SIMPLIFYING AI-SUPPORTED DEVELOPMENT FOR NETWORKING AND COMMUNICATION SYSTEMS

Dissertation

Submitted to

The School of Engineering of the

UNIVERSITY OF DAYTON

In Partial Fulfillment of the Requirements for

The Degree of

Doctor of Philosophy in Engineering

By

Fuhao Li

Dayton, Ohio

August, 2022

SIMPLIFYING AI-SUPPORTED DEVELOPMENT FOR NETWORKING AND COMMUNICATION SYSTEMS

Name: Li, Fuhao

APPROVED BY:

Feng Ye, Ph.D. Advisory Committee Chairman Assistant Professor, Electrical and Computer Engineering	Vijayan Asari., Ph.D. Committee Member Professor, Electrical and Computer Engineering
John Loomis, Ph.D. Committee Member Professor, Electrical and Computer Engineering	Muhammad Usman, Ph.D. Committee Member Professor, Mathematics
Robert J. Wilkens, Ph.D., P.E. Associate Dean for Research and Innovation Professor School of Engineering	Margaret F. Pinnell, Ph.D. Interim Dean, School of Engineering

ABSTRACT

SIMPLIFYING AI-SUPPORTED DEVELOPMENT FOR NETWORKING AND COMMUNICATION SYSTEMS

Name: Li, Fuhao
University of Dayton

Advisor: Dr. Feng Ye

Artificial Intelligence (AI)-based algorithm have demonstrated its robust capability to support in networking and communication systems, such as network traffic classifier (NTC), intrusion detection system, channel state information processing in massive multiple input multiple output (MIMO) wireless communication systems, etc. However, due to the relatively high-dimensional data input and limited computing resources, many of the existing AI implementations are too complicated for efficient processing in networking and communication systems. To address this issue, this dissertation explores a systematic approach that simplifies the AI-supported implementation for multiple networking and communication systems. The proposed approaches mainly evaluate the structure of AI implementations in different scenarios. In specific, for an an AI-supported NTC development, an input feature contribution extraction scheme is developed to weigh each input feature based on both the significance and the uniqueness of the corresponding feature. The optimal set of input features are determined to minimize the complexity of a targeting AI-based NTC while maintaining high performance in classification. Moreover, an autonomous update scheme is proposed to detect the changes in feature contribution and process updates. Evaluations on two fundamental AI-based classifiers, demonstrated that the proposed scheme can significantly reduce the input features and accelerate NTC models by one to two magnitudes

while maintaining high accuracy. The proposed autonomous update scheme can accurately detect a change in feature contributions and update the NTC models to sustain the high accuracy. In addition, we further developed an adaptive pruning for MLP-based NTC to fit the different requirements of NTC due to network congestion. The results demonstrated that the lossless optimization and adaptive pruned network traffic classifier accelerate the baseline MLP based NTC models by about 5 to 10 times based on the requirements. Besides NTC, this dissertation also developed a model simplification scheme targeting the deep learning based massive MIMO CSI feedback process in the next generation wireless communication systems. In this part, a dynamic channel sparsity scheme is proposed to optimize the optimized structure of the compact network. The lesser important channels and nodes are removed after the process while for sustaining the reconstruction performance. Two popular deep learning based CSI feedback models are developed for evaluations. The results demonstrated that the proposed method can accelerate the baseline models by up to 3 times. Furthermore, this dissertation proposed a lossless optimization and simplification scheme. Using the MLP-based NTC as an example, the developed approach is to remove the nodes that contribute the least to the classification result in the hidden layer with pruning method. Comparing with the conventional pruning method, the proposed scheme can reduce the computational complexity while ensuing the accuracy without retraining and fine-tuning. This dissertation have demonstrated that the current AI implementation in networking and communication systems can be simplified for high efficiency. The results have laid a solid foundation for future research in lightweight AI not only for the studied systems, but also for a broader area that have limited computing resources and power supply.

For the people who helped me in my life

ACKNOWLEDGMENTS

Finally it is time for me to acknowledge all those who have inspired me, supported me and helped me to get to the place where I am today. Firstly, I would like to thank my advisor Dr. Feng Ye for providing me the support and motivation to do my job well. His advice will always be valuable to me. I also want to thank Dr. John Lommis, Dr. Vijayan Asari and Dr. Muhammad Usman for introducing me to my research area and guiding me through it. Without them, I would not have accomplished this dissertation. They were always there in times of need with their valuable suggestions. Before moving on, I would also like to thank the department of electrical and computer engineering at University of Dayton for giving me this opportunity to realize my dream. I have a lot of happy memories from here. No words would be enough for my parents to describe their love and support for me. Without my mother I would not have been the person I am today. I would like to acknowledge Qing Shen who is my girlfriend that have been my pillars of support throughout my graduate study. Her constant words of support and inspiration helped me breeze through tough times and part of my success definitely belongs to her. I I want to thank her for all her constructive criticisms, her love and unconditional support that helped me to be the person I am today, both personally and professionally. I also have to acknowledge Dingnan Zhang who is my roommate that knowingly and unknowingly provided me with the much needed humor in stressful times in all these years. I also want to mention my labmates Jielun Zhang, Daidong Ying, Jiahui Yu, Venkat Kumar. Working with all of them has been a pleasure. My graduate studies at UD paved the way for some very important friendships in my life. Finally, I would like to thank all my committee members for providing me with their valuable feedback on my thesis and for working with me under severe time constraints. Overall, it has been a rewarding experience that I will cherish for a long time.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER I

INTRODUCTION

The networking and communication systems are becoming ubiquitous for everyone in the world. The development of the next-generation networking and communication systems is critical. The Hootsuite report in 2022 mentioned that the growth of social media users continuing to trend upwards. There are now 4.62 billion social media users around the world, representing growth of more than 10% of new users since this time last year. The number of social media users is now equivalent to more than 58% of the world's total population [1]. To better serve the rapidly growing network users, new concept in networking and communication systems such as the fifth-generation (5G) and beyond mobile systems, Internet of Things, etc., are being developed. In this chapter, we will briefly introduce the background of the next-generation networking and communication systems, artificial intelligence (AI)-supported development in this field and motivations of this dissertation.

1.1 Next-Generation Networking and Communication Systems

Several concepts have been proposed for the next-generation networking and communication systems, e.g., 5G/6G, Internet of Things (IoT), Internet of Vehicle (IoV), etc. For example, 5G and beyond can provide higher capacity, higher data rate, lower end to end latency, massive device connectivity, reduced cost and consistent quality of experience provisioning [2]. In addition, compared to the fourth generation (4G) mobile systems, 5G can support 10 to 100 times more connected devices, 1000 times higher mobile data volume per area, 10 to 100 times higher data rate, ultra-low latency, 99.99% availability, 100% coverage, etc. [3]. In this case, new architectures, methodologies, and technologies are needed to support and achieve the expected performance of 5G and beyond mobile systems.

Device-to-device (D2D) communication is proposed to not only increase spectrum efficiency but also improve throughput, energy efficiency, delay, and fairness for mobile networks [4]. Massive multiple-input and multiple-output (mMIMO) can increase the system communication capacity without increasing spectrum resources and antenna transmit power for better supporting 5G networks [5]. Millimeter wave (mmWave) enables wide much wider spectrum availability and higher spectrum efficiency for 5G and beyond 5G communication systems [6]. Software-defined networks (SDN), network function virtualization (NFV) are proposed to optimize the distribution of system workload via powerful control panel and networking slicing [7, 8].

1.2 AI-Supported Development for Networking and Communication Systems.

Recently, the AI algorithms have demonstrated its robust capability to solve problem in networking and communication, such as network traffic classifier, intrusion detection system, and massive MIMO CSI feedback process, etc [9–11]. AI Algorithms, especially the deep learning based methods, can adaptively extract deep features from network traffic, avoiding a series of complex operations such as feature engineering. For example, the deep learning based network traffic classifiers, e.g., CNN based and MLP based, etc., can accurately classify both clear and encrypted data traffic for providing better network measurement and management in 5G [12, 13]. The Internet service providers or network operators can manage the overall performance of a network by analyzing and identify different types of applications flowing in a network [14]. The deep learning based intrusion detection systems can accurately monitor and distinguish the coming packets as normal network traffic or cyberattacks to ensure the security in next-generation networking and communication systems. In addition, AI-based CSI feedback models slove the problem of the rapid increase

of feedback overhead in massive MIMO systems, and recover CSI with greatly improved reconstruction quality compared with traditional methods in massive MIMO systems.

Although the existing DL-based NTCs and massive MIMO CSI feedback models show the high classification and reconstruction performance, they rely on neural network models that consist of numerous parameters, where the computation efficiency cannot be guaranteed. It takes both time and storage space for the neural networks to perform NTC and MIMO CSI feedback models. AI based NTC require a lot of computational resources for the internal matrix operations and non-linear activation among the network parameters, the complex network architectures further increase the number of network parameters. For instance, one reason for designing such complex neural network structure is that the existing AI-based NTC models apply full data packets inputs with padding, which result in complex NTC designs. Such a complex AI based AI-based NTC can be hardly implemented on energy constrained networking edge devices with limited computing capability, e.g, Wi-Fi routers, IoT devices, etc. As a result, these DL-based NTC models are not piratical to be deployed in the gateways or host devices with low computational profile, e.g., home Wi-Fi routers, access points, smart phones, etc. Therefore, this dissertation aims to simplify AI-supported development for the next-generation networking and communication systems.

1.3 Structure of the Dissertation.

This rest of this dissertation includes 4 chapters, described in the following:

Chapter II: This chapter presents a background study of the different types of deep learning based system in networking and communication systems, e.g., network traffic classifier, intrusion detection system, etc. Then it discusses the existing algorithms and techniques ,e.g., data mining, feature selection, etc.

Chapter III: This chapter presents an adaptive and lightweight NTC framework. To be specific, an input feature contribution extraction scheme is design to calculate the contribution of each input feature based on both the significance and the uniqueness of the corresponding feature. The optimal set of input features are determined to minimize the complexity of a targeting AI-based NTC without sacrificing performance in classification. Moreover, an autonomous update scheme is proposed to detect the changes in feature contribution and process updates to sustain the classification performance.

Chapter IV: This chapter presents a lossless optimization for a popular AI-based NTC, i.e., Multilayer Perceptron (MLP) based NTC to remove the nodes that contribute the least to the classification result in the hidden layer with pruning method, thus speed up the NTC to be deployed on those IoT devices. In addition, an adaptive pruning for MLP-based NTC is further proposed to fit the different requirements of NTC on operating speed to solve the network problems, e.g, network congestion, waste of resources.

Chapter V: This chapter presents a model simplification scheme for deep learning based massive MIMO CSI feedback model. An dynamic channel sparsity scheme is proposed for automatically deciding the optimized structure of the compact network. After training, some unimportant channels and nodes are removed and a fine tuning is applied for sustaining the reconstruction performance.

Chapter VI: This chapter concludes the dissertation and introduce the future work.

CHAPTER II

BACKGROUND AND RELATED WORK

The proposed simplification schemes are mainly applied to networking traffic classification (NTC) and channel state information processing in this dissertation. Note that the proposed schemes can be extended to other AI-supported networking and communication systems, e.g., intrusion detection system, spectrum management, etc., straightforwardly.

2.1 Network Traffic Classification

Network traffic classification can identify different applications and protocols that exist in a network [15]. Accurate network traffic classification is an important element of network measurement and management such as flow prioritization, bandwidth allocation, etc.

2.1.1 Traditional Approaches for Network Traffic Classification

The traditional approaches of network traffic classification include port-based, payload-based and statistical approaches. The port-based approach is a simple and fast way to classify network traffic. The association of the ports in the TCP/UDP header is used to compare with the well-known TCP/UDP port number assigned by IANA [16]. For example, ports 21 and 22 are for FTP and SSH, respectively. However, due to the dynamic port and tunnels, many network traffic cannot be classified by the port-based approaches. The payload-based approaches classify the packets by inspecting headers and payload information. DPI provides high classification accuracy, however, at the cost of high computational complexity. Moreover, it is labor intensive to keep a packet signature library up to date. Due to the increasing implementation of network traffic encryption, the payload-based approach becomes less useful. Statistical approaches have been used to classify encrypted network

traffic. The traditional approaches use the traffic statistical features such as the packet length, inter-arrival time, etc, to classify network traffic [17]. Machine learning approaches such as support vector machine (SVM), decision tree, principal component analysis (PCA) and k-nearest neighbors (KNN) are always applied with the statistical NTCs for classifying the encrypted network traffic [18–22]. For example, the authors in [23] proposed an SVM based based network traffic classifier, which achieved an average of accuracy of 88% in the testing environment. The authors in [19] proposed a PCA based network traffic classifier. Fast correlation-based filter algorithm is used first to filter data for suitable flow attributes. Then the PCA processes the flow attributes to build features subspace for each category. A k-nearest neighbor approach is then used to classify the traffic samples. The proposed approach achieved an accuracy around 90%. In [24], the authors proposed a network traffic classifier based on several algorithms, e.g., KNN, C4.5 decision tree, naive bayes, flexible naive bayes, which can achieve an average accuracy of 88.9%, 88.6%, 82% and 76.4%, respectively. Despite of the good performance in the testing environment, the classification performance highly depends on the manually selected features [25, 26].

2.1.2 Deep Learning based NTC

Recently, many researchers turn to use deep learning approaches, e.g., Mutilayer Perceptron (MLP), Convolutional Neural Network (CNN), etc., for NTC development [27–29]. Compared to the traditional approaches, deep learning based NTCs are superior in processing encrypted network traffic.

2.1.2.1 MLP based NTC

MLP has been applied to many complex tasks such as image classification, speech recognition, recommendation system, etc. [30]. An MLP based NTC consists of three parts: input

layer, one or more hidden layers and output layer, as illustrated in Fig. 2.1. The input to the classifier is the data packets after pre-processing, e.g., truncating, padding, header removing, normalization, etc. Multiple hidden layers are applied for transferring the information of features after the input layer. For the j-th hidden layer, the output is computed as:

$$\mathbf{z}_j = (W \cdot \mathbf{z}_{j-1}) + b, \tag{2.1}$$

where W is matrix of weights; z_j is the output of the j-th layer, and b is the bias. An activation function is applied after each hidden layer for non-linearity. For instance, Rectified Linear Units (ReLU) is widely used as an activation function for the hidden layers in a MLP based NTC. It reduces the likelihood of the gradient vanishing problem compared with other activation functions such as sigmoid and tanh functions) [31]. The ReLU function is

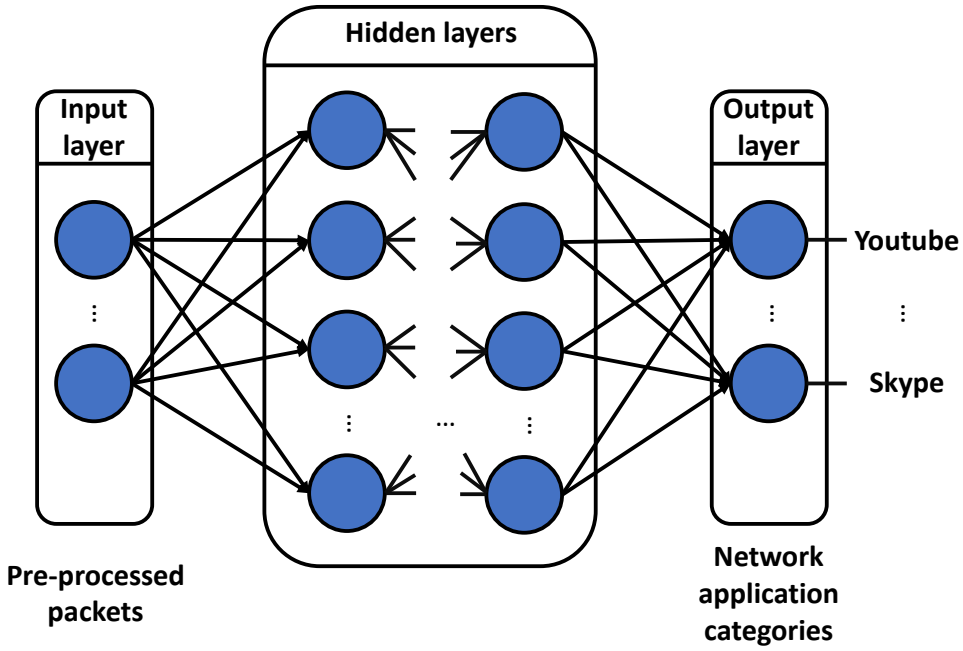

Figure 2.1: MLP based NTC.

computed as:

$$\sigma(\mathbf{z}_j) = \max[0, \mathbf{z}_\mathrm{j}], \tag{2.2}$$

After the hidden layers, a softmax process and an output layer is to classify the samples. The softmax function is used as the activation function that output a multinomial probability distribution, computed as [31]:

$$\mathbf{s} = [s_1, s_2, ..., s_N] = \frac{\exp(z_n)}{\sum_{i=1}^{N} \exp(z_i)}, \tag{2.3}$$

where z is the output of the last dense layer that connected to the Softmax layer; $s_n \leq 1$ and $\sum s_n = 1$ is the categorical probability for the input to be classified into the class n. The MLP based NTC is trained via back propagation. The back propagation uses the output error to adjust the weights of the neural network. MLP can be found widely applied to NTC development [17, 32]. For example, the proposed MLP based NTC achieved an accuracy of 88% [17]. The authors in [32] proposed an effective NTC based on MLP to classify the target traffic into different categories, which achieved an accuracy of around 99%.

2.1.2.2 CNN based NTC

CNN is another popular deep learning architecture used for image classification [31]. Recently, CNN has been introduced to NTC development. A basic CNN based NTC typically consists of convolutional layers, pooling layers, dense/fullyconnected layers, softmax and output layer, as illustrated in Fig. 2.2. The convolutional layers perform the feature extraction by convolution kernels. A data traffic input is usually processed to a 2-dimentional (2-D) matrix. Each input is processed by convolutional kernels in each convolutional layer as follows:

$$z_{k,i,j} = b_k + \sum_{l} \sum_{m=1}^{W} \sum_{n=1}^{H} w_{k,l,m,n} * z_{l,i+m-1,j+n-1}, \tag{2.4}$$

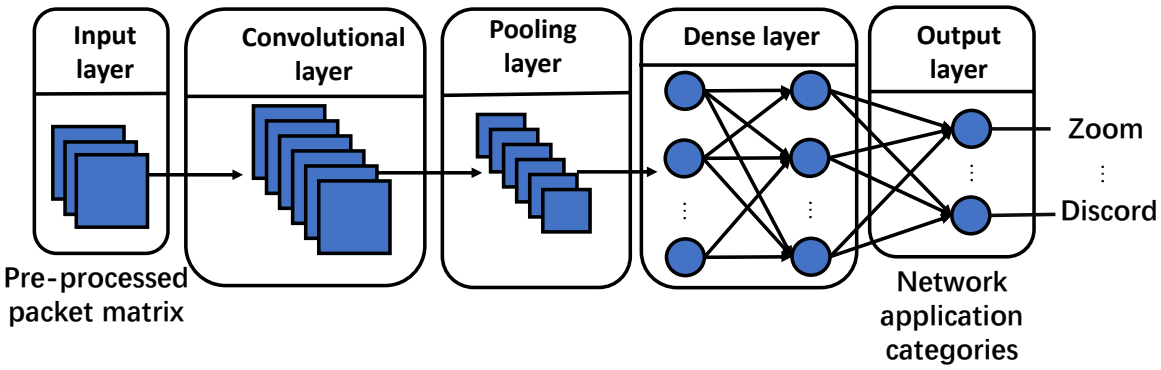

Figure 2.2: General structure of an CNN based NTC.

where '$*$' is the operator for convolution; k is the order of convolution kennels; l is the channel number of the data packets; W and H are the width and length of the convolution kernel; w and b are the weights and bias in each channel, respectively. An activation function, e.g., sigmoid, ReLU, etc, is applied for non-linearity. Pooling layers are usually applied after generating the features from the convolutional layer and activated by activation functions for dimension reduction. Applying pooling can reduce the computational complexity and thus speed up the training and inference process. The outputs of the last convolutional layer are flattened and fed into the dense layers for transferring the information. A softmax and output layer provide the final classification result. Processes of the dense layer and the softmax layer are formulated in Eq. (2.1) and Eq. (2.3), respectively. Efforts are made on CNN based NTC development [33, 34]. The authors in [33] proposed a CNN based NTC trained with a range of images generated from the metadata of encrypted traffic. The proposed method achieved an average accuracy of 94%. The authors in [34] proposed NTCs based MLP, CNN, respectively. The proposed CNN based NTC processes the input data packets which is a 2-D matrix, the output the category, which can achieve an average accuracy of 96%.

2.2 Massive MIMO CSI Feedback Process

The massive multiple-input multiple-output (MIMO) system is a major technology for supporting 5G wireless communication systems [35]. The accurate acquisition of channel state information (CSI) is critical to the performance of beamforming, non-orthogonal multiple access (NOMA), etc. In a frequency-duplex division massive MIMO system, the downlink CSI can be estimated by a feedback process. In this method, the base station transmitter sends a pilot sequence to the user equipment (UE). The downlink CSI matrix is then estimated and compressed. The compressed codebook is then sent back to the base station transmitter for CSI reconstruction. Compressive sensing has been used to compress the CSI matrix in the process [36]. For instance, the authors in [36] use compressive sensing technology to the finite feedback of massive MIMO. Based on the spatial correlation characteristics of massive MIMO channels, Karhunen-Loeve transform (KLT) and discrete cosine transform (DCT) are used to improve the feedback efficiency. However, an increasing number of antennas in a massive MIMO system can lead to the highly increasing of the dimensions and elements in channel correlation matrix, which overwhelms a compressive sensing approach. Recently, Deep learning approaches such as convolutional autoencoder, deep autoenocder, etc., are implemented for CSI feedback reconstruction [37–41].

2.2.1 Convolutional Autoencoder based CSI Feedback Process

Convolutional Autoencoder consists of input layer, convolutional encoder, convolutional decoder and output layer, illustrated as Fig. 2.3. The input is the CSI matrix from UE, the convolutional encoder compresses the CSI matrix by convolutional kernels. The results of the encoder are converted to a codebook and transmitted back to the transmitter for decoding. The convolutional decoder receive the code book processed by the the output from

convolutional encoder, and decompress the codebook and reconstructs the CSI matrix. The output layer generate the reconstructed CSI matrix at BS. Efforts are made in designing the convolutional autoencoder based massive MIMO CSI feedback process [37–39]. In specific, the authors in [37] proposed a convolutional autoencoder based massive MIMO CSI feedback model named CsiNet. In CsiNet, the encoder has 1 convolutional layer and the decoder has 2 refine nets. Each refine net has 4 convolutional layers which have 2, 8, 16, 2 kernels, respectively. The proposed CsiNet achieved a low reconstruction error at -17.36 dB when the compressing ratio is set to 1/4. The authors in [38] proposed an advancement of CsiNet called CsiNet+. Compared with CsiNet, the CsiNet+ increases the number of refine net and thus reached a lower reconstruction error at -27.90 dB. The authors in [39] proposed a DL-based CSI feedback structure called ConvCsiNet for FDD MIMO systems. The proposed scheme uses convolutional layers to extract features, and apply the mean-pooling and upsampling layers to compress and expand the CSI matrix, which achieved a reconstruction error at -13.79 dB when compression raito is set to 1/16. The author also propose a DL-based CSI feedback lightweight structure called ShuffleCsiNet to reduce the complexity. This structure can acquire accurate downlink CSI while consuming low memory space and core computing power, which achieved an average reconstruction NMSE of -9.41 dB.

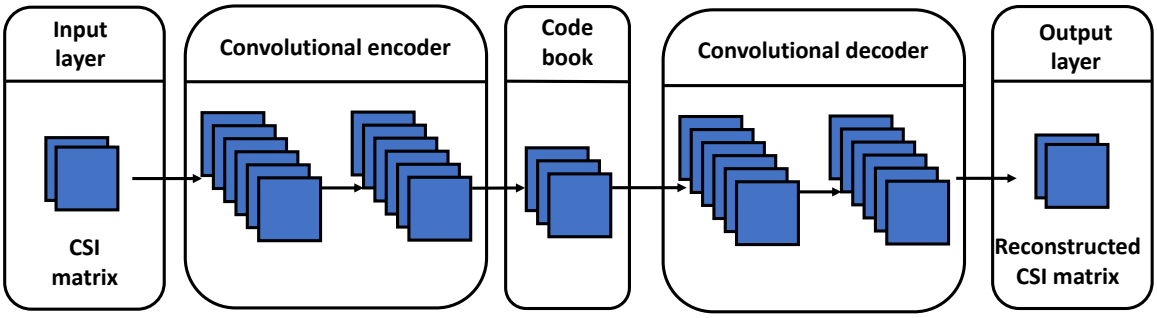

Figure 2.3: Convolutional autoencoder based Massive MIMO CSI feedback model.

2.2.2 Deep Autoencoder based CSI Feedback Process

Deep autoencoder is based on the deep neural network, which consists of an input layer, deep encoder, deep decoder and output layer, as shown in Fig. 2.4. The input in the input layer is the matrix from UE. In the encoder part, one or more hidden layers are applied for transferring the information. An activation function is used for non-linearity after hidden layers generate the features. The decoder based on deep neural network decompresses the codebook and reconstructs the CSI matrix. The authors in [40] propsoed fully connected feedforward neural network based CSI feedback algorithm named CF-FCFNN. CF-FCFNN is able to recover the original CSI from the compressed CSI at a low reconstruction error of -20.07 dB when the compression ratio is set to 1/4. The authors in [42] proposed a deep autoencoder based feedback scheme, which takes into account the feedback errors from the quantization error and noisy uplink channel.

