Classification of Application Layer Protocols using Deep Learning Methods

Report submitted in fulfillment of the requirements for the Exploratory Project of

Second Year IDD

by

Ujjawal Modi

Under the guidance of

Dr. Mayank Swarnkar

Department of Computer Science and Engineering

INDIAN INSTITUTE OF TECHNOLOGY (BHU)
Varanasi 221005, India

May 2023

Dedicated to

My parents, teachers,.....

Declaration

I certify that

1. The work contained in this report is original and has been done by Me and my teammate Aditya Anand(21075006) under the general supervision of my supervisor.
2. The work has not been submitted for any project.
3. Whenever I have used materials (data, theoretical analysis, results) from other sources, I have given due credit to them by citing them in the text of the thesis and giving their details in the references.
4. Whenever I have quoted written materials from other sources, I have put them under quotation marks and given due credit to the sources by citing them and giving required details in the references.

Place: IIT (BHU) Varanasi
Date: 15 May 2023

Ujjawal Modi
IDD Student
Department of Computer Science and Engineering,
Indian Institute of Technology (BHU),
Varanasi, India 221005.

Certificate

*This is to certify that the work contained in this report entitled "**Classification of Application Layer Protocols using Deep Learning Methods**" being submitted by **Ujjawal Modi (Roll No. 21074032)**, carried out in the Department of Computer Science and Engineering, Indian Institute of Technology (BHU) Varanasi, is a bona fide work of our supervision.*

Place: IIT (BHU) Varanasi
Date: 15 May 2023

Dr. Mayank Swarnkar
Department of Computer Science and Engineering,
Indian Institute of Technology (BHU),
Varanasi, India 221005.

Acknowledgments

I would like to express my sincere gratitude to Dr. Mayank Swarnkar, Assistant Professor, Department of Computer Science and Engineering, IIT (BHU) Varanasi for his valuable insights into this project.

Place: IIT (BHU) Varanasi

Date: 15 May 2023 **Ujjawal Modi**

Abstract

The rapid growth of network traffic demands efficient classification techniques to identify application layer protocols for network management and security purposes. More recently, with the development of deep learning techniques, the performance of network traffic classification has been significantly improved due to the powerful feature representations learned by deep neural networks.

In this project, we propose a classification framework that utilizes deep learning models, to classify application layer protocols based on features extracted from payload. We collected a diverse dataset comprising of network traffic samples from various protocols, including Bacnet, BJNP, DNS, NBSS, SSH etc. The extracted features were preprocessed and encoded into numerical representations suitable for deep learning models. We implemented and trained each deep learning model on the dataset. The models were evaluated using standard performance metric recall rate, which assess both overall classification correctness and the ability to correctly identify specific protocols, respectively.

Our experimental results demonstrate that deep learning models achieved promising results in classifying application layer protocols. Overall, this research contributes to the body of knowledge in network traffic analysis and classification by leveraging deep learning techniques.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Network traffic analysis is the process of capturing and analyzing data that is transmitted across a network. It involves the categorization and labeling of network traffic data based on specific attributes or characteristics. In today's interconnected world, network traffic analysis plays a crucial role in various domains such as network management, cybersecurity, and quality of service optimization. Understanding and classifying application layer protocols within network traffic is essential for effective network monitoring, security threat detection, and bandwidth allocation. Traditional methods of protocol classification often rely on port numbers, which can be easily manipulated or obfuscated by attackers. To overcome these limitations, this project focuses on the application of deep learning models for accurate classification of application layer protocols based on the payload content.

To achieve our project goal, a systematic approach was followed. Firstly, we gather a diverse dataset comprising network traffic samples from various protocols, including Bacnet, Kerberos, DNS, QUIC, and Bootp. We utilize pcap files containing network traffic data and performed flow reconstruction to group packets based on source and destination IP addresses and port numbers. This process allows us to isolate and analyze individual flows within the network traffic.

Next, we extracted the payload data from each packet in the reconstructed flows, specifically focusing on the first 32 bits of the payload. These extracted bits served as the input to our deep learning models. By training the models on the labeled dataset, we aim to optimize their performance in classifying different application layer protocols accurately.

Throughout the project, we tried to evaluate the effectiveness of the novel approach i.e. classification by just using the 32 bits of the payload with the use of Deep Learning models by measuring their recall rates. Recall measures the model's ability to correctly identify specific protocols, considering both true positives and false negatives. The findings from this research have significant implications for network management, security, and traffic analysis. In the subsequent sections of this report, we will discuss the work done , method used , results, and analysis of the deep learning models' performance and finally provide conclusions and suggest avenues for future research to further enhance the accuracy and efficiency of protocol classification.

1.1 Motivation of the Research Work

The motivation for proposing a novel method of network classification using extracted 32 bits as features for deep learning models is driven by many reasons. Conventional techniques for network classification, such port-based methods, have limits in their ability to correctly identify protocols. Attackers readily modify or disguise port numbers, which causes incorrect or insufficient protocol classification. The proposed method also addresses the challenge of handling encrypted traffic. With the widespread use of encryption protocols, a significant portion of network traffic is now encrypted, making it difficult to inspect the payload and extract protocol information based solely on port numbers. By utilizing deep learning models, which can learn patterns and statistical properties from the extracted bits, the proposed method aims to effectively classify encrypted traffic based on features beyond port numbers.

Chapter 2

Literature Review

2.1 Research Papers

2.1.1 A novel network traffic classification approach via discriminative feature learning

This research paper is published by Y. Liu, H. Xie, H. Peng It uses a novel feature learning approach based on discriminative autoencoder (DAE) and support vector machine (SVM) for network traffic classification.

2.1.2 A Semi-supervised Stacked Autoencoder Approach for Network Traffic Classification

This research paper is published by X. Wei, K. Sun, J. Liu It uses a semi-supervised stacked autoencoder (SSAE) for network traffic classification that requires fewer labeled samples.

2.1.3 Efficient Keyword Matching for Deep Packet Inspection based Network Traffic Classification

This research paper is published by S. Jangir, S. Laxmi, M. .It uses a keyword matching approach for deep packet inspection (DPI) based network traffic classification.

