the application is correct, but not actually the application. Second, False Negative (FN) indicates that the application is not expecting the result, but the application is actually correct. Third, True Positive (TP) indicates that the application is correct and the application is correct. Finally, True Negative (TN) indicates that the application is not the result of the prediction, but is not actually the application. The definition of accuracy, precision, recall, and f1-score according to the previous definition is as follows:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{1}$$

$$Recall = \frac{TP}{TP + FP} \tag{2}$$

$$Precision = \frac{TP}{TP + FN} \tag{3}$$

$$F1 - score = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{4}$$

C. Experiments Results

The first experiment is a comparison of CNN and ResNet with the packet unit datasets. In order to compare CNN and ResNet, the total f1-score was compared by varying the payload size of the packet to $6\times6(36)$, $8\times8(64)$, $16\times16(256)$, and $32 \times 32(1024)$.

Fig.5 compares f1-score for all applications by payload size in CNN and ResNet. If the payload size of the packet is small, it can be seen that the overall f1-score value of CNN is about 0.4 higher than ResNet. However, as the payload size increases, the training data set size also increases. As a result, f1-score of ResNet is larger than CNN. The reason is that because the learning model of ResNet is complex, less learning data sets are not good for learning. Conversely, the larger the payload size and the larger the training data set, the better the performance of the more complicated ResNet model.

The following experiment is a comparison of RNN, LSTM, and CNN + RNN using flow unit datasets. Fig.6 shows the total f1-score for each packet's payload size and the number of packets per flow. It can be seen that the overall f1-score increases as the payload size of the packet increases. Also, as the number of packets per flow increases, the entire f1-score increases. When the number of packets per flow is 30 and the payload size is 36, the f1-score of each RNN, LSTM, and

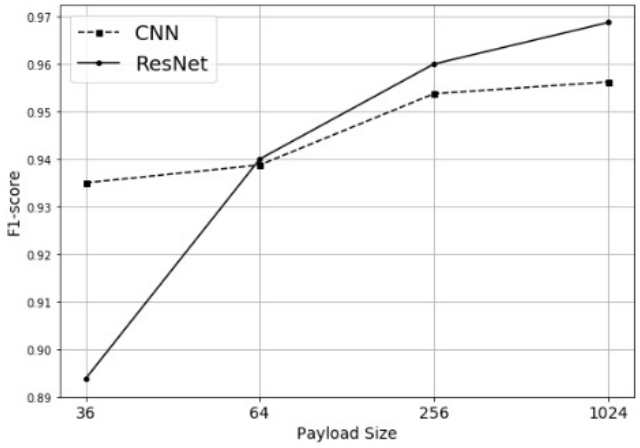

Fig. 5. Comparison of CNN and ResNet with the overall f1-score for each application

CNN + RNN is 0.885, 0.89375 and 0.89. However, when the number of packets per flow is 100 and the payload size is 1024, each f1-score thereof is 0.972, 0.99575 and 0.9925. It is noticed that the f1-score across LSTM is higher than the total F1-score RNN and CNN + RNN. It indicates that LSTM alone can provide similar or better performance without using complex models in the flow unit learning model.

Fig.7 is a graph comparing the f1-score results for each application when using a deep learning model to compare packet units and flow unit classifications. In the case of CNN and ResNet, it represents the f1-score for each application when the payload size is $32 \times 32(1024)$. Also, in the case of RNN, LSTM, and CNN + RNN, it indicates the f1-score for each application when the payload size is 1024 and the number of packets per flow is 100. Packet unit learning models CNN and ResNet show that f1-score on Facebook and Wikipedia are smaller than other applications. The facebook and wikipedia packets are very similar to each other, so they are smaller than the f1-score of other applications' packets. In the case of RNN, we can see that f1-score value of Facebook and Wikipedia are low as in the case of packet unit classification. However, in the case of LSTM and CNN + RNN, the f1-score of both applications is high. Thus, we can see that the LSTM and CNN + RNN learning models are well-categorized and that the LSTM performs better with subtle differences.