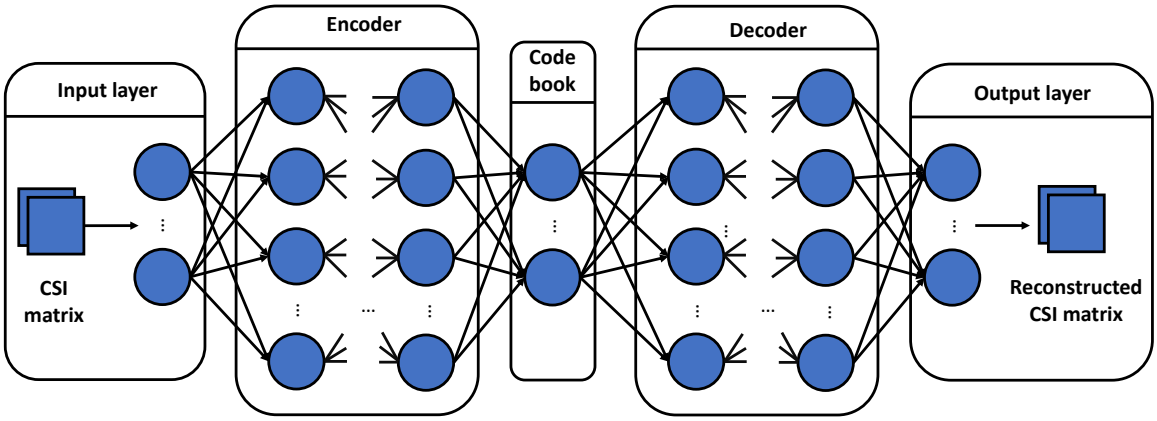

Figure 2.4: Deep autoencoder based Massive MIMO CSI feedback model

2.3 Simplification for deep learning approaches

The deep learning based NTCs and massive MIMO CSI feedback model achieve the high classification and reconstruction performance. However, the existing approaches always have a large number of parameters and high computational complexity. Simplification on the AI development can effectively accelerate the implementation of these systems.

2.3.1 Dimension Reduction

Dimension reduction is a technique that maps the higher order dimensions into lower order dimensions without affecting the salient features for the analysis of data [43]. It consists of feature extraction and feature selection. Feature extraction is a data pre-processing technology that transforms the raw data to numerical features that can be processed while preserving the information in the original data set [44]. Compared with the implementation with raw data, the extracted features can deliver a faster process [44]. Feature selection is a scheme that reduce the number of input, optimize the subset and thus optimize the corresponding neural network architecture. Traditional feature selection algorithms are mainly based on filter, wrapper and embedded methods [45–49]. In specific, the authors in [45] proposed a correlation-based feature selection method to select the useful features based on the correlation between the feature and classes, and the correlation between the features. However, computing correlation for all features may introduce high computational complexity. The author in [46] proposed a fast correlation based filter for feature selection and a naive Bayes classifier to evaluate the proposed scheme. The accuracy is around 88% when using 10% of features. The authors in [47] proposed a feature selection algorithm named Relief to distinguish the samples among the same and different classes that are near each other. However, Relief can hardly remove redundant features due to the limitation

on capturing only the relevance of features to the target [46]. The authors in [50] proposed minimum redundancy maximum relevance based features selection scheme to remove redundant features while ensuring maximum correlation based on mutual information.

2.3.2 Neural Network Architecture Design

Researchers have proposed light-weighted network designs, e.g., XNOR, Binarized Neural Networks (BNN), etc. [51, 52]. In XNOR-Net [51], both inputs and filters to the convolutional layers are binary, which can the implemented with simple convolutional operations with low memory. BNN use binary weigths and activations to train the model and compute the parameters gradients for improving the power-efficiency [52]. Recent efforts have also been made to simplified deep learning models [53–58]. One of the widely accepted approach is pruning, which removes unimportant nodes from an original AI implementation [53]. When pruning, connections in neural network are removed based on the importance of the neurons. After the least important neurons are removed, fine-tuning is applied to avoid degradation of classification accuracy. For instance, the authors in [54] proposed a pruning method named ThiNet, which discards the filter with less importance to accelerate and compress the CNN structure during the training process.

Generally speaking, the AI development in networking and communication systems need to be simplified for deploying on those edge devices, e.g., routers, gateways. However, the existing simplification approaches are not optimized for AI-supported networking and communication systems. In this dissertation, a systematic approach is tailored to simplify and accelerate the AI-supported networking and communication systems.

CHAPTER III

ADAPTIVE AND LIGHTWEIGHT NETWORK TRAFFIC CLASSIFICATION FOR EDGE DEVICES

3.1 Introduction

Advanced network management and measurement, such as network resource provisioning, network anomaly detection, etc., rely on fast and accurate end-to-end traffic classification [15, 34, 59, 60]. Recently, artificial intelligence (AI) approaches, such as Convoluational Neural Network (CNN), Mutilayer Perceptron (MLP), etc. [61–63] are considered good candidates for NTCs because of their high accuracy in processing both clear and encrypted data traffic. The strength of these AI-based approaches is in their high tolerance with respect to data measurement errors, such as result inaccuracies, uncertainties, and approximations [64]. However, the existing AI-based NTC models apply full data packet inputs with padding, which result in complex NTC designs. Such a complex AI-based NTC can be hardly implemented on networking edge devices with limited computing capability, e.g, Wi-Fi routers, IoT devices, etc.

To address this issue, an adaptive and lightweight AI-based NTC framework is developed in this chapter to optimize the selection of input features and consequently to systematically simplify a targeting NTC. The existing feature selection methods such as correlation feature selection (CFS) and minimum redundancy maximum relevance (MRMR) may not work well for simplifying the inputs to NTC models, due to the zero paddings applied to those AI-based NTCs [34]. Those large number of zeros may affect the efficiency of the existing feature selection methods. In the proposed lightweight network traffic classification framework, contribution of features are determined based on both the significance to NTC outputs and

the uniqueness among all input features. The significance of each feature is computed base on weights in a benchmarking model built on Bayesian learning [65–68]. The uniqueness of each feature is computed based on Symmetrical Uncertainty (SU) of each feature with others. An optimal feature selection scheme is designed to minimize the number of chosen features that in turn reduce the complexity of a targeting AI-based NTC, while maintaining high classification accuracy. To further accelerate the optimization process, a practical approach is developed to avoid the complex computation of uniqueness of all features.

With an optimized feature set, a targeting AI-based NTC can be implemented with a much simplified structure for lightweight networking edge devices. However, feature contributions may change due to update of network applications. To address this issue and sustain the high classification accuracy, an autonomous update scheme is developed to detect changes of feature contributions and update the targeting AI-based NTC. In specific, a change of feature contribution is detected based on two factors, i.e., model confidence level and feature discrepancy. Model confidence level is defined as the cumulative distribution function (CDF) value of softmax value in the classification layer of the benchmarking model [69]. Feature discrepancy is defined as difference between the variance of network traffic samples in dataset and the variance of a collection of on-going network traffic. Once a change of feature contribution is detected, the optimal features will be updated by collecting and labeling active network traffic, fine-tuning the benchmarking model, extracting the contribution of each input features, and selecting the new optimal subset for updating the lightweight AI-based NTCs.

The proposed lightweight network traffic classification scheme is evaluated with the AI-NTC models based on two fundamental deep learning based methods, i.e., MLP and CNN [61–63]. The evaluation results demonstrate that the proposed lightweight network

traffic classification scheme can much simplified an AI-NTC design that is faster than the baseline deployment by one to two orders of magnitude without sacrificing the accuracy. The autonomous update scheme can maintain the high accuracy when the feature contributions are changed. To summarize, the major contributions of this chapter are:

- A feature contribution extraction scheme is developed to weigh each input feature in an AI-based NTC design.
- An NTC simplification scheme is developed to find the optimal set of input features that reduces the complexity of a targeting AI-based NTC without sacrificing the performance.
- A trade-off function is further developed to quickly approximate the optimal number of features for NTC model simplification.
- An autonomous detection and update scheme is developed to sustain the performance of AI-based NTC from dynamic changes of feature contributions.
- Performance analysis is provided using AI-NTC models based on two fundamental deep learning based methods and open datasets.

The rest of the chapter is organized as follows. Section 3.2 describes the related work. Section 3.3 introduces the proposed lightweight AI-based NTC framework. Section 3.4 describes the proposed NTC simplification and autonomous update scheme. Section 3.5 demonstrates the evaluation results. Section 3.6 concludes this chapter and describes the future work.

3.2 Related work

3.2.1 AI-based NTC

Efforts have been made on DL-based NTC models, e.g., CNN, MLP, Recurrent Neural Networks (RNN), etc. for building network traffic classifiers [70–73]. In specific, the authors in [70] proposed Deep Packet to classify the network traffic based on stacked autoencoder and CNN, respectively. The authors in [71] proposed a CNN based malware traffic classification model which transform the raw packet data to the images. The authors in [72] proposed a deep learning based packet classifier by combining CNN and recurrent neural network for IoT traffic. Their method returned a classification accuracy above 96%. The authors in [73] proposed a traffic classification mechanism based on capsule network for smart cities, which achieved an average classification accuracy above 99% in a specific testing case. However, applying the full-size packet or intrusion data leads to the complex NTC and intrusion detection systems, with low process bandwidth and a demand for relatively large storage spaces (e.g, 6.7 Mbps bandwidth, 37.5 MB storage in [70]), which can be challenging to edge devices. Hence, the proposed adaptive and lightweight NTC design in this work focuses on optimizing the input feature set so that a targeting AI-based NTC can be simplified afterwards.

3.2.2 Dimension Reduction Techniques

Dimension reduction is a technique that maps the higher order dimensions into lower order dimensions without affecting the salient features for the analysis of data [43]. It consists of feature extraction and feature selection. Feature extraction algorithms are used to extract the most distinct features present in a dataset which are used to represent and describe the data [74]. Efforts have been made on feature extraction algorithms, e.g, Principal

Component Analysis (PCA), Sparse Autoencoder (SAE), etc, for extracting the required features to enhance the performance of DL-based NTC [75,76]. In [75], the authors proposed a network traffic classifier based on the deep convolution recurrent autoencoder neural networks to extract the features with a combination of temporal features and spatial temporal features. In [76], the authors proposed a malware NTC system based on the PCA and artificial neural network. However, the chosen feature extraction schemes still rely on a full data input for each classification process.

Traditional feature selection algorithms are mainly based on filter, wrapper and embedded methods [45–47,50]. In specific, the authors in [45] proposed a correlation-based feature selection method to select the useful features based on the correlation between the features and the output classes, and the correlation among the features. However, computing correlation for all features may introduce high computational complexity. The authors in [46] proposed a fast correlation based filter for feature selection and a Naive Bayes classifier to evaluate the proposed scheme. The accuracies are relatively low at around 88% using around 10% of features in ten datasets. The authors in [47] proposed a feature selection algorithm named Relief to distinguish between the example of the same and different classes that are near each other. However, Relief can hardly remove redundant features due to capturing only the relevance of features to the target [46]. The authors in [50] proposed minimum redundancy maximum relevance based features selection scheme to remove redundant features while ensuring maximum correlation based on mutual information. However, computing all correlations could be an issue for a large number of features. The existing methods may not be effective for AI-based NTC due to the varying nature of network applications and protocols that lead to zero padding of the data packets. In addition, the

Table 3.1: Summary of notations in this chapter.

Notation	Remark
$\mathbf{x}$	Data input.
y	Stochastic output.
$\mathcal{D}$	Original dataset.
b_i	Input feature i.
c_i	The contribution of input feature.
r_i	The signifiance of input feature.
u_i	The uniqueness of input feature.
M	Total number of hidden layers.
J	The number of selected packets in training dataset.
L	The number of nodes in the first hidden layer.
$w_{i,l}^{j}$	Weight between feature b_i and l-th node. by classifying packet j.
$\mathbf{w}^{(m)}$	Weights of the m-th layer in the benchmarking model.
$\rho_{i,j}$	SU between input features b_i and b_j.
N	Total number of input features.
μ	Ratio of selected to total input features.
$\mathcal{S}$	Selected input features for a targeting AI-based NTC.
d_t	Stability of feature contributions at time t.

existing schemes rely on forward selection, backward elimination, and sequential forward selection, which can be time consuming to rank the features for an optimal selection.

The proposed lightweight AI-based NTC design in this work tackles these issues by defining the feature contribution based on both the input feature significance to the NTC output result and the uniqueness of each feature. A fast and practical approach is developed to determine the optimal feature selections by avoiding computing relevance among all features. Moreover, an adaptive detection and update scheme is developed to detect changes of feature contributions during active operations and update the optimal selection of features to maintain performance of the targeting AI-based NTC.

3.3 Lightweight AI-based NTC Design

3.3.1 Overview of the Lightweight AI-based NTC Framework

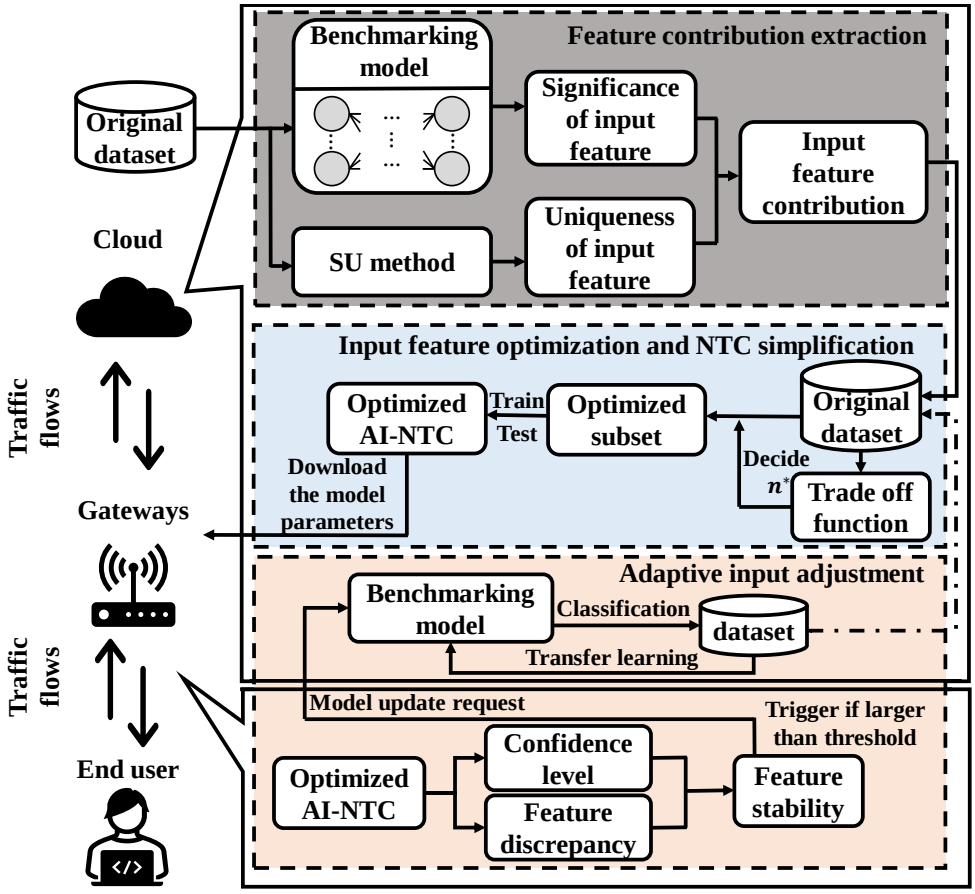

Figure 3.1: Overview of the lightweight AI-based NTC design framework.

The proposed lightweight AI-based NTC design is shown in Fig. 3.1.For better illustration, the notations used in the rest of the chapter is listed in Table 3.1. The input features b_i are defined as the payload bytes in each data packet, e.g., 1456 or 1480 features in several existing NTC designs after zero padding and removal of headers [34, 61–63, 70–72, 77, 78]. The contribution of each feature is measured based on its significance and uniqueness. A benchmarking NTC built on Bayesian learning model [66–68] is applied to weigh the signif-

icance of each input feature. SU is implemented to evaluate the uniqueness of each input feature. With the contributions of input features computed, the optimal set of features are selected according to a trade-off function that balances the complexity and performance of a targeting AI-based NTC model. The targeting AI-based NTC model is finally simplified according to the optimal set of input features. In addition, an autonomous model update scheme is provided to combat the updates and changing contributions of input features during active operations.

3.3.2 Extraction of input feature Contribution

The contribution of a feature b_i is defined as the combination of the significance r_i and correlation u_i of each input feature, computed as follows:

$$c_i = \sigma(r_i) - u_i, \tag{3.1}$$

where $\sigma(\cdot)$ is a normalization function, e.g., max-normalization [79]. A higher contribution indicates that the input feature should be considered more when implementing a targeting NTC model. The significance value r_i measures how an input feature is linked to the output result. A higher significance values indicates that the particular input feature plays a decisive role in the classification process. The uniqueness u_i measures how an input feature is different from the others. A higher value of u_i indicates that the particular feature is highly relevant with others, or lesser unique in other words. The details of calculating r_i and u_i are given below.

3.3.2.1 Input feature significance

Given a data input with full features $\mathbf{x}$ in dataset $\mathcal{D}$, a benchmarking model generates a stochastic output y through Bayesian learning [68], denoted as follows:

$$p(y|\mathbf{x}, \mathcal{D}) = \int p(y|\mathbf{x}, \theta)p(\theta|\mathcal{D})d\theta, \tag{3.2}$$

where θ is the parameter set of the benchmarking model, i.e., weights and bias. In Bayesian learning, weights $\mathbf{w}$ are initialized based on some probability distributions, e.g., Gaussian prior distribution [66]. The outputs of the m-th layer are:

$$\mathbf{z}^{(m)} = f(\mathbf{w}^{(m)} \cdot \mathbf{z}^{(m-1)} + \mathbf{b}^{(m)}),\ m \neq M, \tag{3.3}$$

where M is the total number of hidden layers; $w^{(m)}$ and $b^{(m)}$ are the weights and bias in m-th layer, respectively; $f(\cdot)$ is an activation function, e.g., Softplus [80]. A Softmax and classification layer outputs a prediction result. With a benchmarking BMLP model implemented, all weights in the first hidden layer i.e., $\mathbf{w}^{(1)}$ are optimized and directly connected to the input features. In other words, the output of a node in the first hidden layer i.e., $z_i^{(1)}$ is the summation of the product of every input feature and the corresponding weight. In this case, the ratio of corresponding weight can roughly represent the relevance of the particular input feature to the classification result. Since the weights in the benchmarking model are optimized from multiple training samples, we can calculate the input feature significance r_i as the average value of all ratio of weights linked to feature i processed by different input packet multiple times, which can be denoted as:

$$r_i = \frac{1}{JL} \sum_{j=1}^{J} \sum_{l=1}^{L} \frac{|w_{i,l}^j|}{\sum_{i=1}^{N} |w_{i,l}^j|}, \tag{3.4}$$

where N is the total number of input features; $w_{i,l}^j$ is the weight connected between feature b_i and l-th node by classifying packet j; L is the number of nodes in the first hidden

layer; J is the number of selected packets . Note that J can be the total number of entries in a training dataset, or determined by the user for better efficiency. Each weight in benchmarking model follows an independent distribution [66]. Based on this property, J sets of weights for processing a packet randomly chosen from the dataset by J times.

3.3.2.2 Input feature uniqueness

The uniqueness of an input feature u_i is calculated as the square of correlation between input feature b_i and all features as:

$$u_i = \frac{1}{N-1} \sum_{j \neq i} \rho_{i,j}^2, \tag{3.5}$$

where $\rho_{i,j}$ is the correlation between a given input feature b_i and another feature b_j in the dataset. A lower u_i represents that the particular input feature is highly different from others, while a higher u_i means that this input feature is more related to other input features in the dataset. For illustration purpose, SU is implemented to find the correlations between input features. SU correlation is a measure of the uncertainty of a random variable [81]. In this case, the correlation $\rho_{i,j}$ between the input feature b_i and b_j can be computed as:

$$\rho_{i,j} = \frac{2(H(b_i) - H(b_i|b_j))}{(H(b_i) + H(b_j))}. \tag{3.6}$$

where $H(b_i)$ and $H(b_j)$ are the entropy of the features b_i and b_j in packets. For instance, the entropy of a given feature b_i can be denoted as:

$$H(b_i) = -\sum_{k} p(b_i^k) \log_2 (p(b_i^k)), \tag{3.7}$$

where $p(b_i)$ is the prior probabilities for all values of b_i. $H(b_i|b_j)$ is the entropy of b_i after observing values of another features b_j in packets, which is defined as:

$$H(b_i|b_j) = -\sum_{k_1} p(b_i^{k_2}) \sum_{k_2} p(b_i^{k_1}|b_j^{k_2}) \log_2(p(b_i^{k_1}|b_j^{k_2})), \tag{3.8}$$

where $b_i^{k_1}$ is the input feature b_i with k_1 packets. The difference between $H(b_i)$ and $H(b_i|b_j)$ represents the information gain (IG) [82], i.e, $IG(b_i|b_j) = H(b_i) - H(b_i|b_j)$. Information gain can calculate and compare the correlations between features in packets, e.g., the feature b_i is considered as a higher correlation to the feature b_j than to the feature b_k if $IG(b_i|b_j) > IG(b_i|b_k)$. For two selected features b_i and b_j, the IG method is symmetrical, which is a good property for measuring the correlations between features [82]. Subsequently, applying SU method which has mentioned in Eq. (3.6) can avoid bias from IG towards the features which has more valuable information and the values are normalized to the range [0,1]. The value 1 indicates that either feature entirely predicts the other and value 0 means that the two features in packets (e.g, b_i and b_j) are totally independent. The correlation of each pair of the input features are calculated based on the SU method (i.e, calculated by Eq. (3.6)). The input feature correlation of the given feature b_i is then calculated as the square of correlation between the b_i and all other input features in the dataset.

After input feature significance and input feature correlation are calculated, the input feature contribution is determined based on Eq. (3.1). The overall scheme is shown in as Alg. 1.

3.3.3 Optimal Feature Selection

Let n be the number of the selected features with the highest contributions in an optimal feature set for the simplified network traffic classifier. A larger n (e.g, full-length data packet) can provide a higher classification accuracy, however, may require a more complex

Algorithm 1: Input feature contribution calculation

Input: Full-size packet dataset $\mathcal{D}$;
Output: Input feature contribution c_i;
Initialization;
Initialize a benchmarking model;
for $i = 1; i \leq N; i{+}{+}$ **do**
    Calculate r_i based on Eq. (3.4).
    Calculate self entropy $H(b_i)$ based on Eq. (3.7)
    for $k = 1; k \leq N \ \ \& \ \ k \neq i; k{+}{+}$ **do**
        Calculate $H(b_i|b_k)$ based on Eq. (3.8);
        Calculate $\rho_{i,k}$ based on Eq. (3.6);
        Calculate u_i based on Eq. (3.5).
    end
    Calculate c_i based on Eq. (3.1).
end

AI model design. A smaller n can lead to a lightweight classifier design, however, may suffer from lower classification accuracy due to the loss of information. To balance the complexity and performance, a trade off function is proposed with regard to three factors:

$$g(\mu) = \frac{1}{\beta \cdot \mu + 1}(1 + e^{\alpha \cdot \beta \cdot \mu})^{-2}, \tag{3.9}$$

where μ is the ratio of selected features among the whole features, i.e, $\mu = \frac{n}{N}$. A higher μ indicates indicates more selected features and thus a more complex implementation of the targeting AI-based NTC. α is a parameter which consists of the total number of the features in dataset, i.e., $\alpha = [\log_{10}(N{+}1){+}1]$. β is the average entropy of the input features, i.e., $\beta = \frac{1}{N}\sum_{i=1}^{N} H(b_i)$. A larger β represents more information in each input feature thus fewer features may be needed for a targeting AI-based NTC implementation. The function $g(\cdot)$ provides an insight on the trade-off between complexity of the targeting model and the classification accuracy based on total and the selected feature set. Please note that

the Trade-off function always has a maximizer μ^* due to the property of quasi-concavity of sigmoid function [83].