2.1.4 Multitask Learning for Network Traffic Classification

This research paper is published by Q. Liu, J. Liu, K. Lin. It uses a multitask learning (MTL) approach for network traffic classification that can handle multiple traffic classes.

2.1.5 Network traffic classification using deep convolutional recurrent autoencoder neural networks for spatial–temporal features extraction

This research paper is published by M. Ahmed, A. Bounceur, M. Ahmed. It uses deep convolutional recurrent autoencoder neural networks (DCRNNs) for spatial-temporal features extraction for network traffic classification

2.1.6 Predicting Network Flow Characteristics Using Deep Learning and Real-World Network Traffic

This research paper is published by F. Wang, L. Zheng, Y. Liu. It uses deep learning techniques to predict network flow characteristics such as packet size, interarrival time, and flow duration.

2.1.7 SAM: Self-Attention based Deep Learning Method for Online Traffic Classification

This research paper is published by Y. Zhang, C. Li, Y. Wang. It uses a self-attention based deep learning method called SAM for online network traffic classification.

2.1.8 Efficient Traffic Classification Using Hybrid Deep Learning

This research paper is published by C. Luo, X. Wu, Q. Zhang It uses a hybrid deep learning method that combines convolu- tional neural networks (CNNs) and long short-term memory (LSTM) networks for network traffic classification.

2.1.9 Byte-Label Joint Attention Learning for Packet-grained Network Traffic Classification

This research paper is published by C. Wang, Z. Wei, C. Liu. It uses byte-label joint attention learning (BL-JAL) for packet- grained network traffic classifica- tion.

2.1.10 Enhancing The Performance of Network Traffic Classification Methods Using Efficient Feature Selection Models

This research paper is published by N. Rao, G. Huang, L. Liao. It proposes an efficient feature selection model for network traffic classification that can reduce the number of features required for classification.

2.1.11 Multi class SVM algorithm with active learning for network traffic classification

This research paper is published by S. Shaik, S. Laxmi.It uses a multi-class Support Vector Machine (SVM) algorithm with an active learning approach for network traffic classification.

2.1.12 Tree-RNN: Tree structural recurrent neural network for network traffic classification

This research paper is published by Y. Chen, J. Hu, Y. Zhou . It uses a tree structural recurrent neural network (Tree-RNN) for network traffic classification.

2.2 Shortcomings in the Papers

Traditional methods of network classification had several shortcomings.

- Inability to Detect Encrypted Traffic:
  - Traditional techniques heavily rely on packet header analysis, which is ineffective for identifying encrypted traffic.
  - Encrypted traffic poses a significant challenge for accurate classification and threat detection.
- Limited Protocol Identification:
  - Traditional approaches struggle to identify protocols that utilize numerous ports or non-standard port numbers.
  - Incorrect classification of network traffic may occur due to the inability to accurately identify protocols.
- Limited Capacity to Handle High-Speed Networks:
  - Traditional methods require substantial computational resources to handle the large volumes of network traffic present in high-speed networks.
  - Inadequate capacity can result in delays, congestion, and an inability to keep up with network demands.
- Lack of Flexibility:
  - Older techniques may not easily adapt to changing network protocols and architectures.
  - Limited flexibility hinders the ability to effectively classify and handle evolving network environments.

- Limited Accuracy:
  - Traditional methods primarily rely on packet headers, which are susceptible to spoofing and manipulation.
  - Inaccurate classification of network data can lead to false positives and false negatives in threat detection.
- Insufficient Adaptation to Emerging Technologies:
  - Traditional techniques may struggle to classify traffic generated by emerging technologies, such as IoT devices or new application protocols.
  - Inadequate adaptation to new technologies limits the effectiveness of network traffic classification systems.

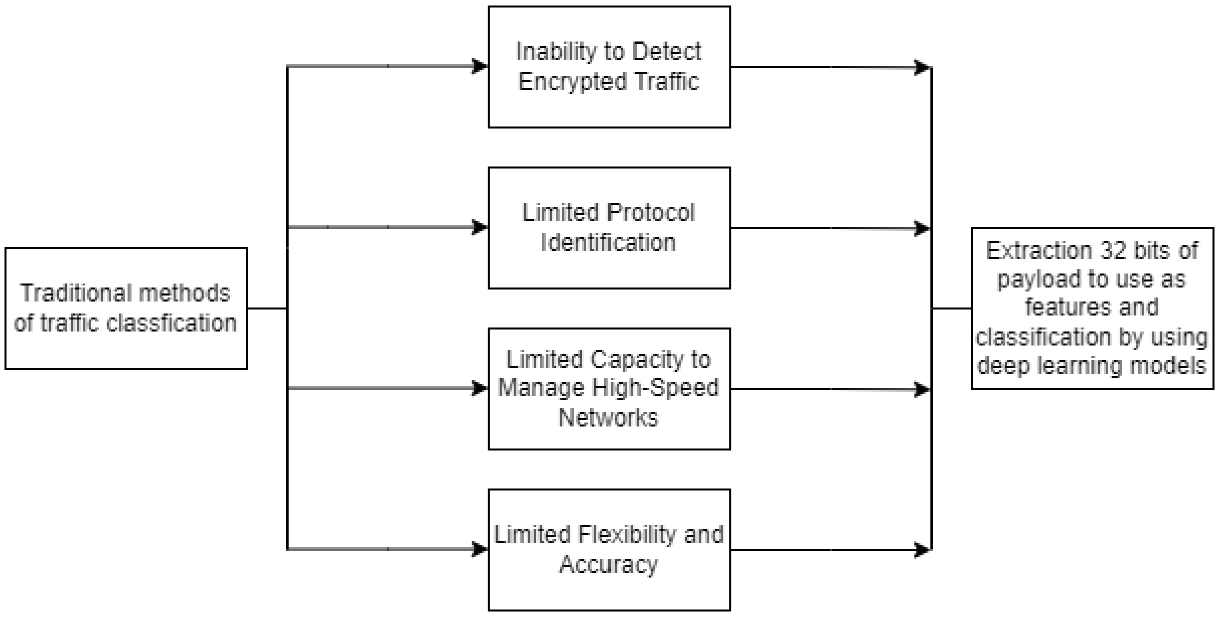
Traditional methods of traffic classfication
Inability to Detect Encrypted Traffic
Limited Protocol Identification
Limited Capacity to Manage High-Speed Networks
Limited Flexibility and Accuracy
Extraction 32 bits of payload to use as features and classification by using deep learning models

Figure 2.1 Shortcomings in previous work

Chapter 3

Methodology

We will discuss the steps, methods and techniques we have used for the project to extract the features and train the models.