VII. Conclusion

In this paper, we classify network traffic using deep learning. In order to classify the network traffic using deep learning, the imaged packet data was generated through preprocessing which was developed by itself, and the packet and the flow unit training data set were created by gathering the packets. We also used model tuning to find the hyper-parameters that best matched each deep learning model. The model of CNN, ResNet, RNN, LSTM, and CNN + RNN compared the packet and flow unit the result of network traffic classification. As a result, the network traffic classification of the flow unit showed better performance. Among the deep learning models, the LSTM and CNN + RNN models performed well, and the

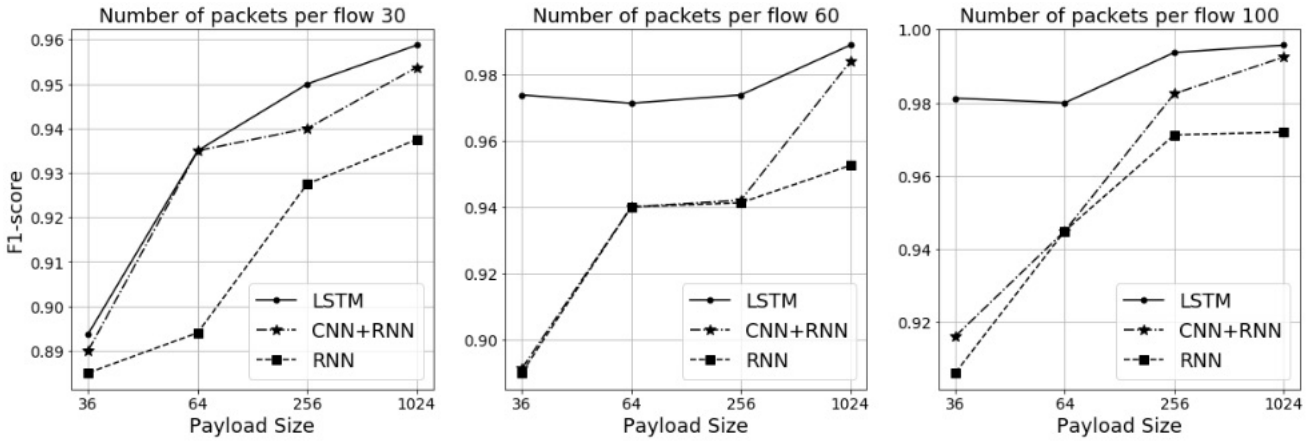

Fig. 6. Comparison of RNN, LSTM and CNN+RNN with the overall f1-score for each application

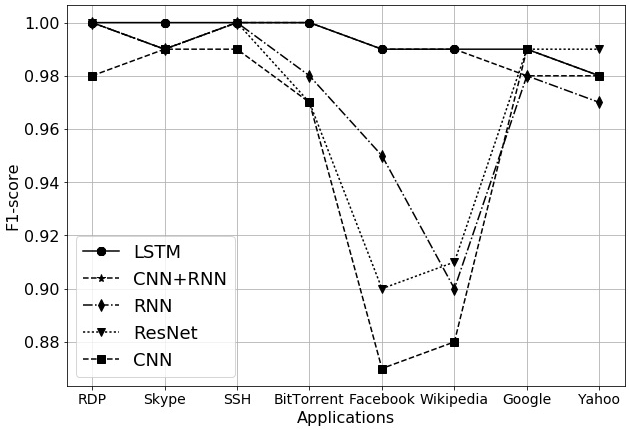

Fig. 7. Comparison of packet unit classification and flow unit classification

LSTM model performed better. Therefore, when using LSTM, it is possible to classify network traffic provided to IoT with high accuracy and to provide better network service.

APPENDIX A
PROOF OF THE FIRST ZONKLAR EQUATION

Appendix one text goes here.

APPENDIX B

Appendix two text goes here.

ACKNOWLEDGMENT

The authors would like to thank...

REFERENCES

[1] H. Kopka and P. W. Daly, *A Guide to LaTeX*, 3rd ed. Harlow, England: Addison-Wesley, 1999.

PLACE PHOTO HERE

Michael Shell Biography text here.

John Doe Biography text here.

Jane Doe Biography text here.