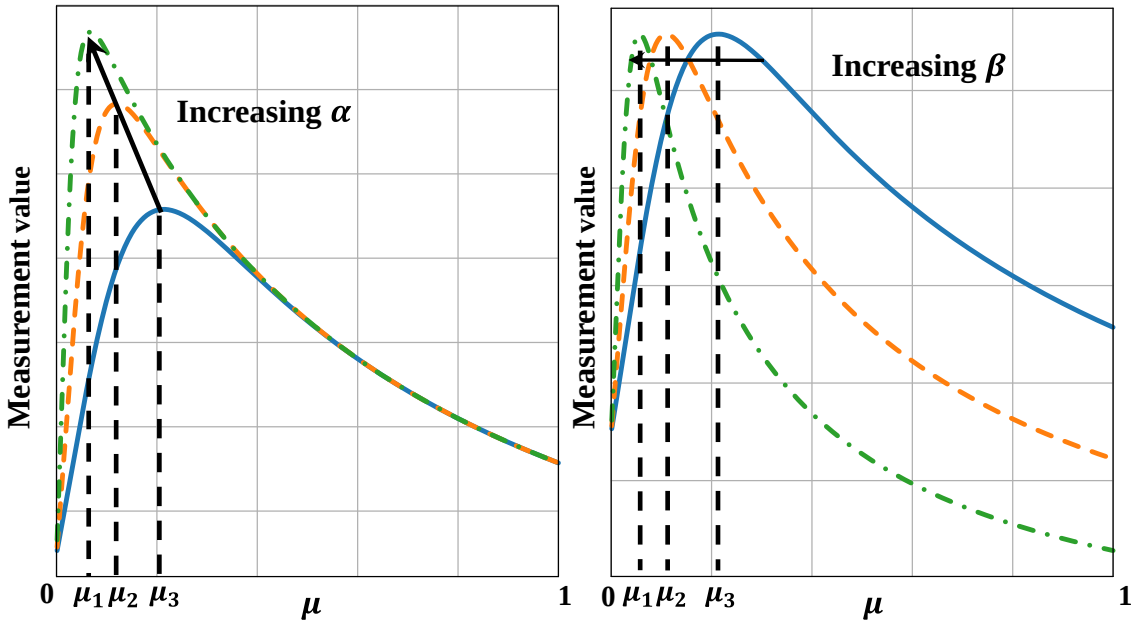

Figure 3.2: The trade off function with (a) different α and (b) diffrerent β.

Fig. 3.2 shows how the maximizer changes when α and β are changed, respectively. In Fig. 3.2 (a), with the increasing of α which means increasing of N, the maximizer μ^* would decrease to avoid selecting too many features. In Fig. 3.2 (b), with a increasing of β which means increasing of information of each input features, the maximizer μ^* would decrease to avoid selecting repeated features. Thus, the maximizer μ^* can be used to estimate the optimal number of features selected for a targeting AI-based NTC implementation, s.t., used to calculate to estimate n^*, which can be denoted as:

$$n^* \approx N \cdot \underset{\mu}{\operatorname{argmax}}(g(\mu)). \tag{3.10}$$

The original dataset can be reduced to the optimal subset with n^* input features. Therefore, the corresponding structure of the network classifiers also becomes more light-weight which

means the nodes or kernels can be reduced according to the selected input features. For best performance, the optimized network classifiers can be trained in cloud, and then be pushed to the networking edge devices (e.g., routers). The overall scheme is summarized in Alg 2, where the $\mathcal{M}^*$ is the simplified network classifier and $\mathcal{D}^*$ is the optimized dataset.

Algorithm 2: Input feature optimization.

Input: $\mathcal{D}$
Output: $\mathcal{D}^*$, $\mathcal{M}^*$
Initialization;
Calculate α with N;
for $i = 1; i \leq L; i{+}{+}$ **do**
  Calculate $H(b_i)$ based on Eq. (3.7).
end
Calculate $\beta = \sum_{i=1}^{N} H(b_i)$;
Find optimizer μ^* for Eq. (3.9);
Compute n^* based on Eq. (3.10);
$\mathcal{D}^* \leftarrow$ Simplify $\mathcal{D}$ with features of top n^* contributions. ;

3.3.4 A Practical Approach to an Optimal Feature Set

In the scheme proposed in the previous subsection, input feature significance and input feature uniqueness of each feature in packet are calculated, respectively. With a large amount of features, the computing cost of finding all correlations $\rho(i, j)$ may not be affordable even with cloud computing. In this case, we further propose a practical approach. Initially, the feature with low input feature significance can be removed based on user needs. After that, the input features are sorted according to their significance values in the descending order, i.e., $b_1^{'}, b_2^{'}, ...$ where $r_1^{'} \geq r_2^{'} \geq ....$ Let $\mathcal{S}$ be the subset of the selected input features, and $c_{\mathcal{S}}$ be feature set contribution. The feature set contribution is computed as:

$$c_{\mathcal{S}} = \frac{1}{n_{\mathcal{S}}} \sum_{i=1}^{n_{\mathcal{S}}} c_i. \tag{3.11}$$

where $n_{\mathcal{S}}$ is the number of input features in $\mathcal{S}$. c_i is the input feature contribution of feature b_i calculated based on Eq. (3.1). Instead of calculating the correlation between the target input features and all other features in dataset, the uniqueness of input feature b_i is evaluated only between b_i and other input features in $\mathcal{S}$, such that:

$$u_i^{'} = \frac{1}{n_{\mathcal{S}}} \sum_{k=1}^{n_{\mathcal{S}}} \rho_{i,k}^2, \quad k \neq i, \tag{3.12}$$

The optimal number of features in the set can be found by computing:

$$n^{'*} = \text{argmax}(\text{c}_{\mathcal{S}}). \tag{3.13}$$

Note that an approximation of $n^{'*}$ can be quickly found by Eq. (3.10). Only the values around the approximated result need to be evaluated instead of an exhaustive search. The overall scheme is summarized in Alg. 3.

3.3.5 NTC Simplification

The structure of the network traffic classifier can be simplified after optimal feature set is chosen. Due to the decrease of the input features, the parameters in deep learning based network traffic classifier also can be reduced. Two popular deep learning based network traffic classifier are chosen, i.e., MLP and CNN based. There are some rules to help decide the structure of the MLP or CNN based network traffic classifier. For instance, two hidden layer is designed in our MLP based model, and the number of hidden layers neurons in MLP can be approximated as the same with the summation of number of input and output [84]. In other words, the corresponding number of nodes in hidden layers can be reduced after the optimal feature set is determined, and thus the simplified deep learning based network

Algorithm 3: Practical approach to an optimal feature set.

Input: $\mathcal{D}, c_i$,
Output: $\mathcal{D}^*$

Initialization;
$\mathcal{D}^* \leftarrow \emptyset$;
$\mathcal{D}^* \leftarrow$ The feature with largest $\bar{w}_i$ in $\bar{\mathbf{w}}$.
for $i = 1; i \leq L-1; i{+}{+}$ **do**
  for $j = 1; j \leq L-i; j{+}{+}$ **do**
    Extract the w_j in $\mathbf{w}$ based on Alg. 1.// the average weights linked to b_j.
    for $v = 1; v \leq i; v{+}{+}$ **do**
      Calculate the $\rho_{j,v}$ in $\mathcal{D}^*$ based on Eq. (3.6).
    end
    Calculate the $u_i^{'}$ by Eq. (3.12).
    Calculate c_j based on Eq. (3.1).
  end
  $\mathcal{D}^* \leftarrow$ the feature b_j with the highest c_j.
  $\mathbf{c}_{\mathcal{S}} \leftarrow$ Eq. (3.11).
end
$n^{'*} \leftarrow \text{argmax}(\mathbf{c}_{\mathcal{S}})$.
$\mathcal{D}^* \leftarrow$ The top $n^{'*}$ features in $\mathcal{D}^*$.

traffic classifier can be built. For CNN based network traffic classifier, the baseline usually designed based on some famous CNN structures, e.g., ResNet, Mobilenet or structures in other papers [85, 86]. The reduced input features will reduce the settings of kernels, such that reduce the kernels according to the reduction ratio of the input matrix.

3.4 Autonomous Update

If the feature contributions in the coming packets are changed due to changed encryption method, the deteriorated information source, etc, optimal feature set may need an update. In this case, an autonomous update scheme is proposed to monitor and update the change of the input features. A triggering algorithm is designed to monitor the coming packets and alarm when the features change. If triggered, a new optimal subset is required to maintain the performance. Here we assume in a close world (i.e., no packets from new applications

come), the well-trained benchmarking model still can classify the coming packets with the full-size packets. If the benchmarking model also fails to classify the coming packets, all models need to be updated. Please refer to our preliminary work [87] for a possible solution. The discussion will be limited to a close-world assumption in this work. The detection algorithm depends on two factors, i.e., confidence level and feature discrepancy.

The confidence level evaluates how confident of a targeting AI-based NTC about the output result. The confidence level can be represented by CDF of maximum output of the softmax function in the optimized AI-NTCs, i.e., $p(\mathbf{s} \leq \delta)$, where $\mathbf{s}$ is the vector of maximum output of the softmax function s^*. δ is a preferred threshold set by the users. A decreasing value of confidence level represents that the classification of the model is uncertain due to the changes of input feature contributions in coming packets. Feature discrepancy is the difference between the variance of the input features in original dataset and the variance of the input features of collect packets in batch. For illustration, the Kullback-Leibler (KL) divergence [88] is chosen to compute the feature discrepancy as:

$$D_{\mathrm{KL}}(p_D(b_i) \mid\mid p_b(b_i)) = \sum_{i=1}^{L} p_D(b_i)) \cdot \log_2 \frac{p_D(b_i)}{p_b(b_i)} \tag{3.14}$$

where $p_D(b_i)$ is the distribution of feature variance in original dataset; and $p_b(b_i)$ is the distribution of feature variance in batch. The feature stability d_t in triggering scheme considers both confidence level and feature discrepancy as:

$$d_t = \frac{1}{n} \sum_{i=t-n+1}^{t} p^{(i)}(\mathbf{s} \leq \delta) \cdot D_{\mathrm{KL}}^{(i)}(p_D(b_i) \mid\mid p_b(b_i)) \tag{3.15}$$

A moving average is applied in Eq. (3.15) for smoothing the process and avoiding the impact of outliers [89]. When d_t exceeds a user defined threshold ϕ, update process is triggered. The overall scheme is summarized in Alg. 4.

Algorithm 4: Autonomous update

Input: $\mathcal{M}^*, \mathcal{D}, \phi$.
Output: d_t; $\mathcal{M}^*$

```
Initialization;
Compute the p_D(b_i).
while Autonomous update do
    if e < φ then
        for Ever batch of collected M coming packets do
            Compute p(s ≤ δ).
            Compute p_b(b_i).
            Compute d_t based on Eq. (3.14).
        end
    else
        Collect and classify the coming packets by benchmarking model.
        Use practical approach to decide new optimal feature set. Fine tuning M*.
    end
end
```

3.5 Evaluation Results

3.5.1 Datasets and Settings of Baseline NTCs

One of dataset used to evaluate the proposed scheme is UTSC-TFC 2016 [71], shown as dataset 1 in Table 3.2. A total number of 242211 benign raw data flows are collect from ten categories, e.g., Outlook, Facebook, etc, and 179252 malware raw data flows from ten type of attacks, e.g., Geodo, Neris, etc, respectively. Another dataset used is from UD-ACNS [67, 87], shown as dataset 2 in Table 3.2, the dataset consists of over 3 million data packets from 8 mobile applications, e.g., Youtube, Discord, etc. The evaluations are conducted on a workstation that is equipped with an Intel®Core(TM) i7-6700 CPU@ 3.40 GHz, 32.0 GB RAM, and a Nvidia GeForce GTX 1080Ti. The AI-based NTC models are implemented on the PyTorch platform. Accuracy, precision, recall rate, and F1 score, bandiwdth on CPU and GPU are the chosen evaluation metrics. Two popular AI-based NTC models are implemented as baseline models, i.e., an MLP-NTC and a CNN-

NTC [66–68]. In particular, based on [90], the baseline MLP based NTC is implemented with 2 hidden layers. The number of hidden nodes are 1456 and 1480, respectively, according to the total number of features. The baseline CNN-NTC takes input a square matrix that is reshaped from the input packet after padding. 2 convolutioinal layers are applied with 32 kernels (3×3) in each layer. The flattened features are downsized to 128. Both baseline NTC models output their prediction results through a standard softmax and classification layer. To initialize the two baseline NTC models, a balanced subsets with 10 applications from dataset 1 are used. 20000 and 2000 samples are randomly chosen from each attack as the training and testing dataset, while a balanced subsets with 7 applications from dataset 2 are randomly chosen, 10000 and 4000 packets are randomly selected for each application as the training and testing dataset. The details of the baseline models are shown in Table 3.4. As we can see, both baseline NTCs can classify the packets with a high average accuracy above 95%.

3.5.2 Feature Contribution Extraction

A Bayesian MLP model is implemented as the benchmarking model. The input is the full-length data packet which has a dimension of $1\times$ 1456 and $1\times$ 1480 for dataset 1 and dataset 2, respectively. The benchmarking model has two hidden layers with 728 nodes, and a softmax and classification layer for the final classification. The benchmarking models can achieve 95.8% and 95.5% using dataset 1 and dataset 2, respectively, which are reasonably high to evaluate feature contribution. With the benchmarking model, K sets of weights are generated processing K data packets randomly chosen from the dataset.

Fig. 3.3 shows the significance, uniqueness and contribution of the input features using two datasets, respectively. For better illustration, the top 50 features according to feature

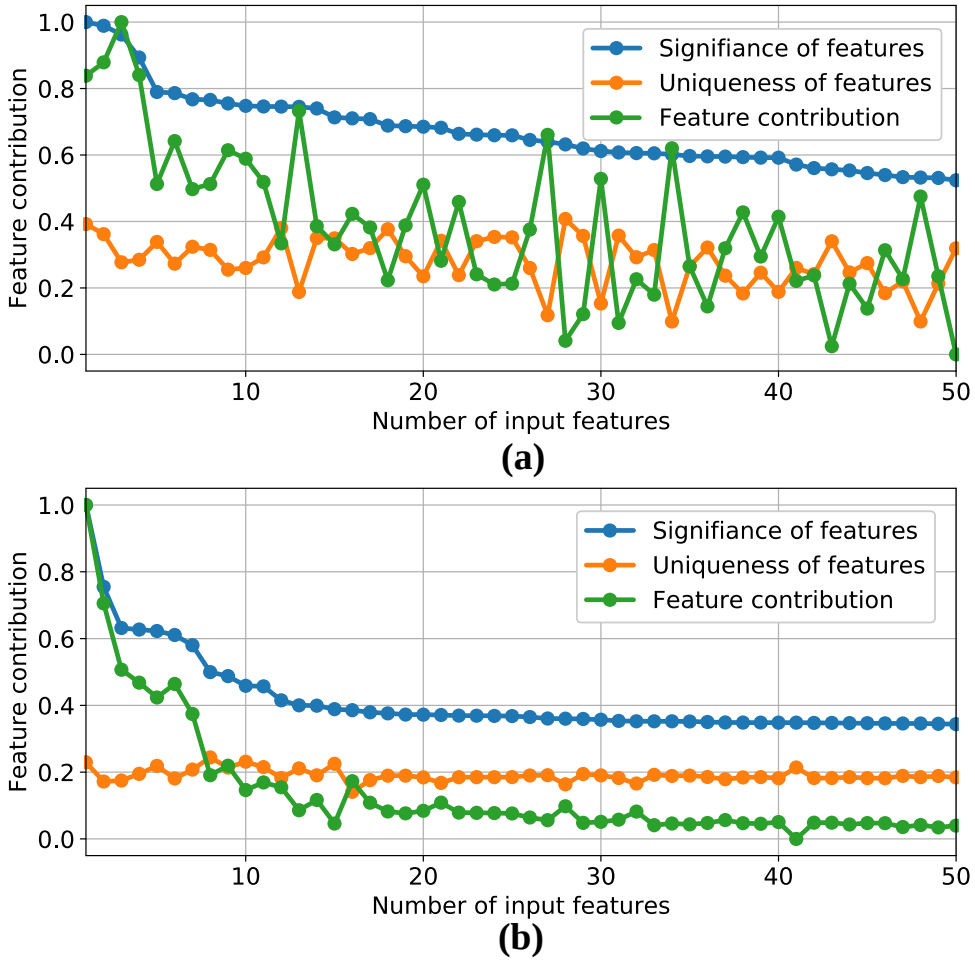

Figure 3.3: The significance, uniqueness, and contribution of features using (a) dataset 1 and (b) dataset 2.

significance are chosen to evaluate the proposed system. Note that the features are sorted based on significance of features from the highest to the lowest.

Fig. 3.4 shows the curve of the trade-off function, mean feature contribution and feature set contribution using two datasets. After feature contributions are calculated, the optimal number of selected features is decided based on trade off function. As we can see, using dataset 1, the trade off function achieve the highest value when the ratio of the selected features is around 34%, which means the optimal number of selected features provided by

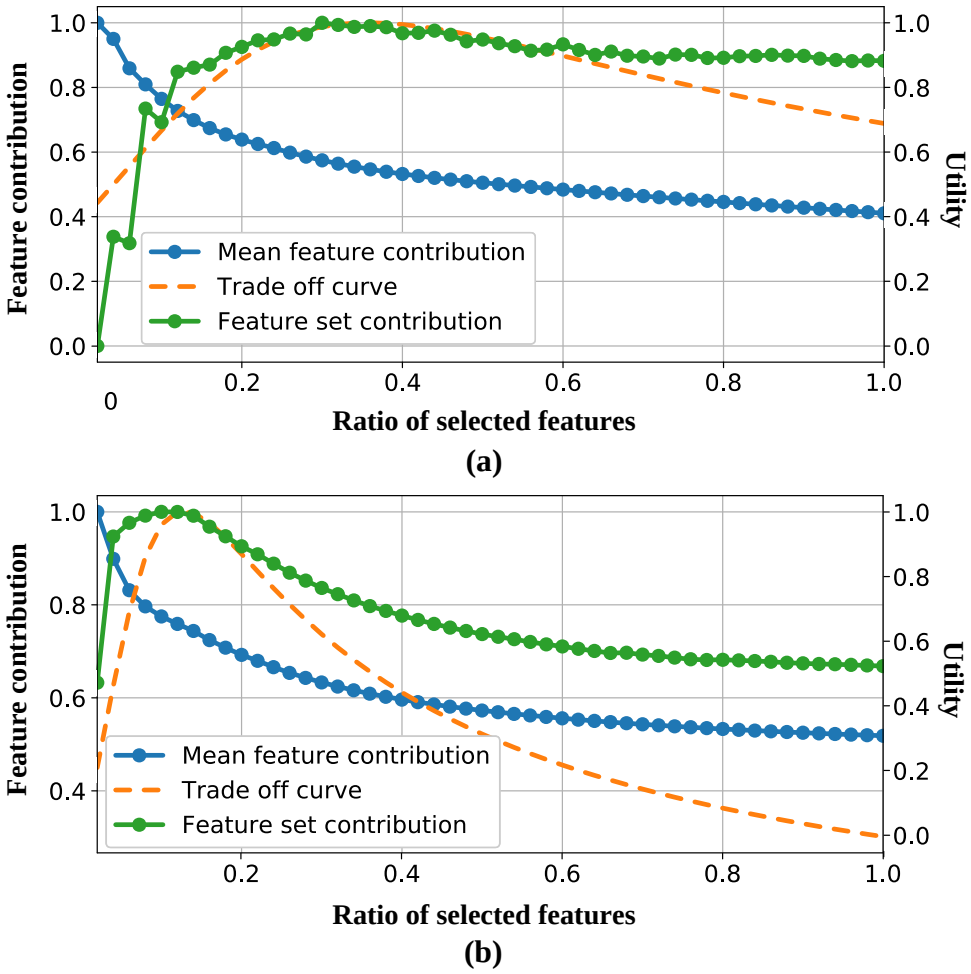

Figure 3.4: The curve of the trade-off function, mean feature contribution and feature set contribution using (a) dataset 1 and (b) dataset 2.

the trade off curve is roughly 17, and the trade off curve has the highest value when the ratio of the selected input features is around 12% which means roughly 6 features with the top 6 input feature contribution are chosen using dataset 2.

Fig. 3.4 also shows the mean feature contribution and feature set contribution using two datasets in the practical approach. Instead of evaluating all the features, the practical approach only compute the uniqueness of features between the target feature and other selected features. The optimal number of selected features achieves the highest feature set

contribution. As we can see, using dataset 1, the mean feature significance decreases with the increasing number of the selected input features because the input features are chosen in order of the input feature significance. The optimal ratio of selected features is based on the largest feature set contribution which is around 30% (i.e., approximately 15 features). Using dataset 2, the ratio of selected features is 10% (i.e., approximately 5 features) which corresponds the optimal feature set contribution.

3.5.3 Evaluations of the Simplified AI-based NTC Models

As discussed earlier, 15 and 5 input features are chosen from dataset 1 and dataset 2, respectively, to optimize the targeting AI-based NTCs. For comparison, 5 existing methods are developed and evaluated with the same optimal subset of features. The comparing schemes are fast correlation based filter (FCBF) [91], minimum redundancy maximum relevance (MRMR) [92], correlation-based feature selection (CFS) [45], relief-based feature selection (ReliefF) [92], CHI-square based feature selection (CHI) [93]. Based on the NTC implementation described in Section 3.3 E [84], the simplified MLP-NTCs have two hidden layers with 25 nodes in each layer with dataset 1, and 12 nodes in each layer with dataset 2. The optimized CNN-NTC has 2 convolutional layers with 8 kernels (3×3). The features after the last convoluational layer is flattened and downsized to 32. Both optimized NTC models has a standard softmax and classification layer. In addition, we also evaluate the smallest number of features when achieving the highest accuracy with the five comparing schemes.

Table 3.4 shows the recall, precision, F1 score, accuracy, bandwidth from CPU and GPU, storage and complexity of the baseline and simplified AI-NTCs. Notice that the complexity is evaluated with floating point operations per second (Flops). Using dataset

1, both baseline and simplified MLP-NTCs and CNN-NTCs provide the accurate classification results above 96%. Using dataset 2, the baseline and simplified MLP-NTCs and CNN-NTCs provide classification performance above 95%. Moreover, in Table 3.3, the two simplified MLP-NTCs accelerate the corresponding processing bandwidth by 13.6, 105 and 14, 184 times, on GPU and CPU, respectively. The two simplified CNN-NTCs accelerate the corresponding processing bandwidth by 141, 153 and 128, 89 times, on GPU and CPU, respectively. Meanwhile, the storage of the two simplified MLP-NTCs have both dropped to approximately 0.1% of the original needs from the baseline implementations. The storage of the two simplified CNN-NTCs have both dropped to approximately 0.05% of the original needs from the baseline implementations. The complexity of the two simplified MLP-NTCs has dropped to 1.6 K and 0.29 K Flops from the 4.4 M and 4.25 M Flops of the baseline implementations. The complexity of the two simplified MLP-NTCs has dropped to 15.9 K from the 41 M Flops of the baseline implementations.

Fig. 3.5 shows the performance of the proposed scheme and built methods with different number of features in packets using two dataset, respectively. We use the baseline MLP-NTC to evaluate these methods. Features are ranked based on each schemes. After specifying the number of selected features (e.g., N), an increasing number of features (e.g., 1% to 20% of total input features in this scenario) are selected and evaluated by the baseline MLP-NTC. Fig. 3.5 (a) shows the change of classification accuracy with the increasing number of selected features using dataset 1. As we can see, with the increasing number of the features in subset, the accuracies of all method keep increasing. The accuracy based on our method keeps the highest which is above 95% with only 1% to 20% of selected features. CHI method keeps around 80% accuracy, and accuracy based on MRMR, FCBF methods from starts from 70% and 55%, and reach around 95% when 11% and 5% features are

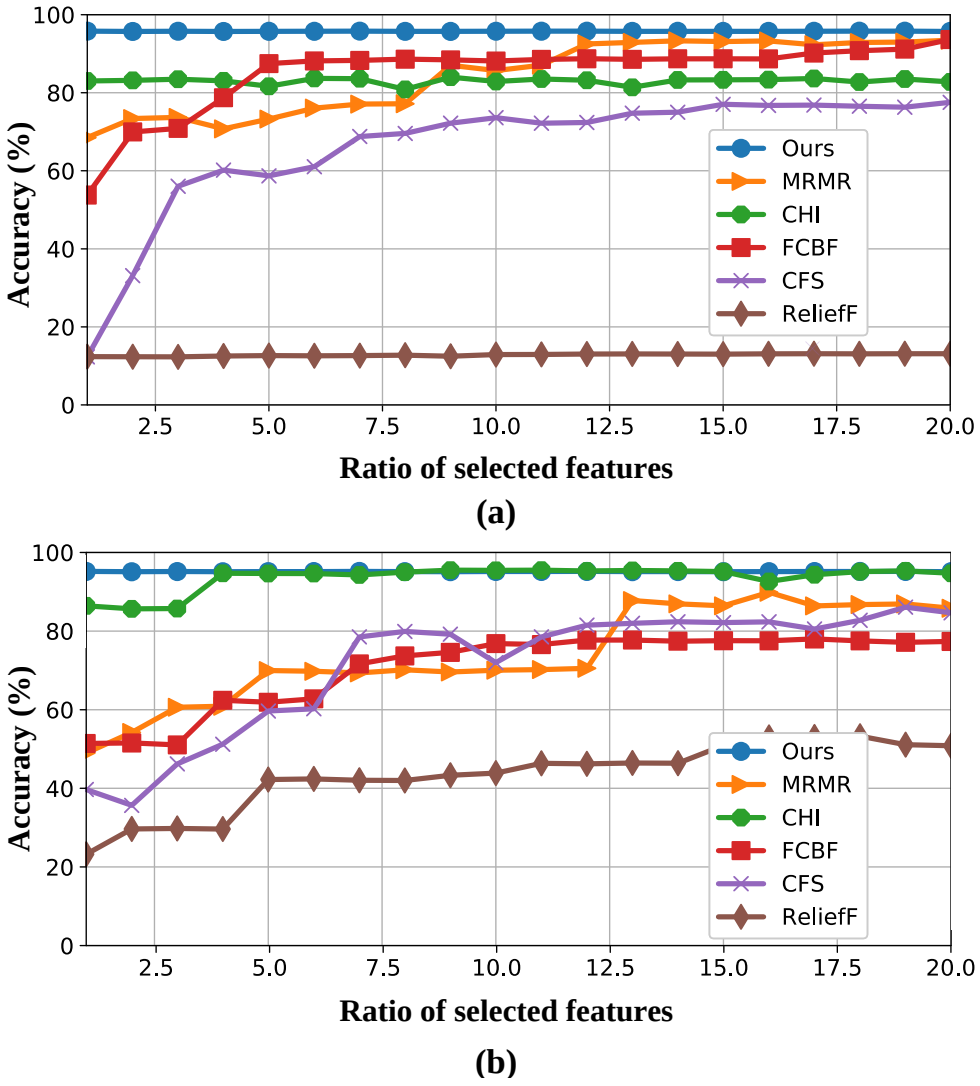

Figure 3.5: The performance of (a) MLP-NTCs and (b) CNN-NTCs based on different schemes.

selected. CFS and ReliefF methods start much lower at around 15%, and with accuracies around 80% and 15% when 20% of features are selected. Fig. 3.5 (b) shows the change of classification accuracy with the increasing number of selected features using dataset 2. Our proposed scheme keep the highest accuracies which is above 95%. CHI method has a lower accuracy in the beginning and can reach the same accuracy with out method when the ratio of selected features is around 5%. The accuracies of FCBF, MRMR, and CFS are much

lower in the beginning, and keep increasing and can reach around 80% by using around 20% of features in packets. The accuracies by using the ReliefF method are the lowest around 20% in the beginning and can reach above 40% when using 20% of features.