3.1 Feature selection and extraction

3.1.1 What is Flow?

A flow, in the field of computer networking, refers to a sequence of packets that exhibit similar characteristics and are treated as a unified entity by network devices. In computer networking, when data is transmitted across a network, it is typically divided into smaller units called packets. These packets are then individually transmitted and may take different paths to reach their destination. The identification and classification of flows are typically based on attributes such as source and destination IP addresses, transport protocol, and port numbers.The concept of flows is of significant importance in various research areas within computer networking. Researchers often analyze flows to gain insights into network behavior, identify patterns, and detect anomalies. Flow analysis provides a higher-level perspective on network traffic, allowing for efficient management, optimization, and resource allocation. DPI (Deep Packet Inspection) and SPI (Shallow Packet Inspection) are two different techniques for classifying network traffic.

DPI, also known as Deep Packet Inspection, is a technique used to analyze and understand the contents of network packets at the application layer. It involves examining the payload or data portion of packets to gain insights into the specific protocols, applications, or services being used. DPI enables the inspection and analysis of packet content beyond the basic header information, allowing for a deeper understanding of the network characteristics. Packet level DPI and Flow level DPI are two different approaches to DPI used in network traffic analysis. Packet level DPI involves examining the contents of each individual packet for classification.Flow level DPI analyzes the behavior of each flow to determine for classification, rather than examining each individual packet.

On the other hand, SPI, or Shallow Packet Inspection, focuses on inspecting the header information of packets, such as the source and destination IP addresses, transport protocol, and port numbers. SPI provides a more limited analysis of flows compared to DPI, as it does not delve into the actual payload or application-layer data. Instead, it relies on the header attributes to make decisions regarding flow classification and treatment.

Both DPI and SPI have their respective strengths and limitations in flow analysis. DPI provides a more comprehensive understanding of flows by inspecting the contents of packets, allowing for detailed protocol identification and application-level analysis. This level of insight is beneficial for tasks such as traffic classification, performance optimization, and security analysis. SPI offers a lightweight and less resource-intensive approach for flow analysis. By focusing solely on packet headers, SPI can provide a quick classification of flows based on basic attributes. It is useful in situations where a more in-depth analysis of packet content is not necessary or feasible, such as in high-speed networks or resource-constrained environments.

The transport layer is responsible for managing the transfer of packets over a network. The two most commonly used transport layer protocols are the Transmission Control Protocol (TCP) and the User Datagram Protocol (UDP).

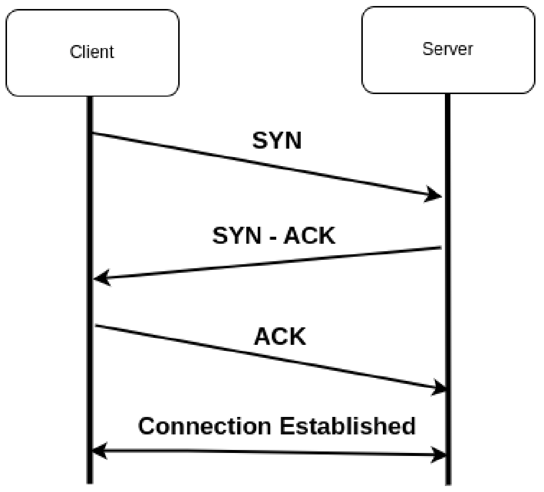

Figure 3.1 Construction of TCP flow

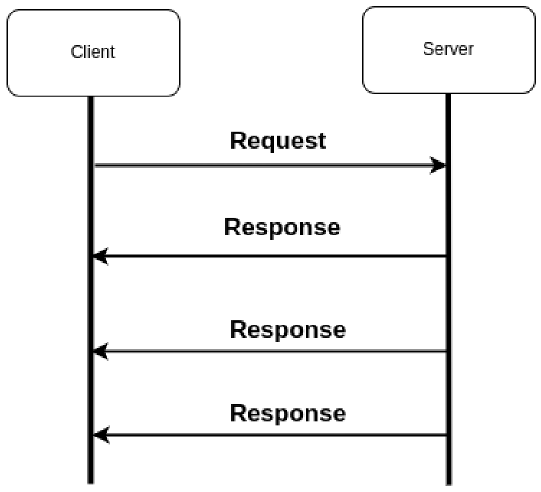

Figure 3.2 Construction of UDP flow

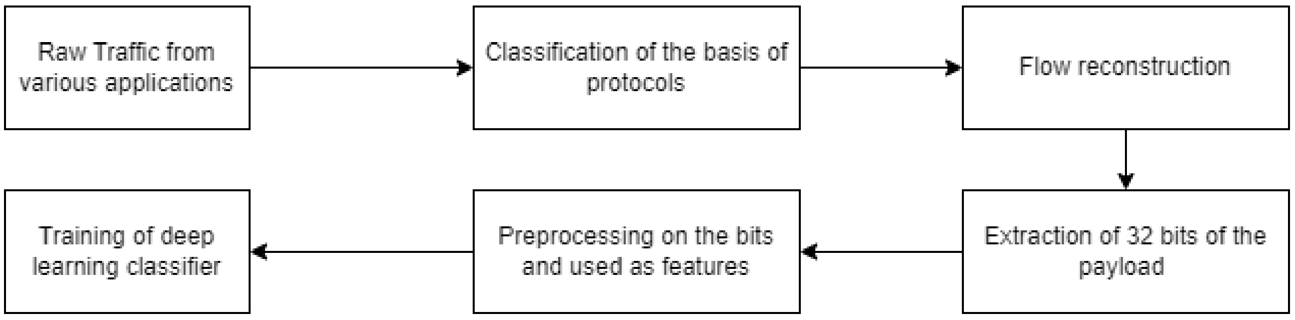
Raw Traffic from various applications
Classification of the basis of protocols
Flow reconstruction
Extraction of 32 bits of the payload
Preprocessing on the bits and used as features
Training of deep learning classifier