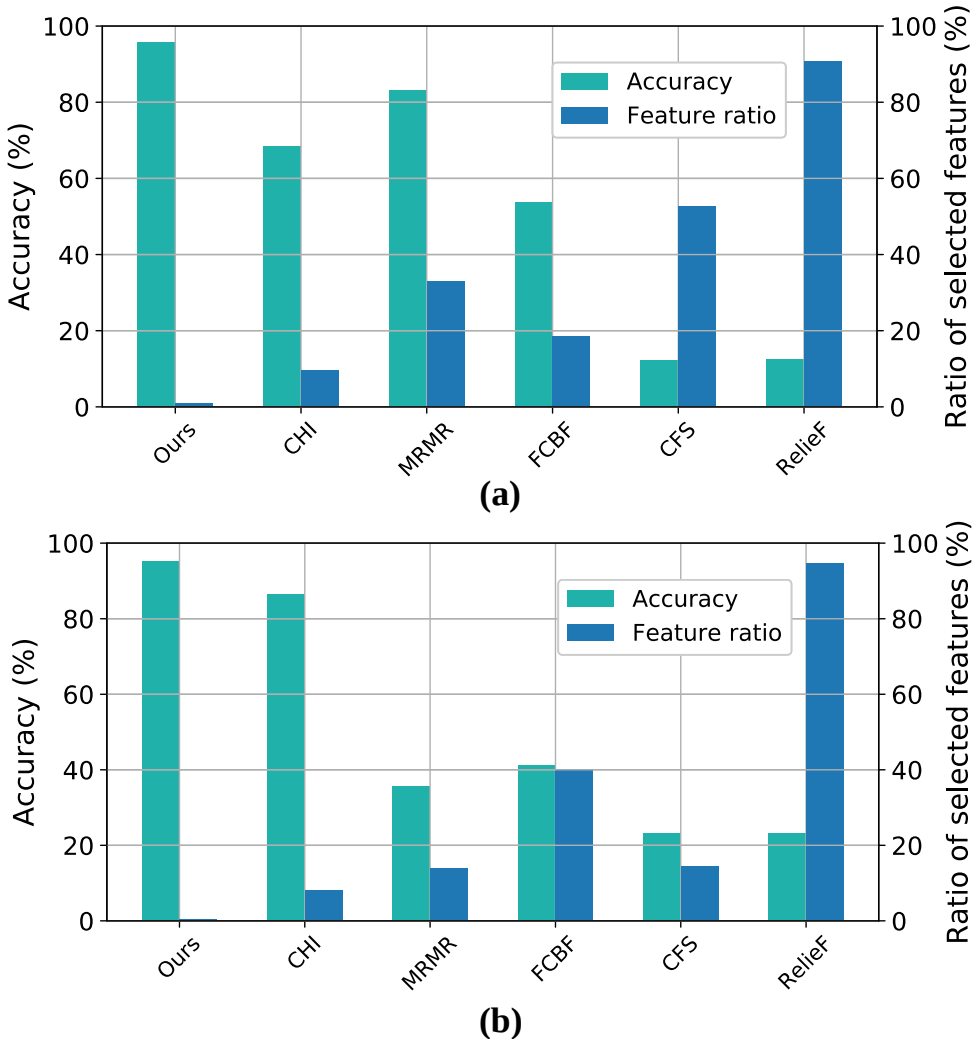

Figure 3.6: The accuracies when using n^* features, and the least ratios of input features for the highest accuracies, respectively, of each method using (a) dataset 1 and (b) dataset 2.

Fig. 3.6 shows the accuracies when using n^* features, and the least ratios of input features for the highest accuracies, respectively. As we can see, with the same number

of selected features n^* our method only requires 1% of the input features to achieve an accuracy above 95%. If the ratio of selected input features is set to be the same with our method, the accuracy with AI-NTC with CHI, MRMR, FCBF, CFS, and Relief methods are round 70%, 82%, 55%, 12% and 12%. To achieve the highest accuracies, top 10%, 30%, 20%, 50% and 90% input features are required by CHI, MRMR, FCBF, CFS and ReliefF, respectively. In Fig. 3.6 (b) shows the comparison using the dataset 2. Our method only requires less than 1% of input features to achieve an accuracy above 95%.. If the ratio of selected input features is set to be the same with our method, the accuracy with AI-NTC with CHI, MRMR, FCBF, CFS, and Relief methods are round 82%, 38%, 41%, 21% and 21%. To achieve the highest accuracies, top 8%, 16%, 40%, 18% and 95% input features are required by CHI, MRMR, FCBF, CFS and ReliefF, respectively.

3.5.4 Evaluation of the Autonomous Model Update Scheme

To evaluate the autonomous model update scheme, some data are collected on a different day for both Amazon music and Discord, to simulate the possible change of feature contributions. Note that only dataset 2 is used to evaluate the model update scheme due to the requirement of collecting new packets. In the evaluation process, there are two scenarios. Each scenario has 40 batches in total. 200 packets in each category are selected for each batch. In this case, 1400 packets with 7 categories are collected in each batch. The first 20 time periods are packets selected from the original dataset, and new packets from Amazon music in scenario 1 and Discord in scenario 2 appear in following 20 batches.

Fig. 3.7 shows the evaluation result of model update scheme. In Fig. 3.7 (a), the accuracy of both AI-NTCs with and without update keep high which is around 95% before seeing new data packets in the first 20 batches. In next 20 batches, the accuracies decrease to

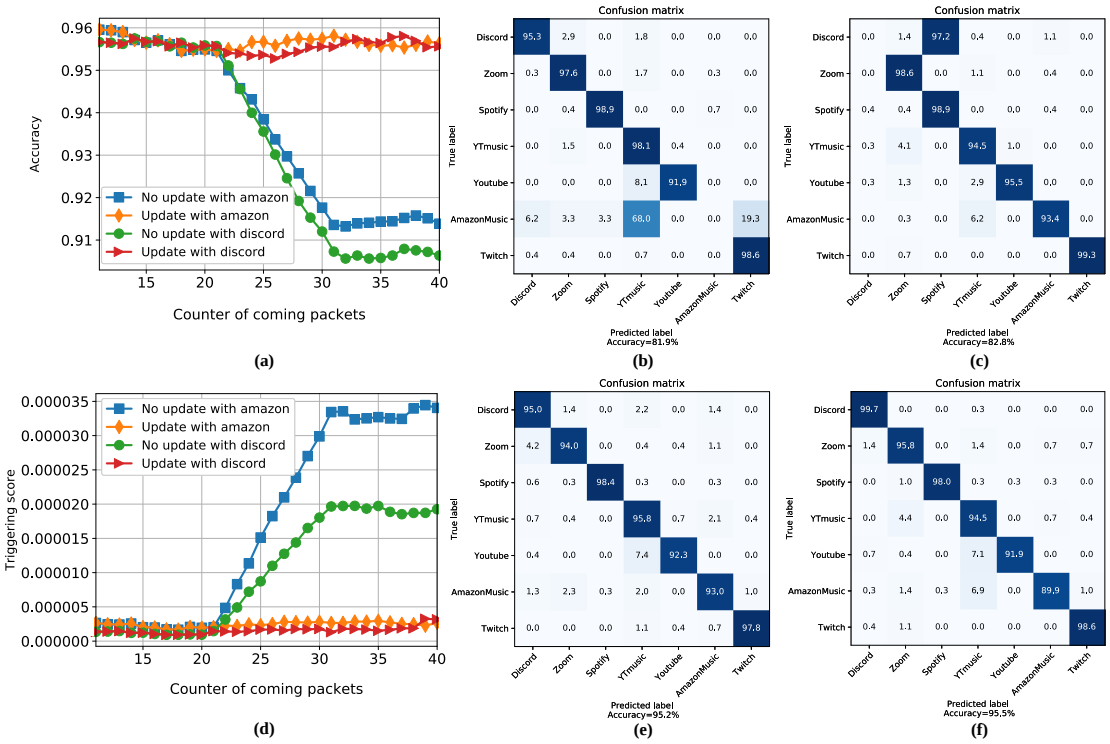

Figure 3.7: The performance of MLP-NTCs without (a) and (b) with adaptive input adjustment, CNN-NTCs without (c) and (d) with adaptive input adjustment.

around 90% due to the change of feature contributions. Fig. 3.7(b) and Fig. 3.7(c) show the confusion matrix of classification performance with new Amazon music and Discord. We can see that the new Amazon music are totally misclassified to YTmusic and Twitch so that the average accuracy decreases to 81.9%. The new Discord are totally misclassified to Spotify so that the average accuracy decreases to 82.8%. Fig. 3.7 (d) shows the triggering score with and without model update scheme. We can see that the feature stability increases because the contributions of each features in new packets are different. After the change of feature stability is detected, a model update is requested for the optimized AI-NTCs. New packets are classified by the benchmarking model to build the new dataset, new feature contributions are extracted to build optimized dataset to update the new AI-NTCs. In

this case, Fig. 3.7(e) shows the classification performance after model update with Amazon music. We can see that the updated AI-NTC can correctly classify the new Amazon music and achieve a high accuracy around 95.2%. Fig. 3.7(f) shows that the updated AI-NTC can correctly classify the new packets from Discord, and it provides a high accuracy around 95.5%.

3.6 Conclusion

Network traffic classification is essential to network measurement, management and security. Recently, Artificial Intelligence (AI) based NTCs are considered as a good choice to identify both clear and encrypted data traffic. To simplify the AI-based NTCs for energy constrained networking edge devices, we proposed an adaptive and lightweight NTC framework. An input feature contribution extraction scheme was proposed to extract the contribution of each feature for determining the optimal set of input features. A practical approach is proposed for acceleration of the optimization process. Moreover, an autonomous update scheme was developed to monitor the change of feature contribution and update AI-NTCs when needed. The results show that the lightweight NTCs can be faster one to two magnitudes without sacrificing classification accuracy. The new AI-NTC updated by proposed model update scheme can still maintain the high accuracy when the feature contribution has changed.

Table 3.2: Overview of the datasets used for evaluation.

Dataset 1

Malware		**Benign**	
Application	**#Total samples**	**Application**	**#Total samples**
Cridex	461,548	BitTorrent	15,000
Geodo	250,000	Facebook	6,000
Htbot	171,569	FTP	360,000
Miuref	88,560	Gmail	25,000
Neris	499,218	MySQL	200,000
Nsis-ay	352266	Outlook	15,000
Shifu	500,000	Skype	12,000
Tinba	22,000	SMB	925,453
Virut	440,625	Weibo	1210,060
Zeus	93,141	WorldOfWarcraft	140,000

Dataset 2

Application	**#Total samples**	**Application**	**#Total samples**
Youtube	26,020	Spotify	45,808
Netflix	341,593	Discord	322,599
Pandora	39661	Amazon video	220,218
Twitch	2103,305	Youtunbe music	505,976

Table 3.3: The evaluation result performance of each type of AI-based NTCs (Cont.)

			Bandwidth-GPU (Gbps)	Bandwidth-CPU (Gbps)	Storage (Mb)	Complexity
			Avg.	Avg.	\	\
Dataset 1	MLP-NTC	Baseline	8.4	0.11	17.24	4.4M
		Update	115.1	11.59	0.02	1.6K
	CNN-NTC	Baseline	0.62	0.009	1.4	41.0M
		Update	87.8	1.38	0.045	15.9K
Dataset 2	MLP-NTC	Baseline	8.06	0.13	16.67	4.25M
		Update	112.2	24.0	0.012	0.29K
	CNN-NTC	Baseline	0.69	0.016	97.4	41.0M
		Update	88.9	1.43	0.045	15.9K

Table 3.4: The evaluation result performance of each type of AI-based NTCs

			Recall (%)			Precision (%)			F1 Score (%)			Accuracy (%)		
			Min.	Avg.	Max.	Min.	Avg.	Max.	Min.	Avg.	Max.	Min.	Avg.	Max.
Dataset 1	MLP-NTC	Baseline	95.1	96.9	98.4	95.1	96.9	98.4	94.9	96.8	98.4	94.8	96.8	98.4
		Update	95.0	96.6	98.4	94.4	96.5	98.3	94.7	96.5	98.4	94.2	96.5	98.4
	CNN-NTC	Baseline	94.8	96.9	98.6	94.8	96.9	98.6	94.8	96.9	98.6	94.6	96.9	98.6
		Update	94.6	96.7	98.5	94.2	96.8	98.6	94.4	96.7	98.6	94.0	96.7	98.4
Dataset 2	MLP-NTC	Baseline	92.7	95.5	97.4	93.0	95.6	97.4	92.8	95.6	97.4	92.4	95.5	97.4
		Update	93.4	95.3	97.1	93.1	95.3	97.1	93.2	95.4	97.3	93.2	95.3	97.4
	CNN-NTC	Baseline	93.8	95.9	97.8	94.1	96.0	97.9	94.1	96.0	97.9	94.0	95.9	97.8
		Update	93.4	96.2	97.8	93.1	95.9	97.7	93.2	96.1	97.7	93.2	95.9	97.6

CHAPTER IV

ADAPTIVE NETWORK TRAFFIC CLASSIFICATION FOR IOT APPLICATIONS

4.1 Introduction

Internet of Things (IoT), a recent communication paradigm which refers to the connection of a number of smart devices such as lights, laptops, sensors, and industrial and utility components, are connected via a network of networks that are forever changing the way we work, live, and play. [94–97]. Users are becoming more dependent on smart and connected devices, while new connected devices are continuously emerging [98]. Fig. 4.1 shows an illustration on the IoT based smart city. Based on the report of CISCO, the number of devices connected to IP networks will be more than three times the global population by 2023, which will be 29.3 billion, up from 18.4 billion in 2018 [99].

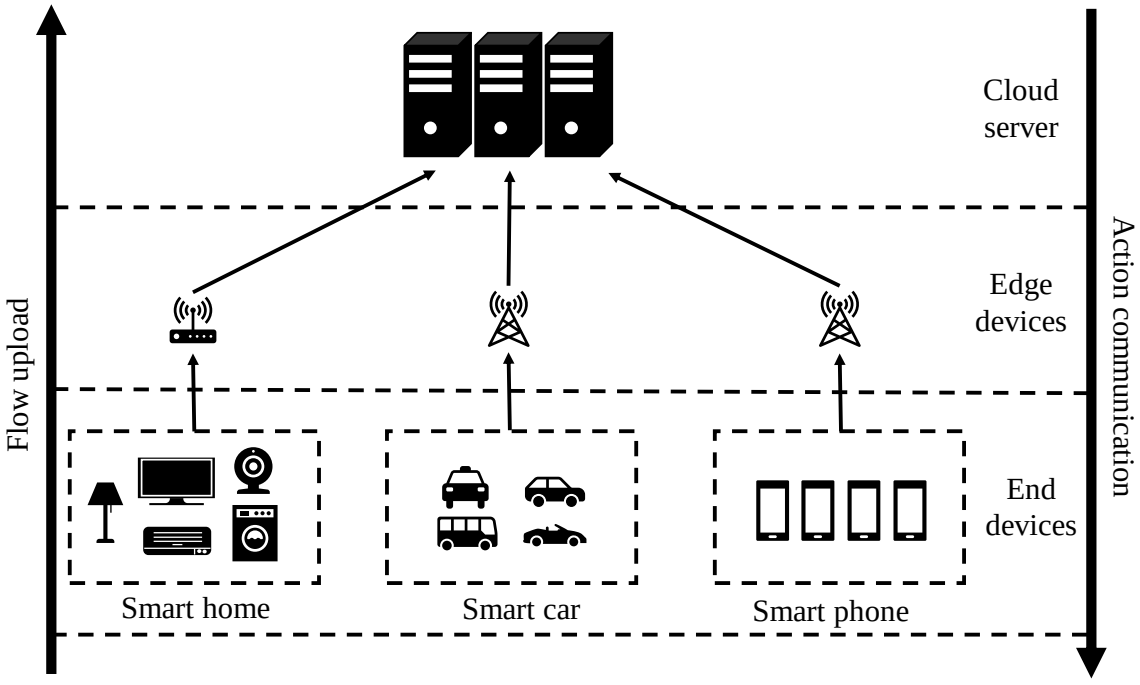

Figure 4.1: IoT based smart city.

With the increasing number of IoT devices comes the rapidly increasing interactive data, fast and accurate classification of these massive data is the key to solving the problems in IoT, e.g., security, network management [72, 100]. Network traffic classification (NTC), especially the Artificial Intelligence (AI) based, plays an important role in both network measurement and management, e.g.,intrusion detection system, quality of service measurement [34, 101, 102], which can be used to solve the IoT problems [103]. First, the security is one of the most problem that specifically benefits from the network traffic classification. An intrusion detection system allows the system to quickly identify the threats and take timely countermeasures [104]. Moreover, the classification of different traffic can help IoT devices improve their work efficiency and quality of service [73].

Traditional approaches to data classification in IoT devices transfer these data to the cloud for storage and processing, and subsequently deliver results to software applications. For instance, data which the control hub receive are transferred to a data centre, possibly thousands of miles away, for storage and processing, and then send the instruction back to the control hub to control the IoT devices. However, as more and more data traffic flows are transmitted from IoT devices, it is undoubtedly a burden for the core computing device, e.g., cloud computing devices deployed with various AI-based models. The high communication overhead means an increasing delay. Edge computing brings a new paradigm that monitor and process the data traffic on the IoT home, etc. Nonetheless, it still has a few challenges. Firstly, it is challenging to deploy the AI-based NTCs on those IoT devices due to the low operating capability. A big and complex structure of NTCs may lead to the data delay, data loss, etc. Secondly, conventional AI-based NTCs are difficult to handle the different network situation, e.g., network congestion by deploying the NTCs.

Recently, researchers deployed the DL-based NTC schemes on IoT based on Convolutional Neural Network (CNN) and Recurrent Neural Network (RNN) in [103,105,106], which show good classification performance. However, these proposed DL-based NTC schemes are complex with CNN and RNN, where the computation efficiency cannot be guaranteed. It takes both time and storage space for the neural networks to perform NTC. As a result, these DL-based NTC models are not piratical a few challenges edge devices with low computational profile. For speeding up the researches have been studying both the design of the network operations and network architectures. New designs of neural network, e.g., MobileNet, were proposed to enable the deployment of deep learning models on the portable devices [86]. These light-weighted network models can perform feature extraction and classification more efficiently than a standard neural networks, e.g., CNN, MLP. However, the classification performance is not ideal compared with the standard ones [107]. Alternatively, pruning is proposed to simplify the DL models from the aspect of the network architecture. It is a scheme designed to reduce the number of network parameters, i.e., weights and basis, to boost the operation efficiency of neural networks. A fine tuning is usually applied after pruning to recover the decreased performance. However, fine-tuning can be challenging to IoT devices. Moreover, the non-returnability of pruning leads to irreversibility of the DL-based NTCs, which is not suitable for the different network situations.

In this chapter, we propose a lossless optimization of an MLP based NTCs to support a satisfactory classification performance with less network parameters. In specific, an MLP based NTC is initialized, namely baseline MLP-NTC. One node is removed from the target hidden layer which is based the calculated probability for each hidden layer. The optimization stops when the classification accuracy reach the target value (e.g., 1-ϵ_0). The optimization of the neural network does not require extra training process to limit the over-

head. Moreover, we further propose two types of adaptive optimizations for MLP based NTC. The two optimizations can select the different parts in the MLP based NTC to complete the classification according to different network situations. The MLP based NTC is continued removing the node until the classification accuracy reach the target value (e.g., 1-ϵ_u). During each removal, the existing node combination and corresponding classification accuracy is stored. the MLP based NTC can choose the different node combination adaptively to provide the different support bandwidth for the different network situations. Evaluations are conducted based on a MLP based NTC models, which is implemented using an open dataset (ICSX-VPNnonVPN [108]) and our new dataset [67, 87]. The results demonstrate that the updated MLP based NTCs can speed up the baseline MLP based NTCs by 5 times using both proposed adaptive methods on CPU implementations without sacrificing the classification accuracy. The adaptive MLP based NTCs can speed up the baseline MLP based NTCs an average of 15/11 times using adaptive optimizations and M/D/1 based adaptive optimizations with a 4%/2% decrease of classification accuracy on CPU implementations. Moreover, we implement the proposed lossless optimization and adaptive optimization on Raspberry Pi 3B. The evaluation results demonstrate that the updated MLP based NTC and adaptive MLP based NTC can speed up an average of 10/20 times than the baseline MLP based NTC. Thus, the major contributions of this chapter sum up as follows.

- We propose a lossless optimization scheme to reduce the parameters in MLP based NTC for speeding up without sacrificing the classification accuracy.
- Secondly, we propose two adaptive optimizations for MLP based NTCs to satisfy with the different network situation by changing the structure of MLP based NTCS. To

the best of our knowledge, this is the first time that the concept of adaptive AI-based NTC for edge devices in IoT to handle the different network simulations.

The organization of this chapter is as follows. Related works are discussed in Section 4.1. We introduce the lossless optimization for the MLP-based NTC in Section 4.2. In Section 4.3, we introduce two types of the adaptive optimization for MLP based NTC. Section 4.4 shows the simulation results and discussions, followed by the conclusions.

4.2 Related Work

4.2.1 Traditional Machine leanring based Network Traffic Classifier

Recently, researchers employed machine learning algorithms as classifier to analyze and classify the IoT traffic. ML algorithms, e.g., support vector machine, decision tree, etc, are adopted to reason the association between the features and the corresponding traffic categories [21,22,109]. In specific, the authors in [109] proposed a intrusion detection system to distinguish the distributed denial-of-service attack based on a variety of conventional machine learning algorithms. The accuracies of machine learning based classifiers ranged from 91% to 99%. In [21], the authors proposed a intrusion detection system based on KNN classification algorithm, which can separates abnormal packets and normal one by observing the the abnormal behaviors. In [22], the authors proposed practical framework called CutSplit based on decision Tree, which can adaptively exploit the benefits of the cutting and splitting techniques. However, the ML-based NTC performance can hardly be guaranteed and the classification accuracy may be unpredictable due to the varying feature selection of the massive data in IoT.

4.2.2 Deep leanring based Network Traffic Classifier

Efforts have been made based on DL-based models, e.g., Convolutional Neural Network (CNN), Multilayer Perceptron (MLP), Recurrent Neural Networks (RNN), etc. for building network traffic classifiers and solve the IoT problems [71, 110, 111] . For instance, In [71], the authors proposed a CNN based malware traffic classification model which transform the raw packet data to the images. The average accuracy of their proposed three classifiers (e.g., binary classifier, 10-class classifier, 20-class classifier) is 99.41%. In [110], the authors proposed a deep learning based packet classification by combining the CNN and RNN for IoT traffic. Their method gives the accurate rate above 96%. Moreover, the authors in [111] proposed an application-based network traffic classifier with MLP model to classify the data traffic. In [73], the author proposed a IoT traffic classification mechanism based on capsule network for smart cities. Ten-class aggressive traffic and ten-class normal traffic are classified and the average accuracy reach above 99%. However, these proposed networks are complex which require numerous computational resources and storage spaces which could be a challenge on the edge devices in IoT.

In this chapter, we propose a lossless optimization scheme for MLP based NTC to remove the nodes in the NTC for speeding up without sacrificing the classification accuracy. Retraining and fine-tuning is not needed in this approach. Moreover, we propose two adaptive optimization scheme to further speed up the NTC to deal with the different network situations, e.g., congestion control .

4.3 Lossless Optimization of an MLP based NTC

4.3.1 MLP based NTC

MLP based NTC is used due its accurate classification accuracy and fast operation speed than others [63, 112]. In a standard MLP based NTC, the input layer is the packet bytes, one or more hidden layers are applied after that for extracting the features. And the output is the APP classes. The obtained data is subjected to nonlinear transformation using an activation function, e.g., sigmoid, ReLU, etc. Softmax and classification layer outputs the results. Suppose there are H hidden layers in an MLP-based NTC, l_i is number of nodes in hidden layer i, n_i^j is the j-th node in hidden layer i, an MLP-based NTC is shown in Fig. 4.2.

The data packet after pre-possessed e.g., parsing, truncating/padding, normalization is used as the inputs MLP based NTC [13]. The hidden layers are applied after the input layer for extracting the features. Each hidden layer (e.g., the i-th layer) has several neurons that are connected with the adjacent layers, computed as:

$$\mathbf{z}_i = \sigma(\mathbf{W}_i \cdot \mathbf{z}_{i-1} + \mathbf{b}_i), \tag{4.1}$$

where $\mathbf{W}_i$ and $\mathbf{b}_i$ are weight matrix and bias vector of hidden layer i, respectively. $\mathbf{z}_i = [n_i^1, n_i^2, ..., n_i^{l_i}]$ is the output vector of hidden layer i (e.g., n_i^j is the j-th node in hidden layer i). As shown in Fig. 4.3, node n_i^j is calculated by all the nodes in the hidden layer $i-1$ and the weights $w_i^{j,1}$ to $w_i^{j,l_{i-1}}$, and it is also used to calculate all the nodes in the hidden layer $i+1$ with the weights $w_{i+1}^{1,j}$ to $w_{i+1}^{l_{i+1},j}$. $\sigma(\cdot)$ is the activation function. For instance, Rectifier Linear Units (ReLU) can provide a faster training process and help avoid gradient

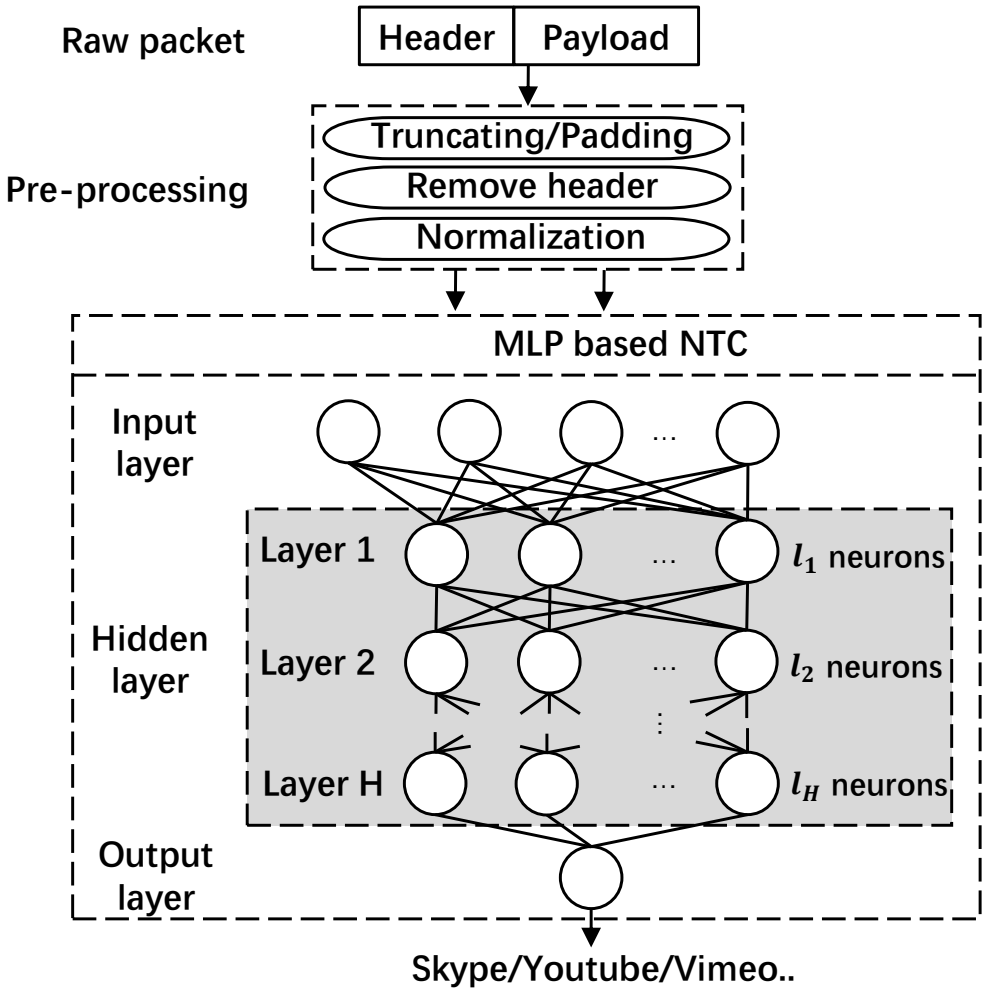

Figure 4.2: Packet layer MLP based network traffic classifier.

vanishing problem compared with other activation functions [31]. The activation process produced by ReLU can be denoted as:

$$n_h^j = \max[0, n_i^j], \tag{4.2}$$

The output layer consists of softmax and classification. softmax is for calculating the cross-entropy for the final classification after the last hidden layer, which is computed as

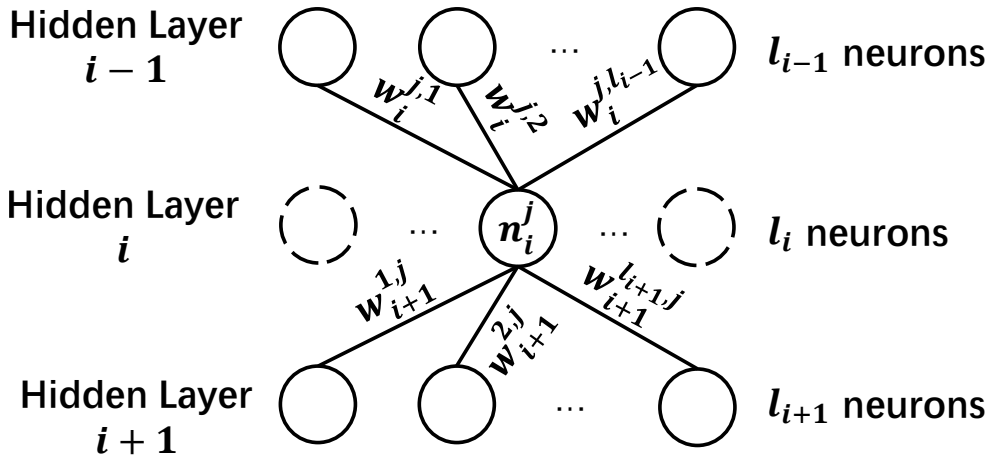

Figure 4.3: The connections of the node n_i^j in MLP based classifier.

follows:

$$s_H = [s_H^1, s_H^2, ..., s_H^N] = \frac{\exp(\mathbf{z}_H)}{\sum_{j=1}^{N} \exp(n_H^j)}, \tag{4.3}$$

where s_H is the categorical probability for each class, N is the the number of APPs.

4.3.2 Node Importance

In this section, we introduce the node importance. Given a MLP based NTC M which is trained by dataset D, M^* is the target model. The importance of nodes in M after training are distinct [66]. Based on this property, removing the nodes that are considered to be the low importance in layers have the least impact on the performance of the entire model Compared to randomly chop off the nodes.

In this case, we defined the node importance I_i^j which considers both the mean and variance value of the weights in classifier M,which can be denoted as:

$$I_i^j = (\text{Mean}(\sigma(w_i^{j,k})) + \text{Mean}(\sigma(w_{i+1}^{p,j})) + b_i^j) \cdot (\text{Var}(\sigma(w_i^{j,k})) + \text{Var}(\sigma(w_{i+1}^{p,j}))) \tag{4.4}$$

where the I_i^j is the importance of the j-th node in hidden layer i. The I_i^j is related to the values of the weights and bias which used to calculate n_i^j (i.e., $w_i^{j,1}$ to w_i^{j,l_i}) and the weights used to calculate the $\mathbf{z}_{i+1}$ in next layer with it (i.e., $w_{i+1}^{1,j}$ to $w_{i+1}^{l_{i+1},j}$) because all these weights and bias will be removed if this node is removed. σ is the normalization function for eliminating the dimension between different data, facilitating data comparison and co-processing. All the weights of each layer will be normalized independently. In this work, Min-Max normalization is applied for illustration. Other normalization schemes such as Z-score can be easily applied. The normalized data is in the same order of magnitude, which can eliminate the effect of the dimensions and units between the indicators and improve the comparability between different data indicators. The original weight matrix $\mathbf{W}$ is linearly mapped to a normalized value $\mathbf{W}'$ which is in the range of 0 to 1. The Min-Max normalization is shown in the Eq. (4.5).

$$\mathbf{W}' = \left| \frac{\mathbf{W} - \min(\mathbf{W})}{\max(\mathbf{W}) - \min(\mathbf{W})} \right|, \tag{4.5}$$

where $\min(\mathbf{W})$ and $\max(\mathbf{W})$ are the minimum and maximum values of the attributed weight matrix $\mathbf{W}$, respectively. The I_i^j is calculated by Eq. (4.4) and sorted After normalization. The smaller I_i^j represents less importance of the node. Thus, the node importance locations are stored in $\mathbf{S}$, i.e., $\mathbf{S} = \left\{ [r_1^1, ..., r_1^{l_1}], ..., [r_H^1, ..., r_H^{l_H}] \right\}$, where r is the node importance location of each node in each layer based on I_i^j (e.g., r_1^1 is the location of the least important node in first hidden layer). The node removal is based on the stored $\mathbf{S}$.

4.3.3 Model initialization and optimization

It is important to decide the parameters of initial MLP architecture, e.g., number of layers, number of nodes, etc. based on [90]. Three hidden layers are chosen for the baseline MLP-based NTC implementation. As mentioned in [84], the number of the nodes in hidden layer can be initialized as 2/3 (or 70% to 90%) of the size of the input layer. In [113], the number of nodes in each hidden layer should be less than twice of the input layer. Therefore, we suggest initializing a three-layer MLP architecture which has 2/3 of the input size in each hidden layer.

The optimization includes two parts: fast compression and node removal. In fast compression, the neural network structure is compressed fuzzy with the fewest nodes in the neural network before the node removal. After the average accuracy is stored for future comparison, half of nodes in each layer are removed for the next training and then the accuracy is compared with the previous stored accuracy. Once the accuracy starts decreasing, the nodes no longer continue to be removed and the neural network structure is the structure before this removal. Then the node removal is designed for removing the nodes in hidden layers to speed up without sacrificing the classification accuracy, i.e., $A_M - A_{\min} \leq \epsilon_0$, where ϵ_0 is the number that is allowed to reduce on accuracy. In node removal, a node in one hidden layer is removed in every iteration. To decide the layer which is removed a node in current iteration, we propose a method which adjust the importance of the layers based on the distribution of the weights in each layer. Straightforwardly, a large number of values close to 0 in the weight leads to a Distribution with low variance. Lower impact occurs when a node in the layer with sharp distribution is removed. In this case, a sharp distribution based on normal distribution is designed to imitate the distribution that need to be cut. Note that lots of distribution methods (e.g., normal distribution, random distri-

bution, etc.) can be used to generate this curve. To compute the similarity between the generated distribution and the distribution of weights in each layer, the Kullback-Leibler (KL) divergence is adopted [114], which can be denoted as:

$$d = D_{\mathrm{KL}}(P(\mathbf{W}_i) \mid\mid Q) = \sum_{j=1}^{N} P(\mathbf{W}_i(y_j)) \cdot \log \frac{P(\mathbf{W}_i(y_j))}{Q(y_j)}, \tag{4.6}$$

where the $P(\mathbf{W})$ is the distribution of the weights, and Q is the generated distribution. KL divergence is actually the expectation of the logarithmic difference between the distribution $P(\mathbf{W})$ and the distribution Q of the data. A vector includes the probability for each layer is calculated the by d, which can be denoted as:

$$\mathbf{d}^{'} = [d_1^{'}, ..., d_H^{'}] = \frac{\mathbf{d}}{\sum_{j=1}^{H} d_j}. \tag{4.7}$$

the least important node in target layer is removed based on $\mathbf{S}$ (e.g., r_i^j-th node is removed when the i-th layer is j-th chosen to remove the node). The node removal is stopped when the current accuracy reach $1 - \epsilon_0$. The whole lossless optimization is shown in Alg. 5.

4.4 Adaptive optimization for MLP based NTC

In this section, we introduce adaptive method for MLP based NTC in IoT device to deal with the different network situations, e.g., network congestion, etc. Different network traffic flow leads to the different requirements for NTC. A large network traffic flow may result the occurrence of network congestion, which is the downgrade of the network transmission performance due to the limited resources of the storage when the number of packets transmitted in the packet-switched network is too large [115]. Network congestion, may lead to the loss of data packets, time delay, etc. The optimized NTC M^* may not avoid the network congestion when the network traffic flow exceed the limitation of it. That means M^* need to further speed up by continuing removing the nodes. Meanwhile, the node removal is

Algorithm 5: LOSSLESS OPTIMIZATION OF AN MLP BASED NTC CLASSIFIER

Input: M,$\mathcal{D}$
Output: M^*

Initialization;
while *1* **do**
    Initialization of M based on L and $\mathcal{D}$;
    Test M to evaluate the accuracy A;
    if $A < \mathcal{A}$ **then**
        $L = L \times 2$;
        break
    else
        $\mathcal{A} \leftarrow A$;
        $L \leftarrow L/2$;
    end
end
Build M with the layer number L;
Train M with the dataset $\mathcal{D}$;
Test M and calculate the accuracy; $A \rightarrow \mathcal{A}$;
Calculate the node importance I based on Eq. (4.4);
while *($A \geq \mathcal{A} - \epsilon_0$)* **do**
    for k = *1:N* **do**
        Calculate the d_i based on Eq. (4.6);
        Calculate the probability $d_i^{'}$ based on Eq. (4.3);
    end
    Choose a layer to remove a node in M based on $d^{'}$;
    Test M and calculate the accuracy A;
end
$M^* \leftarrow M$

irreversible, which means the loss of accuracy is irreversible even the network traffic flow is small. In this case, we propose two types of the MLP-based adaptive pruned classifier for different IoT devices. The architecture in AI-based NTC is adaptive which is based on the network congestion status.

4.4.1 Jump-type adaptive NTC

Here we introduce the proposed jump-type adaptive NTC. We assume that the only thing which can be monitored for those IoT devices with low operating capability is the number of packet in buffer. When the packets in the buffer keeps increasing, the classifier need to be continued removing the nodes with sacrificing the classification accuracy, while the nodes that are already removed are re-add to increase the classification accuracy after the packets in buffer are short enough. In jump-type adaptive NTC, the nodes are continued removing, namely continued node removal. In every round, the index number, classification accuracy and the number of removed nodes in each layer are stored in $\mathbf{P}$, i.e., $\mathbf{P}_j = \{j, A_j, \mathbf{k}_j\}$, where j is the index number, $\mathbf{k}_j$ is vector of the number of the removed nodes in each layer after the j-th continued node removal, i.e., $\mathbf{k}_j = (k_1^{(j)}, ..., k_H^{(j)})$. Let $\mathcal{P}$ as a set of the $\mathbf{P}$, i.e., $\mathcal{P} = \{\mathbf{P}_1, \mathbf{P}_2, ..., \mathbf{P}_m\}$, where m is round number that the classification accuracy of NTC reach $1 - \epsilon_u$.

Fig. 4.4 shows the storage of the models after the all $\mathbf{P}$ are generated. Note that with increasing of j, the optimized NTC becomes more lightweight, while the classification accuracy is uncertain. In this case, those $\mathcal{P}$ with more nodes (i.e., smaller j) and lower classification accuracy are removed, which is show in in Fig. 4.5. Baseline accuracy A_o is set to the classification accuracy of the last model $\mathbf{P}_m$. Classification accuracy of every $\mathbf{P}_j$

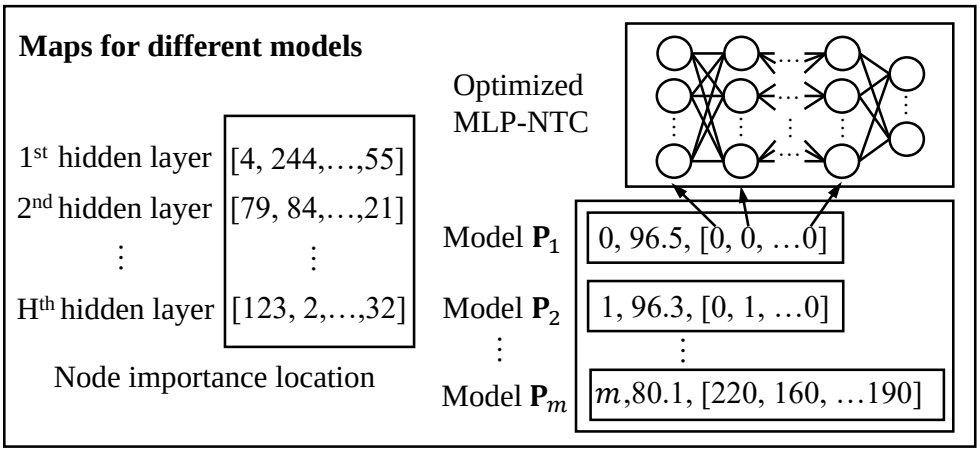

Figure 4.4: The framework of multiple model storage.

is compared with the last $\mathbf{P}_{j+1}$. $\mathbf{P}_j$ is kept and A_o is set A^{j-1} if the accuracy of $\mathbf{P}_{j-1}$ is larger, otherwise, $\mathbf{P}_j$ is dropped.

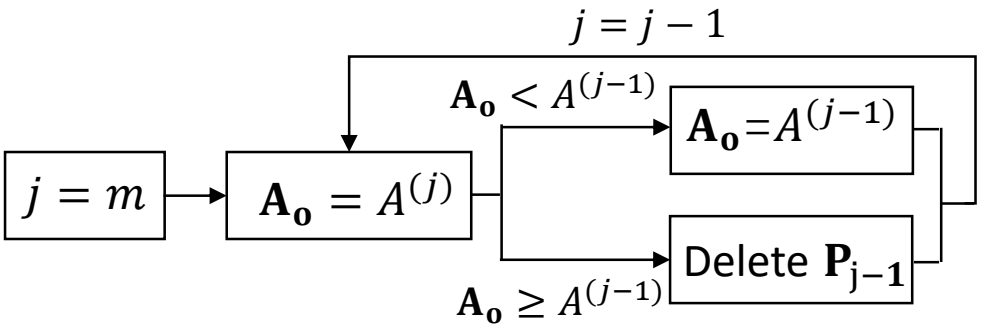

Figure 4.5: The selection method for $\mathbf{P}$

Baseline accuracy A_o is set to the classification accuracy of the last model $\mathbf{P}_m$. Classification accuracy of every $\mathbf{P}_j$ is compared with the last $\mathbf{P}_{j+1}$. $\mathbf{P}_j$ is kept and A_o is set A^{j-1} if the accuracy of $\mathbf{P}_{j-1}$ is larger, otherwise, $\mathbf{P}_j$ is dropped.

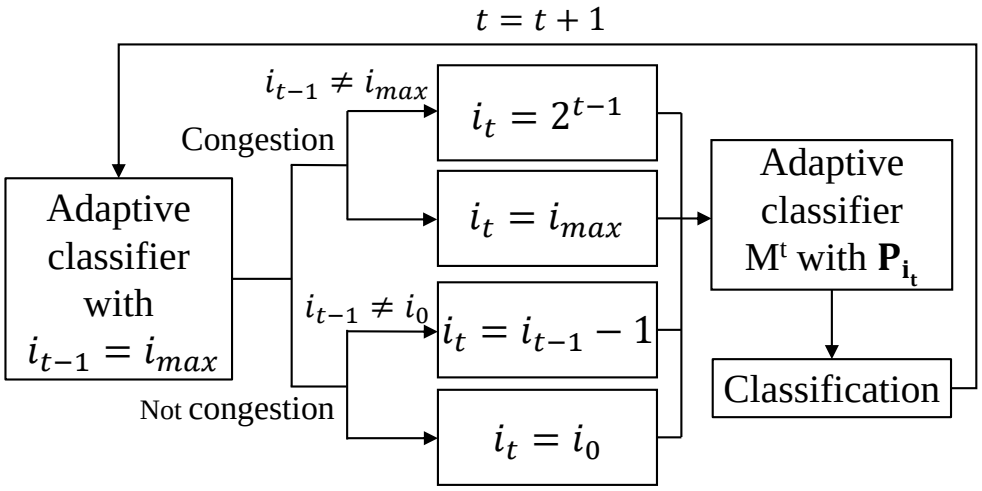

Figure 4.6: Model selection scheme

Fig. 4.6 shows the flow chart of the jump-type adaptive NTC used in the real-time network. Note that i is the index number of $\mathbf{P}$ in $\mathcal{P}$. M^t is the optimized NTC based on $\mathbf{P}_{\mathbf{i_t}}$. Firstly, the system goes to the fast start, which means the network is set to the network with i_{max}, and the choice of the neural network in every period is based on the congestion situation. Here we defined the packet number in buffer at time t is F_t. In every period t, the $\mathbf{P}_{i_t}$ with $i_t = i_{t-1} - 1$ is applied to modify the structure of MLP-based NTC until the i_t equals to i_0 if the network is not congested in period $t-1$ (i.e., F_{t-1} is smaller or equal with the F_{t-2}). Otherwise, the structure of MLP-based NTC is modified based on $\mathbf{P}_{i_t}$ with $i_t = 2^{i_{t-1}}$ to fast increase the processing speed to solve the problem of the network congestion. The jump-type adaptive NTC is shown in Alg. 6.

4.4.2 Precise-type adaptive MLP-NTC

In this subsection, we introduce the other adaptive optimization for MLP based NTC, namely precise-type adaptive MLP-NTC. In last section, we consider the only thing which can be monitored is the status of the buffer for those IoT device with low computational

Algorithm 6: Jump-type adaptive NTC

Input: $M^*, \mathcal{P}, \mathbf{S}, F_{t-1}, F_{t-2}$
Output: M_t

Initialization;
$\mathcal{P} \leftarrow \emptyset$;
// Continued node removal;
while *($A \geq 1 - \epsilon_u$)* **do**
  $\mathbf{P}, \mathbf{k}, \mathbf{d}^{'} \leftarrow \emptyset$;
  Count index number j;
  for *h = 1:H* **do**
    Calculate the $d_h^{'}$ based on Eq. (4.6);
    $\mathbf{d}^{'} \leftarrow d_h^{'}$;
  end
  Remove a node in one layer chosen based on $\mathbf{d}^{'}$ in M^*;
  Record the number of removed node in each layer to $\mathbf{k}_j$;
  Test M^* and calculate the accuracy A;
  $\mathcal{P} \leftarrow \mathbf{P} = \{j, A_j, \mathbf{k}_j\}$;
end
// Selection method of $\mathbf{P}$;
for $j = 1; j \leq m; j{+}{+}$ **do**
  if $A_j < A_{j+1}$ **then**
    Drop $\mathbf{P}_j$;
  else
    $A_o = A_j$;
  end
end
// Adaptive NTC in real-time network;
while *1* **do**
  if $F_{t-1} \leq F_{t-2}$ **then**
    $i_t \leftarrow i_{t-1} - 1$;
  else
    $i_t \leftarrow 2^{i_{t-1}}$;
  end
  $\mathbf{S}^* \leftarrow$Truncate $\mathbf{S}$ based on $\mathbf{P}_{\mathbf{i_t}}$, $\mathbf{P}_{\mathbf{i_t}} \in \mathcal{P}$.
  Reconstruct M^* based on $\mathbf{S}^*$;
  $M_t \leftarrow M^*$
end

power. If those devices can also collect the coming packet number in a period, the selection of $\mathbf{P}$ can be more accurate. That means the most suitable structure of the classifier can be chosen based on the collected coming packet number. According to this hypothesis, we propose the precise-type adaptive MLP-NTC based on the M/D/1 queuing model [116].

In the proposed precise-type adaptive MLP-NTC based on M/D/1 model, the processing speed, which is the average processing speed of the pruned neural network to classify the packets in a time period t,is also evaluate and stored in $\mathbf{P}$ during the continued node removal. i.e., $\mathbf{P}_j = \{[j, A_j, s_j, \mathbf{k}_j\}$, where s is the processing speed. Assuming in period t, the arrival packets numbers are λ_t. Here we set a delay period to solve the problem that the collected λ can not be used to calculated at the same time. In the next period $t+1$, According to the collected λ_t and F_s, which is the expected average queue length we set, we calculate the suitable processing speed μ_{t+1} to process λ_t the for keeping the F_s fixed based on Eq. (4.8).

$$s_{t+1} = \frac{t \cdot Fs}{\lambda_t + \lambda_t \cdot Fs} \tag{4.8}$$

the $\mathbf{P}$ with the lowest difference with the calculated s_{t+1} is found based on s_{t+1}, and the structure of M_t is changed based on $\mathbf{S}$ and $\mathbf{P}$. whose s has the lowest difference with the calculated $s_t + 1$. Meanwhile, the λ_{t+1} is collect at the same time for calculating the s_{t+2}.