Figure 3.3 Architecture diagram for training data

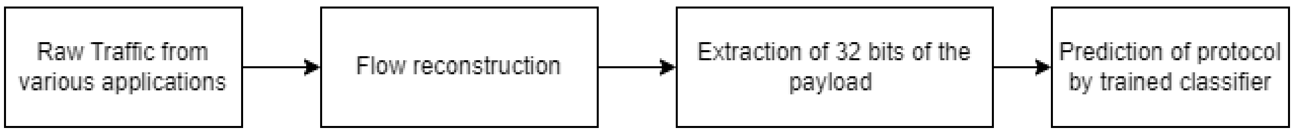
Raw Traffic from various applications
Flow reconstruction
Extraction of 32 bits of the payload
Prediction of protocol by trained classifier

Figure 3.4 Architecture diagram for testing data

3.1.2 Flow Reconstruction

Flow reconstruction refers to the process of reassembling or reconstructing the original data stream or flow from the individual packets that make up the flow.

Flow reconstruction becomes necessary when there is a need to analyze or process the data at the flow level rather than at the packet level. By reconstructing the flow, the original sequence of data can be restored, enabling higher-level analysis and understanding of the transmitted information.

The flow reconstruction process involves identifying packets that belong to the same flow based on their common attributes such as source and destination IP addresses, transport protocol, and port numbers. Once the packets are identified, they are reordered according to their sequence numbers or timestamps to recreate the original data stream.

3.1.3 Payload Bits Extraction

Classification required information of the flows which was provided by the first 32 bits of the payload. As in the contemporary world most of the information is encrypted , only few sections of the payload is accessible.

To achieve this Scapy, a powerful Python library for packet manipulation was used. Flow attributes, such as source and destination IP addresses, transport protocol, and port numbers, are identified to group packets belonging to the same flow. Next, the reconstructed flows was used to extract the bits. Scapy's flexible and intuitive programming interface enables efficient flow reconstruction and facilitates in-depth analysis of network traffic.

3.1.4 Dataset Creation

After extracting the payload bits, we prepared the dataset by labeling each flow with the corresponding network protocol. The dataset prepared consists of flows from 10 different application layer protocols. Further, the dataset was divided into training and testing. Training dataset consists of features from 63204 flows and testing data consists of bits from 63761 flows.

Protocol	**TCP/UDP**	**Flows for Training**	**Size of Training (MB)**	**Flows for Testing**	**Size of Testing (MB)**
BACnet	UDP	9	0.009	11	0.004
BJNP	UDP	34	0.0025	38	0.003
Bootp	UDP	177	0.714	172	4.4
DNS	UDP	59015	19.5	59605	18.1‘
Kerberos	UDP	670	1.6	673	1.9
NBNS	UDP	1272	1.6	1271	8.7
NBSS	TCP	366	2.7	364	3.9
NTP	UDP	403	1.8	401	0.78
QUIC	UDP	126	0.014	92	0.01
SSH	TCP	1132	6.2	1134	6.2

Table 3.1 Description of dataset.

3.2 Training of models

We trained 5 deep learning models for traffic classiffication to get the best performing model.

3.2.1 Convolutional Neural Network (CNN)

It is a type of deep learning network. A CNN is basically made up of several layers which include an input layer, several convolutional layers, pooling layers, fully connected layers, dropout layers, and an output layer.

CNN Architecture

```
CNN = tf.keras.Sequential([
    tf.keras.layers.Conv1D(filters=64, kernel_size=3, activation='relu',
        input_shape=(32, 1)),
    tf.keras.layers.MaxPooling1D(pool_size=2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(32, activation='relu'),
    tf.keras.layers.Dense(10, activation='softmax')
])

CNN.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['
    accuracy'])
```

Explanation of CNN

1. Convolutional 1D Layer: This layer is mainly responsible for feature extraction. The first parameter represents the number of filters in the layer, each filter is responsible for the extraction of a particular feature, this directly means that to capture more complex features we have to keep more filters. However, we must also keep in mind that increasing the number of filters can lead to overfitting, as it increases the number of model parameters and slows down training times. We have set the value of this filter to 64 based on experimentation and observation of an increase in accuracy.

The second parameter is simply the size of the convolutional kernel, which is a small window that moves over the input data to perform the convolution operation. In this case, the kernel size is 3.

The fourth parameter specifies the shape of the input data to the layer. In this case, the input data is a binary string of length 32. The input shape parameter is only used in the first layer of the network, as subsequent layers will automatically infer the input shape from the previous layer's output.

The third parameter is an activation function, which is a mathematical function that produces a non-linear relationship between inputs and outputs, without the activation function model will not be able to learn complex relationships and there will only be a linear relationship between input and output. The activation function used in this layer is the rectified linear unit (ReLU).

Relu is a popular activation function used in neural networks. It helps to solve the vanishing gradient problem by providing a non-linear activation function that outputs 0 for negative input values and the input value for positive input values. As the gradients are propagated back through the layers of the neural network, they can become very small this is called a vanishing gradient problem. Due to this problem, the weights in the earlier layers of the network are not updated effectively, and the network may fail to learn important features of the data.

The ReLU function maps any input to 0 if it's negative, or the input itself if it's positive. Mathematically, it can be expressed as:

$$\text{ReLU}(x) = \max(0, x) \tag{3.1}$$

2.MaxPooling Layer: It is used to grab the most important information from the feature maps generated from the convolutional layer. Input binary string is divided into non-overlapping regions and the size of this non-overlapping region is specified by the pool-size parameter which in this case we have taken 2. Now the maximum value within each region is taken as the output.