4.5 Simulation Results and Discussions

4.5.1 Dataset for Evaluation

Parts of the dataset for the evaluation is selected from a open dataset (ISCXVPN2016 [117]). A total of 206,688 packets, including Skype, TorTwitter, Netflix, etc. Those applications are encrypted by different security protocols, e.g., HTTPS, SSL, SSH, etc. A total of 3,605,180 packets from 8 mobile applications are included in the mobile application dataset [118]. We

Algorithm 7: Precise-type adaptive MLP-NTC

Input: $\lambda_{t-1}, \mathcal{P}, M^*$
Output: S_t

Initialization;
$\mathcal{P} \leftarrow \emptyset$;
// Continued node removal;
while *($A \geq 1 - \epsilon_u$)* **do**
    $\mathbf{P}, \mathbf{k}, \mathbf{d}^{'} \leftarrow \emptyset$;
    Count index number j;
    for h = *1:H* **do**
        Calculate the $d_h^{'}$ based on Eq. (4.6);
        $\mathbf{d}^{'} \leftarrow d_h^{'}$;
    end
    Remove a node in one layer chosen based on $\mathbf{d}^{'}$ in M^*;
    Record the number of removed node in each layer to $\mathbf{k}_j$;
    Evaluate the model to calculate the processing speed s_j; Test M^* and calculate the accuracy A;
    $\mathcal{P} \leftarrow \mathbf{P} = \{j, A_j, s_j, \mathbf{k}_j\}$;
end
// Selection method of $\mathbf{P}$;
for $j = 1; j \leq m; j{+}{+}$ **do**
    if $A_j < A_{j+1}$ **then**
        Drop $\mathbf{P}_j$;
    else
        $A_o = A_j$;
    end
end
// Precise-type adaptive MLP-NTC in real-time;
while *1* **do**
    Collect the λ_t;
    Calculate s_t based on Eq. (4.8) with λ_{t-1};
    $i_t = \arg\min(s - s_t)$, $s \in \mathbf{P} \in \mathcal{P}$;
    $S^* \leftarrow$ Truncate $\mathbf{S}$ based on $\mathbf{P}_{\mathbf{i_t}}$, $\mathbf{P}_{\mathbf{i_t}} \in \mathcal{P}$;
    Reconstruct M^* based on S^*.
    $M_t \leftarrow M^*$
end

capture the packets including Youtube, Spotify, Netflix, etc by using an Android emulator on a desktop to emulate the environment of a mobile device. The details of the dataset is shown in the Table 4.1. Note that the first 24 bytes of the packets in both datasets are removed to focus on the encrypted payload only.

Table 4.1: Summary of the dataset used for evaluation.

ISCX VPN-nonVPN dataset			
Application	**Total samples**	**Application**	**Total samples**
Email clients	4,417	SFTP(download)	4,729
Facebook chat	5,527	SCP(download)	15,390
Netflix	51,932	Skype file	4,607
TorTwitter	14,654	Vimeo	18,755
VOIPbuster	35,469	Youtube	12,738
Mobile application dataset			
Youtube	26,020	Spotify	45,808
Netflix	341,593	YTmusic	505,976
Pandora	38,661	AmazonPrimeVideo	220,128
Twitch	2,103,305	Discord	322,599

The details of the built classification models are summarized in the Table 4.2. A 3-layers MLP based NTC is used as the baseline. The input is the full packets with 1 by 1456. The nodes in each layers are set to 970, which is the two third of the inputs. We also implement the network traffic classifier named Deep Packet which is a 1D-CNN based packet classifier that designed by [105], namely DP-NTC. The input is a packet vector with a dimension of 1 $\times$ 1456 bytes. There are 2 convolutional layers with 200 kernels whose

Table 4.2: Specification of built deep learning based network traffic classifier.

Classifier type	DP-NTC [7]	CNN-NTC [8]	Baseline MLP-NTC
Input size	1×1456	16×16	1×1456
Convolutional kernel size	(1,5)×200 (1,4)×200	(3,3)×64	-
Activation function	ReLU	ReLU	ReLU
Pooling layer size	(2,2)	(2,2)	-
Dense layer size	9	9	970 970 970

kernel size are 1 by 4 and 1 by 5, respectively, followed by a max pooling layer. Then the feature map is fatten as a one-dimensional vector and processed by the followed three dense layers. Here the dropout is applied to avoid the over fitting [119]. In addition, we also build a 2D-CNN based network traffic classifier which is designed by [62], namely CNN-NTC. In CNN-NTC, the input is a 2D tensor which is 16 by 16, reshaped by the first 256 bytes in every packet. There are two conlolutional layers whose kernel size is 5 by 5 with 256 kernels. After the convolutional layer, the feature map is fatten to a 1-dimensional vector. All the three NTCs have a softmax and classification layer is applied here for the final classification. The evaluation and simulation of the proposed scheme are conducted on a workstation that is equipped with an Intel® Core(TM) i7-6700 CPU @ 3.40GHz, 32.0 GB

RAM @ 2133 MHz, 1 TB SSD. Python with Pytorch running in Windows 10 is used for the scheme implementation and evaluation.

To evaluate these deep learning based scheme in terms of the overall performance, we define detection performance metrics such as recall (R), Precision (P), classification accuracy (ACC) and F1 score (F1), which are formulated in Eq. (4.9)-Eq. (4.12). Recall indicates the percentage of the packets in an application category that are correctly classified. Precision is the percentage of the packets that are exactly attributed to the targeted network application. Accuracy is the percentage of the packets that are correctly classify to their categories. And F1 score is a comprehensive metric for evaluating the performance which considers both value of the recall and precision.

$$\mathrm{P} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}}, \tag{4.9}$$

$$\mathrm{R} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}, \tag{4.10}$$

$$\mathrm{F1} = \frac{2 \cdot \mathrm{R}}{\mathrm{P} + \mathrm{R}}, \tag{4.11}$$

$$\mathrm{ACC} = \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{TP} + \mathrm{TN} + \mathrm{FP} + \mathrm{FN}}, \tag{4.12}$$

where TP (true positive) is the number of target data that are correctly classified, TN (true negative) is the number of other categories which are correctly classified, FP (false positive) is the number of other categories which are incorrectly classified to the target class, FN (false negative) is the number of target data which are incorrectly classified as other classes. In addition, the speed which is the number of packets the model can process in every millisecond and the bandwidth are also used to evaluate the models.

4.5.2 Evaluation Results

In this subsection, the training and testing processes of these proposed and built packet classifier models are introduced. we evaluated the modes by using the two datasets respectively. In ISCX-VPNnonVPN dataset, 9 applications are chosen, including Email-client, Facebook chat, SCP Download, SFTP, Skype, Twitter, Vimeo, Voipbuster and Youtube. Without loss of generality, balanced subsets are generated by randomly choosing an equal number of packets from each of the 9 applications in the database to train the classifiers. In each chosen subset, we randomly choose 2000 packets for training and 500 for testing. Thus, there are a total of 18000 packet in the training dataset and 4500 in testing dataset. In mobile applcation dataset, the packets from 8 applications which includes Youtube, Netflix, Spotify, YT music, Amazon Prime Video, Pandora, Twitch and Discord is used to train and test the built classifiers. We randomly choose 3000 and 500 packets to build the training and testing dataset for each class. Thus, there are a total of 24000 and 4000 packets in the training and testing dataset. We train all the models in 300 epochs. The learning rate is set 0.005 and dropped 0.01% after every 10 epochs. In the testing part, we tested the built NTCs by CPU to imitate the real life without the help of the powerful GPU. We test the built NTCs for 100 round. In each round, 2000 packets are randomly chosen from the testing dataset of ISCXVPNnonVPN and mobile application dataset, respectively. The performance of the classifiers are shown in Table 4.3 and Table 4.4.

The evaluation shows the performance of NTCs by using the two dataset. Using the ISCX-VPNnonVPN, it demonstrates that the both the CNN-NTC, MLP-NTC provide high performance on all average of Recall, Precision, and classification accuracy and F1 score, which is above 96%. The DP-NTC shows lower performance on these evaluation metric which is above 95%. Using the mobile application dataset, the DP-NTC, MLP-NTC show

high Recall, Precision, accuracy and F1 score which is above 98%. The CNN-NTC shows the slightly lower performance which is above 97%. Meanwhile, the computational efficiency of the NTCs are also evaluated, which includes the processing speed and bandwidth. By evaluating the proposed and built traffic classifiers with CPU in both two dataset, the MLP-NTC shows the highest processing speed which is around 22 packets per millisecond and the highest bandwidth which is around 0.24 Gbps. The DP-NTC shows lower processing speed which is around 0.6 packets per millisecond and bandwidth which is 6.7 Mbps average value. And the CNN-NTC shows lower processing speed with 0.63 packets per millisecond and the lowest bandwidth which is around 1.2 Mbps.

Subsequently, we remove the nodes from the MLP-NTC based on the Algorithm 5. Firstly, the nodes in each layer are stopped at 242,242,242 due to the decreasing of the $\mathcal{A}$ by continuing removing. Then the node importance I is calculated by Eq. (4.4), which consider both mean and variance value of the weights. We also evaluate the node importance by using other ways for comparison.

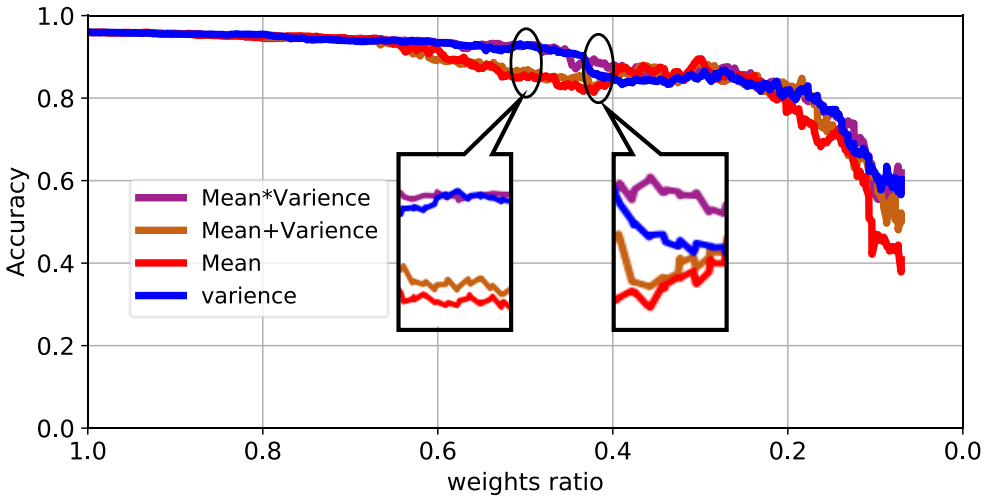

Figure 4.7: The evaluation of the node importance calculation

The Fig. 4.7 shows the change of accuracy with different weights ratio by using the different way to calculate the I, which includes mean value of the corresponding weights, variance value of the weights, summation and product of mean and variance of the corresponding weights. As we can see, in the first frame, the variance, and product of mean and variance shows the high accuracy during the decreasing, while the mean and summation of mean and variance shows the low accuracy. And in the second frame, the variance start decreasing and the product of mean and variance still shows the sustained high accuracy. That means the product of mean and variance of the corresponding weights can offer a more stable accuracy during the weights ratio. After the node importance is evaluated, the node in the target layer is deleted in every round based on the distribution of the weights in each layer.

The nodes in the MLP-NTC is stopped at 212, 186, 188 using ISCX-VPNnonVPN, and 210, 180, 190 using mobile application dataset, as a new structure of the MLP-NTC, namely optimized MLP-NTC. Then, we evaluate the proposed jump-type adaptive MLP-NTC and precise-type adaptive MLP-NTC based on Alg.6 and Alg.7 in two scenarios, receptively. Total of 4500 and 4000 testing data are used in this part. Here we set the ϵ_u to 20%. The nodes in the model is kept removing until the accuracy reach the 1 - ϵ_u. In the simulation, we simulate a network situation in 60 second, and each second has a λ_t which is generated based on the uniform distribution from 0 to 437 which is the largest number of packets the adaptive neural network can classify in 100 ms. In every second, we generate 10 numbers based on the Poisson distribution which is the number of the coming packets in every 100ms. Thus, a total of 600 numbers of the coming packet are generated in this simulation. For comparison, the MLP-NTC and optimized MLP-NTC is also evaluated, whose processing speed are a fixed number.

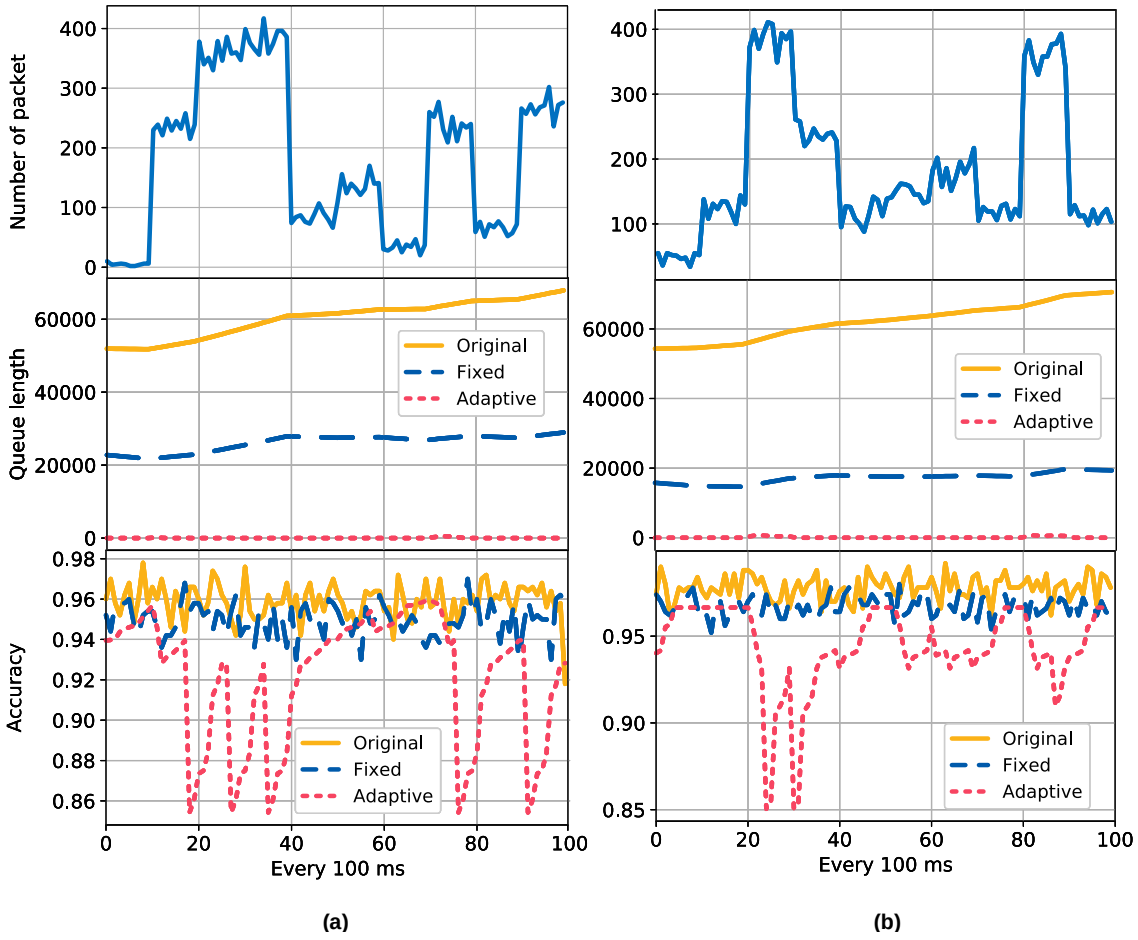

Figure 4.8: The number of packets, changes of queuing length and classification accuracy with time by using (a) ISCX-VPNnonVPN; (b) our new dataset in scenario I.

The Fig. 4.8 shows the curves of the number of the coming packets, the changes of the queuing length and accuracy of the baseline MLP-NTC, the optimized MLP-NTC and the jump-type adaptive MLP-NTC by using two different dataset in scenario I. Here we randomly intercept 10 seconds in the whole 60s to plot this figure. As we can see, the baseline MLP-NTC, optimized MLP-NTC shows a high queuing length, while the jump-type adaptive MLP-NTC shows much lower queuing length. Form 2nd to 4 th second, the coming packets are increasing, which lead to the increasing of the queuing length in baseline MLP-NTC and optimized MLP-NTC. Meanwhile, the adaptive MLP-NTC still keep low

queuing length by changing the structure of the MLP-NTC. Changes of the structure in adaptive MLP-NTC leads to the change of the accuracy. As we can see, the accuracy of the adaptive MLP-NTC is up and down to adapt to the different amounts of packets. Accuracy is from 82% to 95%, which is still acceptable.

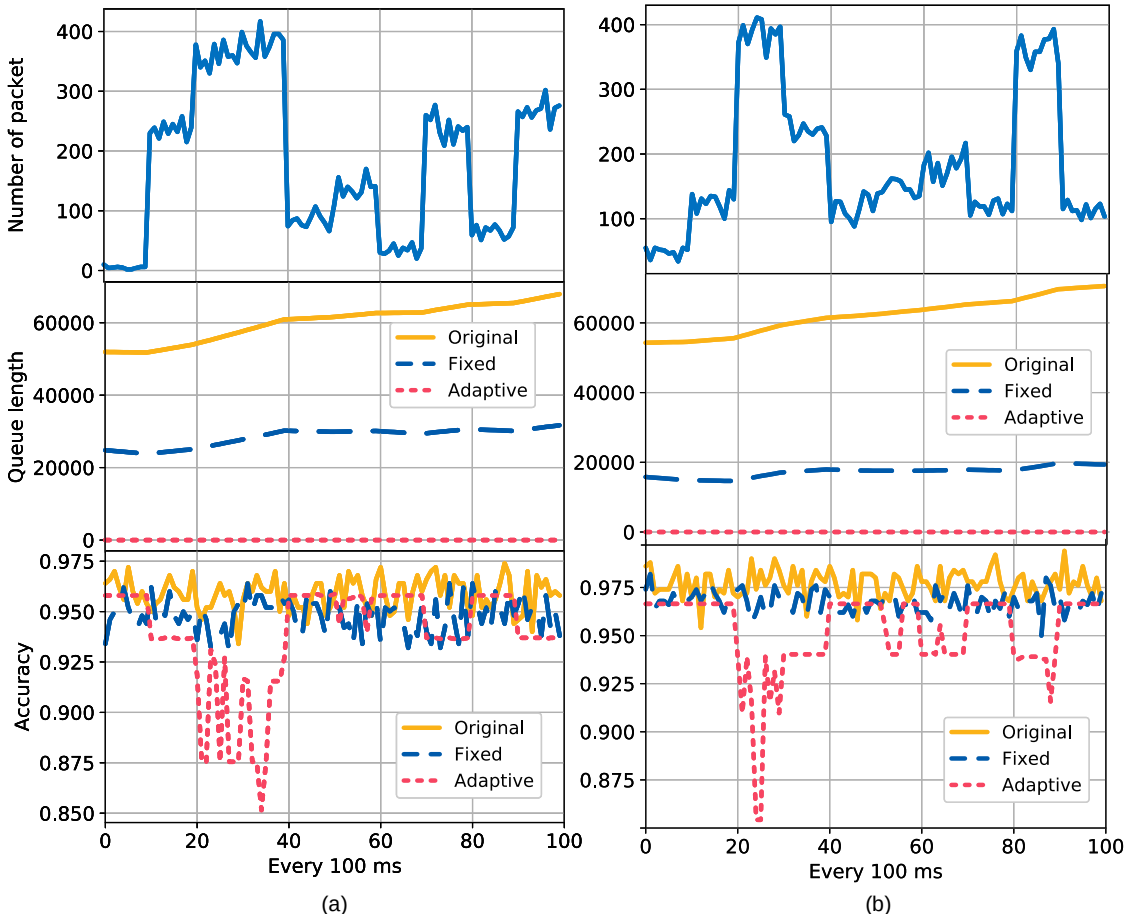

Figure 4.9: Values of λ, number of packets, changes of queuing length and accuracy with time by using (a) ISCX-VPNnonVPN; (b) our new dataset

The Fig. 4.9 shows number of packets, changes of queuing length and accuracy of baseline MLP-NTC, the optimized MLP-NTC and the precise-type adaptive MLP-NTC in scenario II. As we can see, the precise-type adaptive MLP-NTC keeps low queuing length in the

whole 10s, while the baseline MLP-NTC and optimized MLP-NTC show much high queuing length. The classification accuracy of precise-type adaptive MLP-NTC is from 85% to 94% which are also acceptable.

Table 4.5 and Table 4.6 show the accuracy, queue length, processing speed and the bandwidth of built MLP-NTCs in scenario I. Using the open dataset, the baseline MLP-NTC shows the highest queue length. It has 50441 packets as the average queue length and 103321 as the maximum queue length. And the queuing length of the optimized MLP-NTC is much lower than the baseline MLP-NTC, with an average of 21330 and maximum queue length is 44654. And jump-type adaptive MLP-NTC shows the lowest minimum, average, and maximum queue length, which are 0, 46, and 1012 packets. Meanwhile, the baseline shows the slowest processing speed and smallest bandwidth, which are 21 packets per millisecond and 220 Mbps. The optimized MLP-NTC shows much higher speed and larger bandwidth with 113 packets per millisecond 1.22Gbps, which increases 4.38 and 4.54 times from the baseline MLP-NTC. The jump-type adaptive MLP-NTC shows the fastest processing speed and the largest bandwidth. The average and the largest processing speed and bandwidth are 294/400 packets per millisecond, 3.16/4.34 Gbps which increase 13/18 times from the baseline MLP-NTC. For accuracy, the baseline MLP-NTC shows the highest classification accuracy above 96%. The optimized MLP-NTC shows a slightly lower accuracy around 95%. And the jump-type adaptive shows the average accuracy around 91%. Using the mobile application dataset, baseline MLP-NTC shows the highest queue length around 53545. The optimized MLP-NTC shows much lower queue length around 15081. The jump-type adaptive MLP-NTC shows the lowest queue length around 55. Meanwhile, the baseline MLP-NTC shows the lowest processing speed and bandwidth with 21 packet/ms and 230 Mbps. The optimized MLP-NTC shows a higher processing

speed and bandwidth round 120 packets/ms and 1.3 Gbps, which increase round 6 times than baseline MLP-NTC. The jump-type adaptive MLP-NTC shows the highest processing speed and bandwidth with 317 packets/ms and 3.44 Gbps, which increase round 12 times than baseline. Note that the largest bandwidth the jump-type adaptive MLP-NTC can support is 5.9 Gbps, which increase round 24 times than baseline.

The lower parts in Table 4.5 and Table 4.6 show the performance of the built MLP-NTCs in scenario II. Using the open dataset, the performance of the baseline MLP-NTC and the optimized MLP-NTC are almost the same with the performance of baseline MLP-NTC and the optimized MLP-NTC in the upper table. Meanwhile, the precise-type adaptive MLP-NTC has the lowest queue length with minimum, average and maximum queue length are all 0. And it shows high processing speed and large bandwidth. The average and maximum processing speed and bandwidth are 255 and 544 packets/per millisecond/2.77 and 5.90 Gbps, which increase 11/24 times than the baseline. Using the mobile application dataset, The precise-type adaptive MLP-NTC shows the lowest queue length with 0. Meanwhile, it shows the highest processing speed and bandwidth with 267 packets/ms and 2.9 Gbps, which increase round 12 times than baseline. The largest bandwidth the precise-type adaptive MLP-NTC can support is 5.31 Gbps, which increase round 21 times than baseline. Note that the average bandwidth of precise-type MLP-NTC is lower than the jump-type MLP-NTC. It is because the precise-type MLP-NTC choose the most suitable structure to process the packets in every period. It always choose the structure which has the enough processing speed and the high accuracy. As we can see, the average accuracy of the MLP-NTC only decreases around 2% than the baseline using two dataset.

4.5.3 Case Study

In this subsection, we simulate the situation that our proposed lossless and adaptive optimization for MLP-based NTC to perform packet classification on a real router. According to the low computation capability of routers, we choose the the Raspberry Pi 3B+ as our hardware platform. Raspberry Pi 3B+ is a microcomputer which includes 64-bit quad-core ARM Cortex-A53 with 1.2GHz, 1G memory, 802.11n Wi-Fi, Bluetooth 4.2 (BLE). The models are implement and evaluated on the Raspberry Pi 3B+, with Python and Pytorch 1.1.3 running in Ubuntu. We test 100 round, and in every round we randomly choose 100 packets in the testing dataset to evaluate the models.And the details are shown in the Table. 4.7.

The results show the processing speed and support bandwidth of the baseline MLP-NTC, optimized MLP-NTC and jump-type adaptive MLP-NTC by using two dataset. As we can see, it demonstrates that the baseline MLP-NTC shows the lowest processing speed and bandwidth which are 0.058 packets/ms, 0.64 Mbps by using the open dataset and 0.055 packets/ms, 0.61 Mbps by using the mobile application dataset. And the optimized MLP-NTC shows the higher speed and bandwidth, the average value is around 1.00 packets/ms, 11.06 Mbps, 1.12packets/ms and 12.44 Mbps which increase around 7 times than the baseline MLP-NTC. The jump-type adaptive MLP-NTC shows the highest processing speed and support bandwidth. The processing speed and support bandwidth are 2.26 packets/ms and 25.16 Mbps, 2.44 pakcets/ms and 27.10 Mbps, which increase 33/36 times than the baseline.