3.Flatten Layer: This layer flattens the output from the previous layer into a one-dimensional array, which is then passed on to the next dense layer.

4 .Dense Layer: Is a fully connected layer that has 32 neurons with ReLU activation function. It takes the output of the Flatten layer as input, which is a flattened array of the feature maps produced by the convolutional and pooling layers. The Dense layer applies a matrix multiplication operation to the input followed by a bias addition, and then applies the ReLU activation function to each neuron's output. The output of this layer is a feature vector with 32 dimensions.

Finally, the output of this layer is passed through another Dense layer with 10 neurons and a softmax activation function, which produces a probability distribution over the 10 possible classes.

Softmax function is commonly used in multi-class classification problems, where the goal is to predict the probability of an input belonging to each of several classes. By using the softmax activation function in the output layer, the CNN model is able to output class probabilities that can be interpreted and compared directly, making it easier to evaluate the model's performance and make decisions based on its predictions.

We compiled the model using the Adam optimizer because it performs well on large datasets like ours. Adam uses a dynamic learning rate that adjusts for each weight during training based on the average of the previous gradients. This feature helps the optimizer to converge faster and more accurately than other optimizers. We used the categorical cross-entropy loss function because our problem involves classification, and this loss function is known to perform well in such scenarios.

3.2.2 Recurrent Neural Networks (RNN)

Recurrent Neural Networks (RNNs) are a type of neural network architecture that are designed for processing sequential data. Unlike feedforward neural networks, which process inputs in a fixed order and do not have any memory of previous inputs, RNNs can maintain an internal state that allows them to capture temporal dependencies in the input sequence.

RNN Architecture

```
RNN = tf.keras.Sequential([

    tf.keras.layers.Embedding(input_dim=2, output_dim=32, input_length=32)

    tf.keras.layers.SimpleRNN(32),
    tf.keras.layers.Dense(10, activation='softmax')
])
```

Explaination of RNN Model

1. Embedding Layer: The Embedding layer in the RNN model is used to learn a dense representation of the input sequence. It takes an integer-encoded input sequence and maps each integer to a corresponding dense vector of specified dimensions.

As the integer sequence in our data in binary the input dimension is 2. The output dimension of the embedding layer is 32, which means that each input token is mapped to a dense vector of length 32. The input length of the layer is also 32, which means that the input sequence has 32 time steps.

2.RNN Layer: This is a recurrent layer that processes the sequence of dense vectors produced by the Embedding layer in a sequential manner. The SimpleRNN layer used has 32 units, which means that it maintains an internal state vector of length 32 that is updated at each time step. The SimpleRNN layer applies the same set of weights at each time step to the current input and the previous state, producing a new state vector. This allows it to capture temporal dependencies in the input sequence.

Finally, the output of RNN layer is passed through another Dense layer with 10 neurons and a softmax activation function, which produces a probability distribution over the 10 possible classes.

3.2.3 Long Short-Term Memory (LSTM)

It is a type of Recurrent neural network that is able to process information that is presented in a sequence. Standard RNNs have the problem of retaining important information over long periods of time, which can limit their ability to effectively process and classify sequential data.LSTM solves this problem by allowing the network to forget or retain information of past time steps based on the current input.

LSTM Architecture

```
LSTM = tf.keras.Sequential([

    tf.keras.layers.Embedding(input_dim=2, output_dim=32, input_length=32)

    tf.keras.layers.LSTM(32),

    tf.keras.layers.Dense(10, activation='softmax')

])
```

Explaination of LSTM Model

1. Embedding Layer: The Embedding layer in the LSTM model is used to learn a dense representation of the input sequence. It takes an integer-encoded input sequence and maps each integer to a corresponding dense vector of specified dimensions.

As the integer sequence in our data in binary the input dimension is 2. The output dimension of the embedding layer is 32, which means that each input token is mapped to a dense vector of length 32. The input length of the layer is also 32, which means that the input sequence has 32 time steps.

2.LSTM Layer: The LSTM layer is a type of recurrent neural network (RNN) layer that processes the input sequence in a sequential manner, while maintaining a memory of previous inputs. The output of the Embedding layer is fed into the LSTM layer, which applies a series of nonlinear transformations to the input at each time step, producing an output vector at each step. The output of the last time step is used as the output of the LSTM layer.

The 32 parameter in this layer is the number of output nodes in the layer, which can be changed based on the complexity of the problem being solved. Each output node corresponds to a different feature or pattern of the input sequence that the

LSTM layer has learned to recognize and extract. Increasing the number of output nodes can increase the model's capability to capture complex patterns in the input sequence, but it can also lead to overfitting if the model becomes too complex.

Finally, the output of LSTM layer is passed through another Dense layer with 10 neurons and a softmax activation function, which produces a probability distribution over the 10 possible classes.

3.2.4 Gated Recurrent Unit (GRU)

GRU is similar to the LSTM (Long Short-Term Memory) architecture because it also uses gating mechanisms to allow the network to selectively retain or update and forget information with the passage of time. However, unlike LSTM, GRU has only two gates - an update gate and a reset gate - which reduces the complexity of the architecture and reduces the number of parameters to be learned.

GRU Architecture

```
GRU = tf.keras.Sequential([
    tf.keras.layers.Embedding(input_dim=2, output_dim=32, input_length=32)
    tf.keras.layers.GRU(32),
    tf.keras.layers.Dense(10, activation='softmax')
])
```

Explaination of GRU Model

1. Embedding Layer: The Embedding layer in the GRU model is used to learn a dense representation of the input sequence. It takes an integer-encoded input sequence and maps each integer to a corresponding dense vector of specified dimensions.

As the integer sequence in our data in binary the input dimension is 2. The output dimension of the embedding layer is 32, which means that each input token is mapped to a dense vector of length 32. The input length of the layer is also 32, which means that the input sequence has 32 time steps.

2.GRU Layer: The GRU layer is a recurrent layer that takes the sequence of dense vectors produced by the Embedding layer and processes them in a sequential manner. The GRU layer has 32 units, which means that it maintains an internal state vector of length 32 that is updated at each time step. The GRU layer uses gating mechanisms to selectively update and forget information from the internal state vector, allowing it to capture long-term dependencies in the input sequence.

Finally, the output of GRU layer is passed through another Dense layer with 10 neurons and a softmax activation function, which produces a probability distribution over the 10 possible classes.

3.2.5 Artificial Neural Networks (ANN)

An ANN consists of layers of interconnected nodes, or neurons, that process input data and produce output predictions. The basic building block of an ANN is a neuron, which takes one or more inputs, applies a linear transformation to those inputs, and then applies a non-linear activation function to the result. The output of the neuron is then passed on to other neurons in the network, forming a chain of interconnected computations. It is the simplest model among the models that were trained.

ANN Architecture

```
ANN = tf.keras.Sequential([
    tf.keras.layers.Dense(64, activation='relu', input_shape=(32,)),
    tf.keras.layers.Dense(32, activation='relu'),
    tf.keras.layers.Dense(10, activation='softmax')
])
```

Explaination of ANN Model

The first layer is a dense layer with 64 units and a ReLU activation function. The input shape is set to (32,), as the input to the model is a 1D tensor of 32 length. This layer performs a linear transformation of the input followed by a rectified linear unit (ReLU) activation function, which introduces non-linearity into the model.

The second layer is also a dense layer with 32 units and a ReLU activation function. This layer takes the output of the previous layer as input and performs another linear transformation followed by a ReLU activation.

The third and final layer is a dense layer with 10 units and a softmax activation function. This layer takes the output of the previous layer and produces a probability distribution over 10 classes. The class with the highest probability is taken as the predicted class for the input.

Chapter 4

Experimental Results

We evaluated the trained models to determine whether they are efficient in classifying network traffic or not. For all the trained models we calculated the Recall Rate to evaluate the clasification accuracy for each class.

$$\mathrm{R}ecall = TP/(TP + FN) \tag{4.1}$$

where for each category, TPs (True Positives) are the flows correctly classified, FNs (False Negatives) are the flows incorrectly rejected.

Results showed that CNN and ANN perform better than other models for classifying network traffic with overall accuracy scores of 99.92 % and 99.97 % respectively and classified some protocols like Bacnet, BJNP,NTP,SSH perfectly .

For each model we have also shown confusion matrix. In each bracket of matrix count of testing flows is mentioned. Example if BACnet has 11 testing flows and they are 100 % matched with BACnet than 11 was written in BACnet and 0 in remaining cells. If BJNP has 15 flows and out of them 2 testing flows are mismatched with DNS then 13 was wriiten in (BJNP to BJNP) cell and 2 in (BJNP to DNS) cell.

	BACnet (11)	**BJNP (38)**	**Bootp (172)**	**DNS (59605)**	**Kerberos (673)**	**NBNS (1271)**	**NBSS (364)**	**NTP (401)**	**QUIC (92)**	**SSH (1134)**
BACnet	11	0	0	0	0	0	0	0	0	0
BJNP	0	38	0	0	0	0	0	0	0	0
Bootp	0	0	172	0	0	0	0	0	0	0
DNS	0	0	0	59605	0	8	0	0	6	0
Kerberos	0	0	0	0	670	1	2	0	0	0
NBNS	0	0	0	0	0	1262	0	0	4	0
NBSS	0	0	0	0	0	0	362	0	0	0
NTP	0	0	0	0	0	0	0	401	0	0
QUIC	0	0	0	0	0	0	0	0	86	0
SSH	0	0	0	0	0	0	0	0	0	1134

Table 4.1 Confusion Matrix of CNN.

Protocol	**Recall Rate (%)**
BACnet	100
BJNP	100
Bootp	100
DNS	100
Kerberos	99.55
NBNS	99.29
NBSS	99.45
NTP	100
QUIC	93.47
SSH	100

Table 4.2 Recall of CNN.

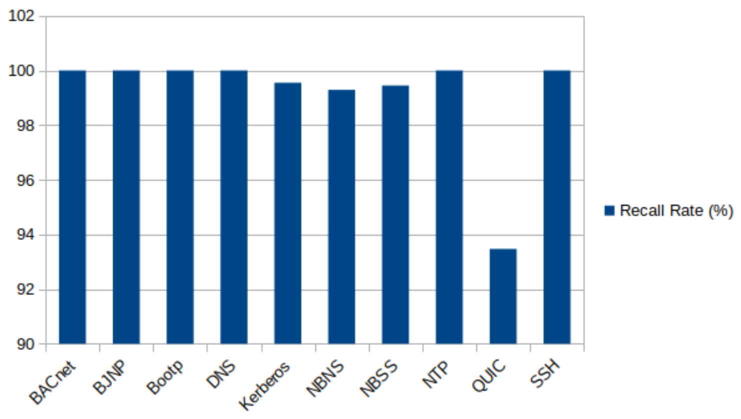

Figure 4.1 Recall of CNN

	BACnet (11)	**BJNP (38)**	**Bootp (172)**	**DNS (59605)**	**Kerberos (673)**	**NBNS (1271)**	**NBSS (364)**	**NTP (401)**	**QUIC (92)**	**SSH (1134)**
BACnet	5	0	0	0	0	0	0	0	0	0
BJNP	0	38	0	0	0	0	0	0	1	0
Bootp	0	0	172	0	0	0	0	0	0	0
DNS	0	0	0	59602	14	8	0	0	5	0
Kerberos	0	0	0	0	364	0	0	0	1	0
NBNS	6	0	0	3	1	1263	0	0	4	0
NBSS	0	0	0	0	4	0	359	0	1	0
NTP	0	0	0	0	207	0	3	401	12	0
QUIC	0	0	0	0	83	0	2	0	68	0
SSH	0	0	0	0	0	0	0	0	0	1134

Table 4.3 Confusion Matrix of RNN.