4.5.4 Conclusion

Network traffic classification (NTC), especially AI-based, plays an important role in IoT applications. To speed up the AI-based NTCs for those IoT devices with low operating capability, we proposed a lossless optimization scheme for MLP based NTC, which can speed up the MLP based NTC without sacrificing the classification accuracy. Moreover, to deal with the different situation of real-time network, we further proposed two types of adaptive optimization for MLP based NTC (i.e., jump-type adaptive MLP-NTC and precise-type adaptive MLP-NTC). The results demonstrated that the MLP based NTC can increase 5 times with a slightly loss of accuracy ϵ_0 using lossless optimization. Meanwhile, the two type of adaptive MLP-NTC provided round 3/5.5 Gbps bandwidth, which are 14/22 times than the baseline.

Table 4.3: Classification performance of the deep learning based classifiers.

		Recall (%)			Precision (%)			F1 Score (%)		
		Min.	Avg.	Max.	Min.	Avg.	Max.	Min.	Avg.	Max.
ISCX VPNnonVPN 2016	DP-NTC	93.6	95.3	97.6	93.5	95.3	97.5	93.5	95.3	97.5
	CNN-NTC	**94.2**	96.1	**98.3**	**94.2**	96.1	**98.3**	**94.2**	96.2	**98.3**
	MLP-NTC	93.8	**96.3**	98.1	93.8	**96.3**	98.2	93.8	**96.3**	98.2
Our new dataset	DP-NTC	96.4	**98.0**	99.2	96.3	**98.0**	**99.2**	96.4	**98.0**	**99.2**
	CNN-NTC	95.8	97.4	99.0	95.8	97.5	99.0	95.8	97.5	99.0
	MLP-NTC	**96.6**	97.8	**99.3**	**96.6**	97.8	**99.2**	**96.6**	97.9	99.1

Table 4.4: Classification performance of the deep learning based classifiers. (Cont.)

		Accuracy (%)			Speed (CPU) (packets/ms)			Bandwidth (CPU) (Mbps)		
		Min.	Avg.	Max.	Min.	Avg.	Max.	Min.	Avg.	Max.
ISCX VPNnonVPN 2016	DP-NTC	93.4	95.4	97.6	0.60	0.61	0.62	6.5	6.7	6.8
	CNN-NTC	**94.2**	96.2	**98.2**	0.61	0.63	0.65	1.1	1.2	1.3
	MLP-NTC	93.8	**96.4**	**98.2**	**21.4**	**22.1**	**22.6**	**0.2326**	**0.2397**	**0.2451**
Our new dataset	DP-NTC	96.2	**98.0**	**99.2g**	0.60	0.61	0.62	0.0065	0.0067	0.0068
	CNN-NTC	95.8	97.5	99.0	0.61	0.63	0.65	0.0011	0.0012	0.0012
	MLP-NTC	**96.6**	97.9	**99.2**	**20.7**	**21.5**	**22.5**	**0.2299**	**0.2389**	**0.25**

Table 4.5: The evaluation result of MLP-based network traffic classifier in scenario 1.

		Accuracy (%)			Queue length (packets)			Speed (packets/ms)			Bandwidth (Gbps)		
		Min.	Avg.	Max.	Min.	Avg.	Max.	Min.	Avg.	Max.	Min.	Avg.	Max.
Open dataset	Baseline MLP-NTC	**92.6**	**95.9**	**98.2**	24	50441	103321	21	21	22	0.23	0.23	0.24
	Optimized MLP-NTC	92.2 (-0.4%)	94.9 (-1%)	97.3 (-0.9%)	0	21330	44564	120 (+471%)	120 (+471%)	121 (+476%)	1.30 (+465%)	1.30 (+465%)	1.31 (+446%)
	Adaptive MLP-NTC	84.1 (-8.5%)	91.4 (-4.5%)	96.0 (-2.2%)	**0**	**46**	**1012**	**121 (+476%)**	**317 (+1410%)**	**510 (+2218%)**	**1.31 (+470%)**	**3.44 (+1396%)**	**5.53 (+2204%)**
Mobile application dataset	Baseline MLP-NTC	**95.8**	**97.8**	**99.6**	171	53545	106095	21	21	22	0.23	0.23	0.24
	Optimized MLP-NTC	95.0 (-0.8%)	96.7 (-1.1%)	98.7 (-0.9%)	43	15081	29295	**149 (+610%)**	149 (+610%)	150 (582%)	**1.62 (+546%)**	1.62 (+546%)	1.63 (+546%)
	Adaptive MLP-NTC	84.4 (-11.3)	94.5 (-3.3%)	96.7 (-2.9%)	**0**	**55**	**695**	**149 (+610%)**	**277 (+1219%)**	**544 (+2373%)**	**1.62 (+546%)**	**3.01 (+1209%)**	**5.90 (+2358)%**

Table 4.6: The evaluation result of MLP-based network traffic classifier in scenario 2.

		Accuray (%)			Queue length (packets)			Speed (packets/ms)			Bandwidth (Gbps)		
		Min.	Avg.	Max.	Min.	Avg.	Max.	Min.	Avg.	Max.	Min.	Avg.	Max.
Open dataset	Baseline MLP-NTC	**92.8**	**95.8**	**98.2**	24	50441	103321	21	21	22	0.23	0.23	0.24
	Optimized MLP-NTC	92.6 (-0.16%)	95.0 (-0.9%)	96.8 (-1.2%)	0	23364	48694	**120 (+471%)**	120 (+471%)	121 (+450%)	**1.30 (+465%)**	1.30 (+465%)	1.31 (+446%)
	Apaptive MLP-NTC	84.0 (-8.8%)	93.5 (-2.3%)	95.9 (-3.3%)	**0**	**0**	**0**	**120 (+471%)**	**255 (+1114%)**	**544 (+2373%)**	**1.30 (+465%)**	**2.77 (+1104%)**	**5.90 (+2358%)**
Mobile application dataset	Baseline MLP-NTC	**95.4**	**97.7**	**99.4**	160	53415	106010	21	21	22	0.23	0.23	0.24
	Optimized MLP-NTC	94.4 (-1%)	96.8 (-0.9%)	98.4 (-1%)	60	15030	29231	**149 (+524%)**	149 (+524%)	150 (+523.8%)	**1.62 (+604%)**	1.62 (+604%)	1.63 (+545.5%)
	Adaptive MLP-NTC	85.4 (-10.0%)	95.0 (-2.7%)	96.7 (-2.7%)	**0**	**0**	**0**	**149 (+524%)**	**267 (+1171%)**	**489 (+2123%)**	**1.62 (+604%)**	**2.90 (+1161%)**	**5.31 (+2123%)**

Table 4.7: The speed and support bandwidth of the MLP-NTC on Raspberry Pi 3B.

		Speed (packets/ms)			Bandwidth (Mbps)		
		Min.	Avg.	Max.	Min.	Avg.	Max.
ISCX VPNnonVPN 2016	Baseline MLP-NTC	0.053	0.058	0.066	0.59	0.64	0.73
	optimized MLP-NTC	0.49 (+825%)	0.51 (+779%)	0.53 (+703%)	5.44 (+822%)	5.66 (+784%)	5.89 (+707%)
	Adaptive MLP-NTC	**0.51 (+862%)**	**1.00 (+1624%)**	**2.26 (+3324%)**	**5.66 (+859%)**	**11.06 (+1628%)**	**25.16 (3347%)**
Our new dataset	Baseline MLP-NTC	0.052	0.055	0.066	0.58	0.61	0.73
	optimized MLP-NTC	0.63 (+1112%)	0.70 (+1173%)	0.78 (+1082%)	7.00 (+1106%)	7.78 (+1175%)	8.66 (+1086%)
	Adaptive MLP-NTC	**0.65 (+1150%)**	**1.12 (+1936%)**	**2.44 (3597%)**	**7.22 (+1145%)**	**12.44 (+1939%)**	**27.10 (+3612%)**

CHAPTER V

SIMPLIFYING DEEP LEARNING BASED MASSIVE MIMO CSI FEEDBACK PROCESS

5.1 Introduction

Massive multiple-input multiple-output (MIMO) systems have shown great promise in delivering high spectrum and energy efficiency for 5G and future wireless communication systems [120]. The downlink channel state information (CSI) needs to be obtained at the base station (BS) so that the MIMO system can acquire the performance gain with beamforming. In frequency division duplexing (FDD) mode, downlink CSI is usually estimated at the user equipment (UE) and fed back to the BS. However, the dimension of CSI matrix is sharply increased in massive MIMO system. The bandwidth consumed by CSI feedback is therefore unacceptable. Thus, the CSI matrix should be compressed before sending back to reduce the overhead. The traditional compressed sensing (CS) method cannot work well because the measurement matrix is usually non-optimal and the recovery is time consuming.

Recently, Deep learning (DL) based algorithms such as convolutional neural network (CNN) are considered as a good candidates for CSI feedback reconstruction due to its high performance. For instance, The authors in [121] used CNN to build CSI compression and recovery neural network called CsiNet. This network can learn how to use the channel architecture effectively to convert from CSI to the codebook. The reconstruction performance of CsiNet can highly increase the reconstruction performance more than the traditional compressed sensing methods. Subsequent related studies expanded the original scope of the network. The authors in [122] proposed a novel neural network based on multiresolution architecture. The authors in [123] proposed a DL-based novel CSI feedback scheme by

utilizing non-local block and dense connectivity. The authors in [124] proposed a multiple-rate DL-based CSI feedback framework that can switch under different compression ratios. However, the existing deep learning based CSI feedback models are usually based on the CNN architecture which has high computational complexity and huge amount of parameters, which is challenging to be implemented on a mobile terminal. To tackle this issue, we propose a model simplification scheme for deep learning based massive MIMO CSI feedback model. In specific, an dynamic channel sparsity scheme is proposed for automatically deciding the optimized structure of the compact network. We define the channel importance as the L1-norm of each channel in layer. A elimination threshold θ is defined to divide the channels in each layer as two parts, e.g., cultivable channels and marginalized channels. The cultivable channels the most important channels in their layer, and the marginalized channels are limited by L1-norm regularization. Two factors are designed to calculate θ for each layer which are channel baseline and layer complexity. The channel baseline is computed as the average value of the L1-norm of each channels or nodes in the particular layer. The layer complexity is calculated based on number of channels or nodes in the target and adjacent layer. The loss function is modified for different group of channels (e.g., cultivable channels and marginalized channels) in each layer based on the elimination threshold. A gap threshold is defined to quickly decide the structure of the optimized network, and a fine tuning may be applied to sustain the performance. Two popular deep learning based CSI feedback models, e.g., CsiNet, DualNet, are developed as the baseline for evaluation result. WINNER II is used to generate the channel state information. The results demonstrate that the proposed method can speed up 3 times as average value comparing with the baseline models without sacrificing the construction performance. To summarize, the major contributions of this chapter are:

- A model simplification scheme for deep learning based massive MIMO CSI feedback model is proposed.
- An dynamic targeted sparsity scheme is proposed for automatically deciding the optimized structure of the compact network.
- A autonomous stopping criteria is proposed to decide the timing to stop the dynamic targeted sparsity.
- Two deep learning based massive MIMO CSI feedback model are implemented as the baseline models for evaluation. Evaluation results demonstrated that the propose scheme reduce over 50% of parameters and Flops from baseline model without loss of reconstruction performance.

The rest of the chapter is organized as follows. Section 5.2 introduces the studied CSI feedback model. Section 5.3 describes The proposed model simplification for DL-based CSI networks. Section 5.4 shows the evaluation results. Section 5.5 concludes this chapter and describes the future work.

5.2 Studied CSI Feedback Model

We consider an FDD mmWave massive MIMO communication system. The downlink channel is divided into N orthogonal sub-carriers. Assume that the base station equips M antennas and all the users are single-antenna. The channel is denoted as:

$$\mathbf{h}_n = \sum_{p=1}^{P_n} \mathbf{h}_{n,p}, \tag{5.1}$$

where P_n is the number of total paths of n-th sub-carrier. In specific, the element of $\mathbf{h}_{n,p} \in \mathbf{C}^{M\times 1}$ is denoted as:

$$h_{n,p,m} = \rho_{n,p}\frac{1}{\sqrt{M}}\exp[-j2\pi\frac{d}{\lambda_n}(m-1)\cos\theta], \tag{5.2}$$

where m is the indexes in the channel vector $m = 1, ..., M$. $\rho_{n,p}$ is the path gain of pth path. θ is the Angle of Departure (AoD). The AoDs of all sub-carriers are assumed to be the same. λ_n is the wave length of the carrier. For mmWave, λ is between 1-10 millimeters. d is the distance of two antennas in the antenna array. The received signal at the nth carrier is denoted as:

$$y_n = \mathbf{h}_n^H \mathbf{A}_n \mathbf{D}_n \mathbf{s}_n + z_n, \tag{5.3}$$

where $(\cdot)^H$ is the Hermitian transpose. $\mathbf{s}_n \in$ is the transmitted data stream of nth sub-carrier. $\mathbf{D}_n$ is the digital precoder and $\mathbf{A}_n$ is the analog precoder. z_n is the noise in nth sub-carrier and $z_n \sim \mathcal{CN}(0, \sigma^2)$. The channel matrix is denoted as: $\mathbf{H} = [\mathbf{h}_1, ..., \mathbf{h}_N]$. And the channel matrix in the delay-angular domain is denoted as: $\tilde{\mathbf{H}} = \mathbf{F}^H\mathbf{H}$, where $\mathbf{F}$ is the Discrete Fourier Transform (DFT) matrix. The data rate of the user is the sum data rate of all the sub-carriers.

In practice, the channel keeps changing due to the movement of the user. The base station will send a training sequence which is known by the both side at the beginning of the channel estimation stage. After receiving the training sequence, the user is able to calculate its channel information. Then the user feeds the channel information back to the base station. Conventionally, we assume that the channel keeps same in one coherent time slot. Because of the Doppler effect and multi-path phenomenon, the length of the coherent time depends on two aspects: (1) the speed of the movement and (2) the frequency of the carrier. Briefly, if the user moves faster or the frequency of the carrier is higher, the coherent time slot gets shorter. Then channel need to be more frequently estimated then fed back

to the base station. The frequently channel feed back procedure consumes more resources if all the channel are fed instantaneously. The maximum length of the coherence time slot is denoted as [125]:

$$T_s = \frac{1}{4D_s}, \tag{5.4}$$

where $D_s = \frac{2f_c v}{c}$ is the Doppler spread, f_c is the frequency of the carrier, v is the movement speed of the user and c is the speed of light. Due to the movement of the scatters, even if the user is static, the channel also might change. We give few examples of the length of coherence time slot here. Assume that the user is sitting in a car driven under 30 meter per second, and the base station is transmitting data by using mm-wave carrier ($30GHz - 300GHz$). Then the coherence time is between $0.04ms$ and $0.004ms$. If the user is supported by the current 5G system which is working under sub-6G frequency, the coherence time is $0.2ms$. And we also discuss a example that the user is under a slow movement e.g., walking or jogging with 1 meter per second. Then the coherence time are $0.12ms$ to $1.2ms$ and $6ms$ respectively.

To reduce the channel feedback overhead, there are a encode procedure before feedback: (1) channel information compression and (2) quantization. We denote the estimated channel in delay-angular domain at the user side as $\hat{\mathbf{H}}$. Then the sent channel information is denoted as:

$$\hat{\mathbf{H}}_f = \mathcal{Q}\{\mathcal{C}\{\hat{\mathbf{H}}, C_e\}\}, \tag{5.5}$$

where $\hat{\mathbf{H}}_f$ is the sent channel information after compression and quantization. $\mathcal{Q}\{\cdot\}$ is the quantization function and $\mathcal{C}\{\cdot\}$ is the compression function. And C_e is the compression parameter. The decoded channel at the base station side is denoted as:

$$\bar{\mathbf{H}} = \bar{\mathcal{C}}\{\bar{\mathcal{Q}}\{\hat{\mathbf{H}}_f\}, C_d\}, \tag{5.6}$$

where $\bar{\mathcal{Q}}\{\cdot\}$ and $\bar{\mathcal{C}}\{\cdot\}$ are the decompression and dequantization functions. And C_d is the decompression parameter. Assume that the channel is perfectly estimated at the user side, then the goal of the channel information feedback stage is to minimize the error between the decoded channel and the estimated channel. The problem is denoted as:

$$\begin{aligned} &\min_{\mathcal{C},\mathcal{Q},\mathcal{D}} ||\hat{\mathbf{H}} - \bar{\mathbf{H}}||_2^2 \\ &s.t. \quad \tau \leq T_s, \end{aligned} \tag{5.7}$$

where τ is the time duration between user generate $\hat{\mathbf{H}}$ and the base station decode $\bar{\mathbf{H}}$.

5.3 The Proposed Model Simplification for DL-based CSI Networks

5.3.1 The whole framework

In this section, we introduce the proposed structure model simplification for lightweight DL-based CSI networks. The proposed scheme simplify the kernels and nodes in convolutional layer and dense layer, respectively. First, an initial network is built, it can be designed based on some public CSI feedback model, e.g., CsiNet, or designed by users for personal requirements. After that, an dynamic targeted sparsity scheme is proposed for automatically deciding the optimized structure of the compact network. To be specific, θ is defined as the elimination threshold that determines the kernels which are marginalized in each iteration. Two factors which are kernel baseline and layer complexity are designed to calculate the elimination threshold for each layer. The kernel baseline is computed as the average value of the L1-norm of each kernel in the particular layer, which is the summation of the weights and bias in this kernel. The layer complexity is calculated based on number of channels or nodes in the target and adjacent layer. The loss function is modified for different group of channels or nodes in each layer based on the elimination threshold. In training process, the channels or nodes whose L1-norm is bigger than the elimination threshold are trained to be close with the kernel with the largest L1-norm, while those channels or nodes whose

L1-norm is smaller than the elimination threshold are limited by L1-norm regularization. An autonomous stopping criteria is proposed for stopping the dynamic targeted sparsity and removed the nodes in the marginalized group, and a fine tuning is applied to sustain the performance.

5.3.2 Dynamic Targeted Sparsity for Convolutional Layer

After the structure of initial network is determined, the proposed dynamic targeted sparsity automatically group the kernels in convolutional layers, includes computing the kernel importance, divide kernel in two groups which are cultivable kernels and marginalized kernels based on elimination threshold, and make different actions. In specific, for a data sample of channel state information $\hat{\mathbf{H}}$ in the format of 2-D matrix, it is processed by convolutional kernels in convolutional layer as follows:

$$z_{k,i,j} = b_k + \sum_{l} \sum_{m=1}^{W} \sum_{n=1}^{H} w_{k,l,m,n} * \hat{\mathbf{H}}_{l,i+m-1,j+n-1}, \tag{5.8}$$

where '$*$' is the convolution operator, k is the order of convolution kennels; l is the depth order of the channel state information; W and H are the width and length of the input data samples; w and q are the weights and bias, respectively.

During the training process, the weights and bias of kernels in each convolutional layer are optimized based on each batch of input data samples. The value of weights and bias in convolutional layer can roughly represent the relevance of the input packets to the classification result. In other words, a larger weight indicates higher contributions from the input [66]. Thus, weights and bias are extracted for calculating the importance of each kernel. Here We apply L1-norm to represent the importance of each kernels in convolutional

layer. For i-th kernel whose size is K by K in j-th layer, the kernel importance can be computed as:

$$u_{i,j}^{(c)} = |b_{i,j}| + \sum_{m=1}^{K} \sum_{n=1}^{K} |w_{m,n}|, \tag{5.9}$$

where u_j^i is the kernel importance; $w_{m,n}$ is the weights in the particular kernel. A larger u_j^i indicates that the particular kernel play more important role in this layer, while a lower u_j^i means that this kernel has less importance than others. After the kernel importance are calculated, an elimination threshold are designed to divide the kernels into two parts: cultivable kernels and marginalized kernels. The cultivable kernels are the kernels whose kernel importance are larger than the elimination threshold. They are trained without restrictions during the training process to achieve a better reconstruction performance. The marginalized kernels are the kernels whose kernel importance are lower than the elimination threshold. They are constrained by L1-norm regularization during training to make their kernel importance smaller and smaller to minimize the impact on the model when they are finally removed. The elimination threshold are calculated based on two factors, kernel baseline and layer complexity, kernel baseline are defined as the average kernel importance of each layer, computed as:

$$o_j^{(c)} = \frac{1}{L^{(c)}} \sum_{i=1}^{L^{(c)}} u_{i,j}^{(c)}, \tag{5.10}$$

where $o_j^{(c)}$ is the kernel baseline, $L^{(c)}$ is the number of kernels in the j-th layer. Channel baseline represents the bottom line of the importance of kernels. Kernel complexity is calculated as the ratio based on the number of kernels in particular and adjacent layers. Note that only continuous convolutional layers are considered in this case due to the different impact of number of kernels in convolutional layer and nodes in dense layer. For the j-th

convolutional layer, the kernel complexity is computed as:

$$\beta_j^{(c)} = \frac{\sum_{i=j-1}^{j+1} L_i^{(c)}}{\sum_{m=2}^{P-1} \sum_{n=m-1}^{m+1} L_n^{(c)}}, \tag{5.11}$$

where $\beta_j^{(c)}$ is the layer complexity. P is the number of layers. A larger β indicates that the particular layer has a larger amount of kernels which may be redundant, or has large number of kernels in adjacent layers which means reduce more complexity when kernels in this layer are removed. A lower β means that the particular layer has a few amount of kernels that can not be removed, or has less number of kernels in adjacent layers which means reduce just a few complexity when kernels in this layer are removed. The elimination threshold is the product of the kernel baseline and layer complexity, which can be computed as:

$$\theta_j^{(c)} = \beta_j^{(c)} \cdot o_j^{(c)}, \tag{5.12}$$

Based on the $\theta_j^{(c)}$, the kernels in j-th layer are divided into two groups. A larger $\theta_j^{(c)}$ leads that more kernels are added into the group of marginalized kernels. A smaller $\theta_j^{(c)}$ means that more kernels are classified as the cultivable kernels. The cultivable kernels whose channel importance is larger than elimination threshold have no restrictions during training process, the marginalized kernels whose kernel importance is less than elimination threshold are limited by the L1-norm regularization.

5.3.3 Dynamic Targeted Sparsity for Dense Layer

In this section, we introduce the proposed simplification scheme for dense layer in DL-based CSI feedback process. Typically, reshaping process and dense layer are applied after the last convolutional layer to compress the feature map to the code word. In addition, some researchers proposed to directly use dense layer to build the deep learning based CSI feedback process. The dense layer receives input feature from all the neurons of previous layer, and output the result based on corresponding weights and bias, computed as:

$$z_{i,j} = \sigma(b_{i,j} + \sum_{m=1}^{l} w_{m,i,j} \cdot z_{m,j-1}), \tag{5.13}$$

where l is the number of nodes in j-th dense layer. Similar with the process in convolutional layer, the weights and bias in each dense layer are used to calculate the node importance during the training process. L1-norm is applied to calculate the importance of each nodes in dense layer. For i-th node in j-th dense layer, we calculate the node importance as:

$$u_{i,j}^{(d)} = |b_{i,j}| + \sum_{n=1}^{N_j} |w_{m,i,j}|, \tag{5.14}$$

where N_j is the number of corresponding weights for calculating the i-th nodes in j-th layer. A larger $u_{i,j}^{(d)}$ indicates that the particular node is more important than others, while a smaller $u_{i,j}^{(d)}$ means that this node has less importance among all nodes in this layer. After determining all node importance, we calculate the elimination threshold for dense layer to divide the nodes into two parts: cultivable nodes and marginalized nodes. The cultivable nodes are the nodes whose node importance are larger than the elimination threshold. They are trained without restrictions during the training process to achieve a better reconstruction performance. The marginalized nodes are the nodes whose node importance are lower than the elimination threshold. They are constrained by L1-norm regularization during training to minimize the impact on the model when they are finally removed. In this case, we also define the node baseline which is the average node importance of each layer, computed as:

$$o_j^{(d)} = \frac{1}{L^{(d)}} \sum_{i=1}^{L^{(d)}} u_{i,j}^{(d)}, \tag{5.15}$$

where $o_j^{(d)}$ is the node baseline, $L^{(d)}$ is the number of nodes in the j-th layer. Node baseline represents the bottom line of the importance of nodes. Node complexity is defined as the ratio based on the number of nodes in particular and adjacent dense layers. Similar with

the process for convolutional layer, only continuous dense layers are considered in this case due to the different impact of number of kernels in convolutional layer and nodes in dense layer. For the j-th dense layer, the node complexity is computed as:

$$\beta_j^{(d)} = \frac{\sum_{i=j-1}^{j+1} L_i^{(d)}}{\sum_{m=2}^{P^{(d)}-1} \sum_{n=m-1}^{m+1} L_n^{(d)}}, \tag{5.16}$$

where $\beta_j^{(d)}$ is the layer complexity. $P^{(d)}$ is the number of layers. A larger $\beta^{(d)}$ indicates that the particular dense layer has a larger amount of nodes which may be redundant, or has large number of nodes in adjacent layers which means reduce more complexity when nodes in this layer are removed. A lower $\beta^{(d)}$ means that the particular dense layer has a few amount of nodes that can not be removed, or has less number of nodes in adjacent layers which means reduce just a few complexity when nodes in this layer are removed. The elimination threshold is the product of the node baseline and node complexity, which can be computed as:

$$\theta_j^{(d)} = \beta_j^{(d)} \cdot o_j^{(d)}, \tag{5.17}$$

Based on the $\theta_j^{(d)}$, the nodes in j-th layer are divided into two groups. A larger $\theta_j^{(d)}$ leads that more nodes are added into the group of marginalized nodes. A smaller $\theta_j^{(d)}$ means that more nodes are classified as the cultivable nodes. The cultivable nodes whose node importance is larger than elimination threshold have no restrictions during training process, the marginalized nodes whose node importance is less than elimination threshold are limited by the L1-norm regularization.