Protocol	**Recall Rate (%)**
BACnet	45.45
BJNP	100
Bootp	100
DNS	99.99
Kerberos	54.08
NBNS	99.37
NBSS	98.62
NTP	100
QUIC	73.91
SSH	100

Table 4.4 Recall of RNN

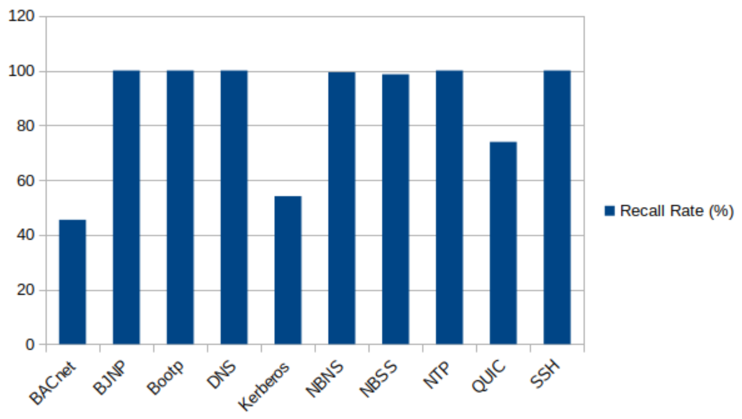

Figure 4.2 Recall of RNN

	BACnet (11)	**BJNP (38)**	**Bootp (172)**	**DNS (59605)**	**Kerberos (673)**	**NBNS (1271)**	**NBSS (364)**	**NTP (401)**	**QUIC (92)**	**SSH (1134)**
BACnet	11	0	0	0	0	0	0	0	0	0
BJNP	0	38	0	0	0	5	0	0	0	0
Bootp	0	0	172	0	0	0	0	0	0	0
DNS	0	0	0	59601	0	29	0	0	10	0
Kerberos	0	0	0	4	668	0	3	0	9	0
NBNS	0	0	0	0	0	1233	0	0	1	0
NBSS	0	0	0	0	5	3	360	0	0	0
NTP	0	0	0	0	0	0	1	401	1	0
QUIC	0	0	0	0	0	1	0	0	71	0
SSH	0	0	0	0	0	0	0	0	0	1134

Table 4.5 Confusion Matrix of LSTM.

Protocol	**Recall Rate (%)**
BACnet	100
BJNP	100
Bootp	100
DNS	99.99
Kerberos	99.25
NBNS	97.01
NBSS	98.9
NTP	100
QUIC	77.17
SSH	100

Table 4.6 Recall of LSTM

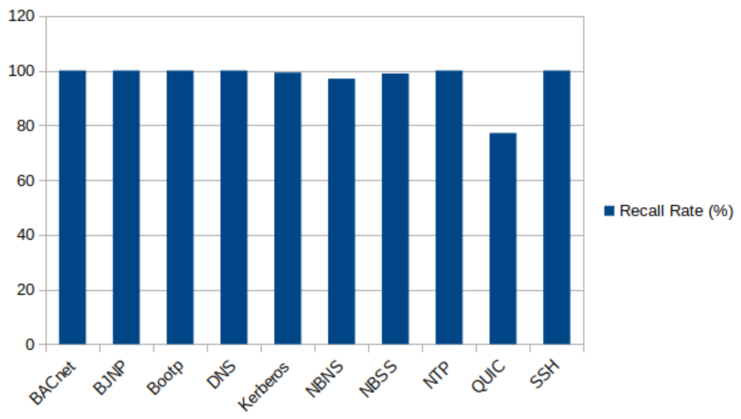

Figure 4.3 Recall of LSTM

	BACnet (11)	**BJNP (38)**	**Bootp (172)**	**DNS (59605)**	**Kerberos (673)**	**NBNS (1271)**	**NBSS (364)**	**NTP (401)**	**QUIC (92)**	**SSH (1134)**
BACnet	11	0	0	0	0	0	0	0	0	0
BJNP	0	38	0	0	0	2	0	0	0	0
Bootp	0	0	172	0	0	0	0	0	0	0
DNS	0	0	0	59602	0	21	1	0	3	0
Kerberos	0	0	0	1	670	7	3	0	6	0
NBNS	0	0	0	1	0	1240	0	0	4	0
NBSS	0	0	0	0	3	1	360	0	0	0
NTP	0	0	0	0	0	0	0	401	0	0
QUIC	0	0	0	1	0	0	0	0	79	0
SSH	0	0	0	0	0	0	0	0	0	1134

Table 4.7 Confusion Matrix of GRU.

Protocol	**Recall Rate (%)**
BACnet	100
BJNP	100
Bootp	100
DNS	99.99
Kerberos	99.55
NBNS	97.56
NBSS	98.9
NTP	100
QUIC	85.86
SSH	100

Table 4.8 Recall of GRU

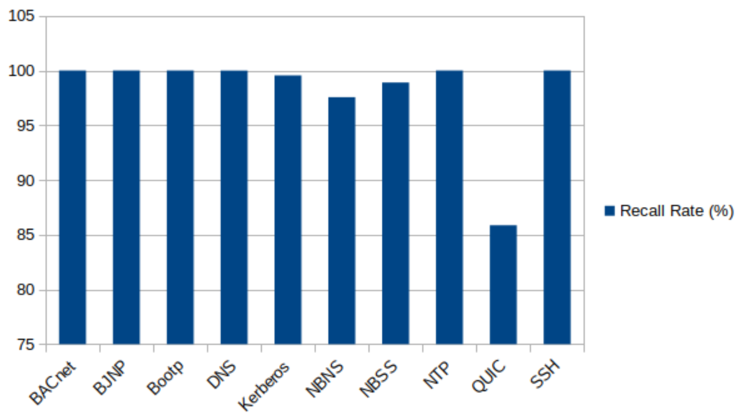

Figure 4.4 Recall of GRU

	BACnet (11)	**BJNP (38)**	**Bootp (172)**	**DNS (59605)**	**Kerberos (673)**	**NBNS (1271)**	**NBSS (364)**	**NTP (401)**	**QUIC (92)**	**SSH (1134)**
BACnet	11	0	0	0	0	0	0	0	0	0
BJNP	0	38	0	0	0	0	0	0	0	0
Bootp	0	0	172	0	0	0	0	0	0	0
DNS	0	0	0	59605	0	12	0	0	0	0
Kerberos	0	0	0	0	672	0	2	0	0	0
NBNS	0	0	0	0	0	1259	0	0	0	0
NBSS	0	0	0	0	1	0	362	0	0	0
NTP	0	0	0	0	0	0	0	401	1	0
QUIC	0	0	0	0	0	0	0	0	91	0
SSH	0	0	0	0	0	0	0	0	0	1134

Table 4.9 Confusion Matrix of ANN.

Protocol	**Recall Rate (%)**
BACnet	100
BJNP	100
Bootp	100
DNS	100
Kerberos	99.85
NBNS	99.05
NBSS	99.45
NTP	100
QUIC	98.91
SSH	100

Table 4.10 Recall of ANN

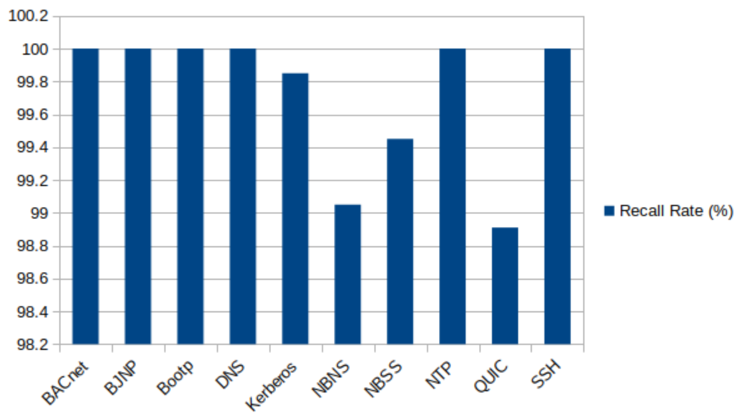

Figure 4.5 Recall of ANN

Chapter 5

Conclusion and Future Work

This research demonstrates the effectiveness of deep learning models in accurately classifying network protocols using payload features. The process of flow reconstruction and payload extraction proved crucial in preparing the data for training. The results highlight the potential of deep learning models in network traffic analysis and classification tasks.

Future work could be expanding the dataset to include a wider range of protocols and a larger volume of traffic samples can enhance the capability of the models. Also, integrating additional information, such as packet timing or flow-based features, along with the extracted payload features could improve the models' ability to discriminate between protocols with similar payload patterns. This could enhance the overall classification accuracy and address any limitations associated with payload-based classification alone.

In conclusion, this project successfully implemented deep learning models to classify network protocols based on extracted payload features. The achieved accuracy rate demonstrates the models' ability to effectively capture patterns in the payload data. The findings contribute to the field of network management and security, providing insights for improving protocol classification techniques in real-world scenarios.

Bibliography

[1] Zhao, L., Cai, L., Yu, A., Xu, Z. and Meng, D., 2020, March. A novel network traffic classification approach via discriminative feature learning. In Proceedings of the 35th annual ACM symposium on applied computing (pp. 1026-1033).

[2] Khandait, P., Hubballi, N. and Mazumdar, B., 2020, January. Efficient keyword matching for deep packet inspection based network traffic classification. In 2020 International Conference on COMmunication Systems NETworkS (COMSNETS) (pp. 567-570). IEEE.

[3] Rezaei, S. and Liu, X., 2020, August. Multitask learning for network traffic classification. In 2020 29th International Conference on Computer Communications and Networks (ICCCN) (pp. 1-9). IEEE.

[4] D'Angelo, G. and Palmieri, F., 2021. Network traffic classification using deep convolutional recurrent autoencoder neural networks for spatial–temporal features extraction. Journal of Network and Computer Applications, 173, p.102890.

[5] Xie, G., Li, Q., Jiang, Y., Dai, T., Shen, G., Li, R., Sinnott, R. and Xia, S., 2020, August. Sam: Self-attention based deep learning method for online traffic classification. In Proceedings of the Workshop on Network Meets AI ML (pp. 14-20).

[6] Ren, X., Gu, H. and Wei, W., 2021. Tree-RNN: Tree structural recurrent neural network for network traffic classification. Expert Systems with Applications, 167, p.114363.

[7] Mao, K., Xiao, X., Hu, G., Luo, X., Zhang, B. and Xia, S., 2021, June. Byte-Label Joint Attention Learning for Packet-grained Network Traffic Classification. In 2021 IEEE/ACM 29th International Symposium on Quality of Service (IWQOS) (pp. 1-10). IEEE.

[8] Sarhangian, F., Kashef, R. and Jaseemuddin, M., 2021, April. Efficient traffic classification using hybrid deep learning. In 2021 IEEE International Systems Conference (SysCon) (pp. 1-8). IEEE.

[9] Alam, F., Kashef, R. and Jaseemuddin, M., 2021, April. Enhancing The Performance of Network Traffic Classification Methods Using Efficient Feature Selection Models. In 2021 IEEE International Systems Conference (SysCon) (pp. 1-6). IEEE.

[10] Jin, Z., Liang, Z., Wang, Y. and Meng, W., 2021. Mobile network traffic pattern classification with incomplete a priori information. Computer Communications, 166, pp.262-270.

[11] Dong, S., 2021. Multi class SVM algorithm with active learning for network traffic classification. Expert Systems with Applications, 176, p.114885.

[12] Eom, W.J., Song, Y.J., Park, C.H., Kim, J.K., Kim, G.H. and Cho, Y.Z., 2021, April. Network traffic classification using ensemble learning in software-defined networks. In 2021 International Conference on Artificial Intelligence in Information and Communication (ICAIIC) (pp. 089-092). IEEE.