5.3.4 Autonomous Stopping Criteria and Fine-tuning

After training with the dynamic sparsity applied on convolutional layer or dense layer, the importance of kernels or nodes in marginalized group are smaller and smaller to reduce the impact when they are removed. In this case, we redesign the loss function as:

$$\mathcal{L} = \min_{\mathcal{C},\mathcal{Q},\mathcal{D}} ||\hat{\mathbf{H}} - \bar{\mathbf{H}}||_2^2 + \lambda_1 \sum_{j=1}^{P^{(c)}} \sum_{i=1}^{L_m^{(c)}} ||c_{i,j}||_1 + \lambda_2 \sum_{j=1}^{P^{(d)}} \sum_{i=1}^{L_m^{(d)}} ||d_{i,j}||_1, \tag{5.18}$$

where $\mathcal{L}$ is the whole loss for deep learning based CSI feedback model. $||\mathbf{H} - \bar{\mathbf{H}}||_2^2$ is the reconstruction loss between the CSI matrix and reconstructed CSI matrix by decoder. $c_{i,j}$ is the i-th kernels in j-th convolutional layer; $d_{i,j}$ is the i-th nodes in j-th dense layer. $L_m^{(c)}$ and $L_m^{(d)}$ are the number of the kernels and nodes in marginalized group, respectively. λ_1 and λ_2 are the coefficient to control the L1-norm regularization of convolutional layer and dense layer. Note that λ_1 or λ_2 is set to 0 if there are no convolutional layers or dense layers applying in the model.

By resetting the loss function, the difference between the importance of kernels or nodes in cultivable and marginalized group are larger and larger. For fast deciding the optimal structure of the deep learning based Massive MIMO feedback model by stopping the dynamic targeted sparsity, we propose a autonomous stopping criteria to stop the dynamic targeted sparsity for convolutional or dense layer, as illustrated in Fig. 5.1.

After θ_j is calculated, the kernels or nodes are automatically divided into two camps: the cultivable and marginalized groups. Thus, the o_j in cultivable and marginalized group in convolutional or dense layer can be computed based Eq. (5.10) and Eq. (5.15). We judge the tendency for kernels or nodes to change camps through the two kernels or nodes in the two camps that are closest to the θ_j, respectively. Here two tendency threshold, i.e., γ_j^c and γ_j^m, are defined for cultivable and marginalized group, respectively. For cultivable group, The tendency threshold γ_j^c is to estimate the tendency of the kernels or nodes which is closest to θ_j, which can be computed as:

$$\gamma_j^c = \tau_2^c - \tau_1^c, \tag{5.19}$$

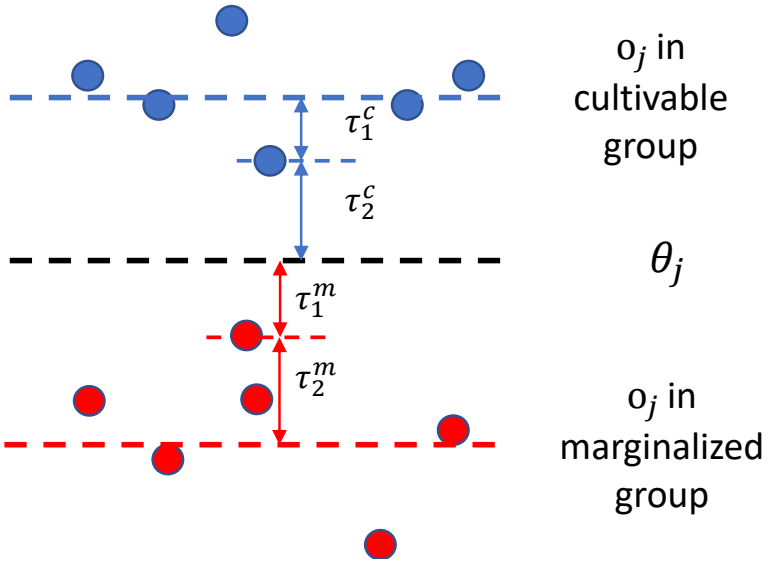

Figure 5.1: Autonomous stopping criteria.

where γ_j^c is the tendency threshold. τ_1^c is the distance between the importance of the particular kernel or node and o_j in cultivable group; τ_2^c is the distance between importance of the this kernel or node and θ_j; A positive γ_j^c indicates that the particular kernel or node are closer to the average importance in cultivable group than the θ_j, which means the kernels or nodes in cultivable group have a less possible to change the camps, the dynamic targeted sparsity can be stopped in this case, while a negative γ_j^c means that this kernel or node are closer to θ_j than the average importance in cultivable group. This means the kernels or nodes in cultivable group may still have a chance to drop to the marginalized group and thus the dynamic targeted sparsity need to be continued. For marginalized group, the tendency threshold γ_j^m is to estimate the tendency of the kernels or nodes which is closest to θ_j in the marginalized group, which can be computed as:

$$\gamma_j^m = \tau_2^m - \tau_1^m, \tag{5.20}$$

where γ_j^m is the tendency threshold for marginalized group. τ_1^m is the distance between importance of the this kernel or node and θ_j; τ_2^m is the distance between the importance of the particular kernel or node and o_j in marginalized group; A positive γ_j^m represents that this kernel or node are closer to θ_j than the average importance in marginalized group, which means the kernels or nodes in marginalized group still have a chance to jump to the cultivable group and thus the dynamic targeted sparsity need to be continued, while a negative γ_j^m indicates that the particular kernel or node are closer to the average importance in marginalized group than the θ_j, which means the kernels or nodes in marginalized group may not have a good chance to change the camps. In this case, the dynamic targeted sparsity can be stopped. γ_j^c and γ_j^m are calculated during the training process, the dynamic targeted sparsity is stopped only the γ of each convolutional or dense layer satisfies the stopping condition, i.e., $\gamma_1^c, \gamma_2^c, ..., \gamma_{L^{(c)}}^c > 0$ and $\gamma_1^m, \gamma_2^m, ..., \gamma_{L^{(c)}}^m < 0$ for convolutional layer, and $\gamma_1^c, \gamma_2^c, ..., \gamma_{L^{(d)}}^c > 0$ and $\gamma_1^m, \gamma_2^m, ..., \gamma_{L^{(d)}}^m < 0$ for dense layer.

Once the stopping criteria is met, the dynamic kernel sparsity is stopped because we argue that the kernels in cultivable and marginalized kernels are already stable and no kernels or nodes change the camps. After stopping the dynamic targeted sparsity scheme, the kernels in marginalized kernels in each layer are removed, a fine-tuning is applied .

5.4 Evaluation Results

5.4.1 Data Generation and Settings of Baseline model

To generate the channel state information, the WINNER II channel model is used. WINNER II channel stochastically determined the channel parameters based on statistical distributions extracted from channel measurement in practice. We assume that both the base station and users are in the urban area. The base station equips 32 antennas and the

users has one antenna each. The center frequency is set at 30 GHz. The total bandwidth is 3.2 GHz, and it is divided into 32 sub-carrier and each of them has 100MHz bandwidth. Since in this chapter, the users are considered to be static or move with very low velocity, the coherent bandwidth is larger than 100 MHz. Because of the high density of the buildings in the urban area, we assume that the line-of-sight (LoS) path does not exist, all the paths are none-line-of-sight (NLoS). The channel is sampled at the end of every time slot. Since the users are static, then the coherent time slot can be relatively long. However, the channel also changes with the change of the environment e.g., the movement of the scatters. Which means the channel can still change significantly between every two sample slots. At the end of every time slot, a 32×32 channel in space-frequency space are generated. Then the channel will be transformed to delay-angular domain by using the fore-mentioned DFT matrix. Due to the sparsity of the channel in the delay-angular domain, most of the data in the complex channel matrix are 0 or very close to 0. To evaluate the accuracy of the CSI recovery, we use normailized MSE, which can be denoted as:

$$NMSE = \frac{1}{n}\sum_{k=1}^{n} ||\mathbf{H}_d^k - \bar{\mathbf{H}}_d^k||^2 / ||\mathbf{H}_d^k||^2, \tag{5.21}$$

where k and n are the index and total number of samples in the testing set, respectively. The evaluations are conducted on a workstation that is equipped with an Intel®Core(TM) i7-6700 CPU@ 3.40 GHz, 32.0 GB RAM, and a Nvidia GeForce GTX 1080Ti. The AI-based NTC models are implemented on the PyTorch platform. We compare the CsiNet and FCFNN with and without the proposed simplification scheme. Details are shown in the following sections.

5.4.2 Evaluation Result of Dynamic Targeted Sparsity

In this section, we introduce the evaluation result of dynamic targeted sparsity. Fig. 5.2 shows the comparison of weights distribution in kernels trained with, without L-1 norm regularization and the dynamic targeted sparsity. Each row of the figure is the weights of a whole kernel. the top part in Fig. 5.2 is the weights distribution of the kernels trained without the L1-norm, the large weights which are the darker blocks are scattered. Each kernel has the large weights, which means the impact is large when the kernels are removed. The middle part in Fig. 5.2 is the weights distribution of the kernels trained with L1-norm, the large weights are also scattered. The L1-norm make the entire weights lower. However, large weights in this magnitude are still scattered, the impact of each kernel is still large when the kernels are removed. The bottom part in Fig. 5.2 is the weights distribution of the kernels trained with the proposed dynamic targeted sparsity. The proposed scheme are only limited the targeted kernels with the L1-norm regularization. We can see that only three kernels has the larger weights, and other kernels are all small weights. That means when these kernels are removed, the impact is least.

5.4.3 Evaluation Result of Autonomous Stopping Criteria

In this section, we introduce the proposed autonomous stopping criteria. The dynamic targeted sparsity is stopped when autonomous stopping criteria is satisfied.

The Fig. 5.3 shows the kernel baseline of each kernel, the corresponding cultivable group and marginalized group in convolutional layer in CsiNet. We apply the dynamic targeted sparsity in four layers. As we can see, the green, blue and orange line are the kernel baseline of cultivable group, kernel baseline in whole kernels, kernel baseline of marginalized group,

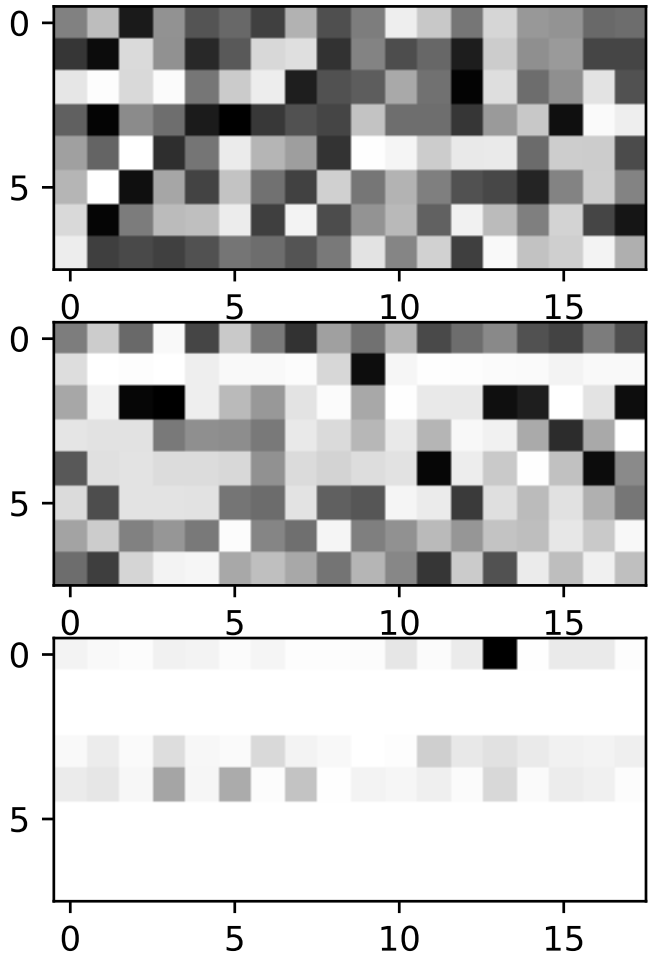

Figure 5.2: The Comparison of weights distribution in kernels trained with, without L-1 norm regularization and the dynamic targeted sparsity.

respectively. During the training process, the kernel baseline of cultivable group keep higher and higher because all the kernels are trained to have the same kernel importance with the kernel which has the highest kernel importance. The kernel baseline of marginalized group keep lower due to the limitation of L1-norm regularization.

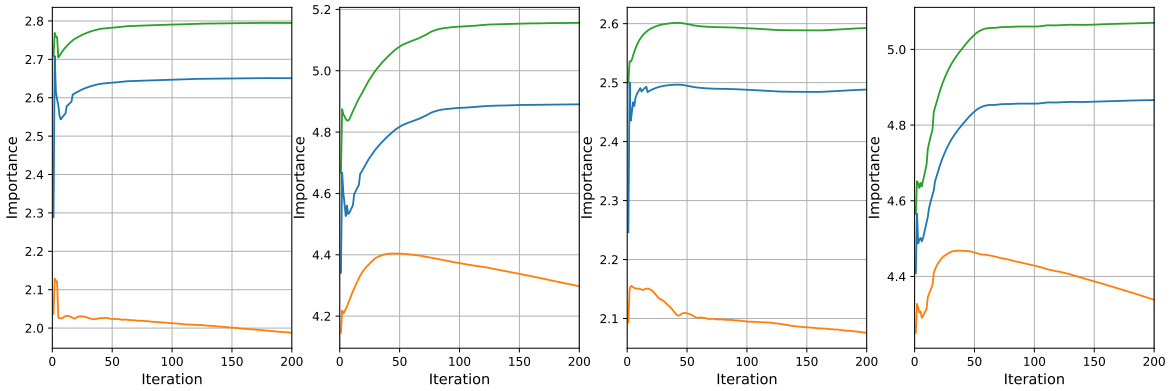

Figure 5.3: The kernel baseline of each kernel, the corresponding cultivable group and marginalized group in convolutional layer.

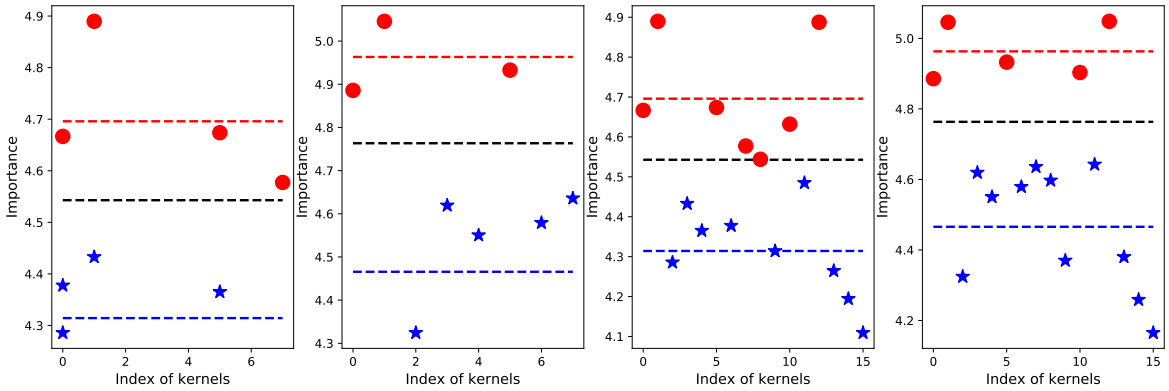

Figure 5.4: The distribution of kernel importance of each kernel in convolutional layer.

The Fig. 5.4 shows the distribution of kernel importance of each kernel in convolutional layer. The left two figures show two stages of the distribution of kernel importance in the first kernel (i.e., the left figure is before and the right figure is after). In previous stage, there are 4 kernels in cultivable group and 4 kernels in marginalized group. In cultivable group, there is a kernel around the kernel baseline. In the behind stage, the kernel close to the kernel baseline is moved to the marginalized group and limited by the L1-norm regularization. The kernels in cultivable group move higher because they are trained to

close to the top kernel. The right two figures show the two stages of the distribution of kernel importance in the fourth kernel (i.e., the left figure is before and the right figure is after). In the previous stage, there are 7 kernels in cultivable group and 9 kernels in marginalized group, the kernels close to the kernel baseline are moved to the marginalized group during the training process in the behind stage.

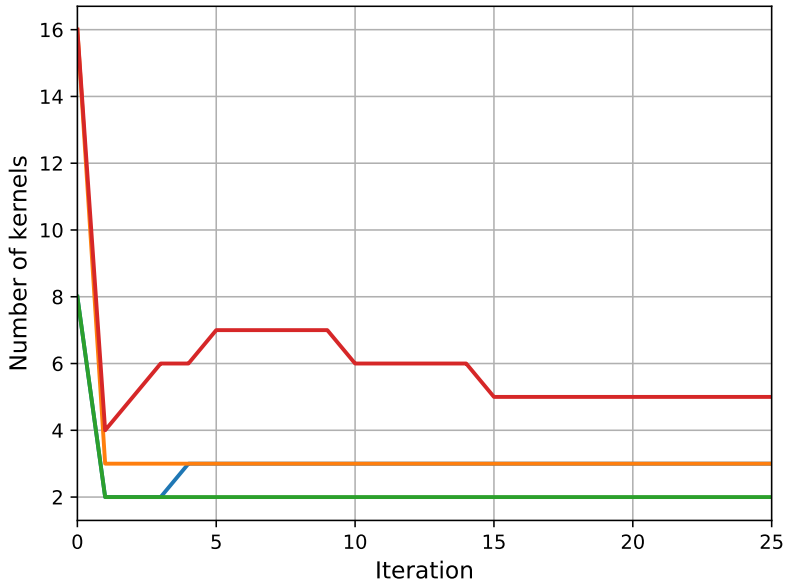

Figure 5.5: The Comparison of weights distribution in kernels trained with, without L-1 norm regularization and the dynamic targeted sparsity.

The Fig. 5.5 shows the change of the number of retained kernels in CsiNet during the training process. As we can see, in the beginning, the volatility of the number of kernels is large, and are more and more stable during the scheme. When the stopping criteria is satisfied, the scheme is stopped and the model will be trained based on the number of retained kernels.

5.4.4 Parameter Numbers and Flops of the Networks

In this section, we introduce the Parameter Numbers and Flops of the implemented DL-based CSI networks with and without the proposed simplification scheme, which is shown in Table 5.1. Note that the Flops is the number of multiplications (adds and nonlinear transformation are ignored). In Table 5.1, we evaluate the baseline model and update model with different compression ratio. When compression ratio is set to 1/16, the parameter number of CsiNet and FCFNN are 262308, 1697144, while the parameter number of simplified CsiNet and FCFNN are 87436 and 812322 which is 30% of the corresponding baseline model. The Flops of CsiNet, FCFNN are 561152, 1703234, while the parameter number of simplified CsiNet and FCFNN are 187050 and 567744 which is 30% of the corresponding baseline model. When compression ratio is set to 1/32, the parameter number of CsiNet and FCFNN are 131236 and 1297144, while the parameter number of simplified CsiNet and FCFNN are 82436 and 792313 which is 30% of the corresponding baseline model. The Flops of CsiNet and FCFNN are 299088 and 1403234, while the parameter number of CsiNet* and FCFNN* are 147050 and 264104 which is 30% of the corresponding baseline model.

5.4.5 CSI reconstruction performance of the networks

For comparison, we trained the deep learning based CSI feedback model with the original loss function, which only consider the reconstruction loss, and use the reconstruction loss and the L1-norm of all channels in all layer, respectively. Table 5.2 shows the reconstruction performance (i.e., NMSE) of the baseline and simplified models with different compression ratio. When the compression ratio is set to 1/4, the NMSE of CsiNet, FCFNN are -17.29 dB and -16.62 db, while the NMSE of simplified CsiNet, FCFNN achieve almost the same reconstruction performance with the baseline model which are -17.20 db and 16.89

Table 5.1: Overview of parameter numbers and Flops of the developed and simplified models.

CR	Method	Parameter number		Flops	
		Overall	From component	Overall	From component
1/4	CsiNet	2.1M	3186	5.41M	3.32M
	Simplified CsiNet	2.07M	918	2.95M	0.86M
	FCFNN	27.28M	26.24M	27.26M	25.2M
	Simpified FCFNN	8.23M	6.13M	8.22M	6.12M
1/8	CsiNet	1.04M	3186	4.37M	3.32M
	Simpified CsiNet	1.02M	810	1.75M	0.74M
	FCFNN	26.23M	25.3M	26.21M	25.2M
	Simpified CsiNet	5.66M	4.61M	5.66M	4.61M

db,respectively. When compression ratio is set to 1/8, the NMSE of CsiNet, FCFNN are -9.41 dB and -9.37 dB, while the NMSE of simplified CsiNet, FCFNN achieve almost the same reconstruction performance with the baseline model which are -9.45 db and 9.39 db,respectively.

Table 5.2: Overview of reconstruction performance of the developed and simplified models.

CR	**Method**	**NMSE**
1/4	CsiNet	-17.29
	Simplified CsiNet	-17.20
	FCFNN	16.62
	Simplified FCFNN	16.89
1/8	CsiNet	-9.41
	Simplified CsiNet	-9.45
	FCFNN	-9.37
	Simplified FCFNN	-9.39

5.5 Conclusion

Massive multiple-input multiple-output (MIMO) systems have shown great promise in delivering high spectrum and energy efficiency for 5G and future wireless communication systems. The downlink CSI needs to be obtained at the base station so that the MIMO system can acquire the performance gain with beamforming. In this chapter, a model simplification scheme for deep learning based massive MIMO CSI feedback model is proposed. To be specific, a dynamic channel sparsity scheme is proposed for automatically deciding the optimized structure of the compact network. After training, some unimportant channels and nodes are removed and a fine tuning is applied for sustaining the reconstruction performance. Two popular deep learning based CSI feedback models are implemented for evaluations. The results demonstrated that the proposed method can speed up 3 times as an average comparing with the baseline models.

CHAPTER VI

CONCLUSION AND FUTURE WORK

The development of the next-generation networking and communication systems is critical. Recently, artificial intelligence based algorithms are used in many technologies in next-generation networking and communication systems. In this dissertation, we explores a systematic approach that simplifies the AI-supported implementation for multiple networking and communication systems in this dissertation. To be specific, In chapter III, we proposed an adaptive and lightweight NTC framework, which can significantly reduce the input features and accelerate NTC models by one to two magnitudes while maintaining high accuracy. The proposed autonomous update scheme can accurately detect a change in feature contributions and update the NTC models to sustain the high accuracy. In chapter IV, we propose a lossless optimization for a popular AI-based NTC, i.e., Multilayer Perceptron (MLP) based NTC to remove the nodes that contribute the least to the classification result in the hidden layer with pruning method, thus speed up the NTC. Moreover, the propose adaptive pruning for MLP-based NTC to fit the different requirements of NTC on operating speed to solve the network problems, e.g, network congestion. The result demonstrate that the lossless optimization and adaptive pruned network traffic classifier can speed up an average of 5/10 times than the baseline MLP based NTC models. In chapter V, we propose a model simplification scheme for deep learning based massive MIMO CSI feedback model. To be specific, an dynamic channel sparsity scheme is proposed for automatically deciding the optimized structure of the compact network. The results show that the proposed method can reduce over 50% of parameters and Flops comparing with the baseline models. Our future work will focus on developing more simplified methods for more technologies in next-generation network and communication systems.

BIBLIOGRAPHY

[1] "The global state of digital 2022," https://www.hootsuite.com/newsroom/press-releases/digital-2022-report, accessed: 01-26-2022.

[2] A. Gupta and R. K. Jha, "A survey of 5g network: Architecture and emerging technologies," *IEEE Access*, vol. 3, pp. 1206–1232, 2015.

[3] N. Panwar, S. Sharma, and A. K. Singh, "A survey on 5g: The next generation of mobile communication," *Physical Communication*, vol. 18, pp. 64–84, 2016.

[4] U. N. Kar and D. K. Sanyal, "An overview of device-to-device communication in cellular networks," *ICT express*, vol. 4, no. 4, pp. 203–208, 2018.

[5] L. Lu, G. Y. Li, A. L. Swindlehurst, A. Ashikhmin, and R. Zhang, "An overview of massive mimo: Benefits and challenges," *IEEE journal of selected topics in signal processing*, vol. 8, no. 5, pp. 742–758, 2014.

[6] Y. Niu, Y. Li, D. Jin, L. Su, and A. V. Vasilakos, "A survey of millimeter wave communications (mmwave) for 5g: opportunities and challenges," *Wireless networks*, vol. 21, no. 8, pp. 2657–2676, 2015.

[7] F. Hu, Q. Hao, and K. Bao, "A survey on software-defined network and openflow: From concept to implementation," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 4, pp. 2181–2206, 2014.

[8] B. Han, V. Gopalakrishnan, L. Ji, and S. Lee, "Network function virtualization: Challenges and opportunities for innovations," *IEEE Communications Magazine*, vol. 53, no. 2, pp. 90–97, 2015.

[9] M. M. Raikar, S. Meena, M. M. Mulla, N. S. Shetti, and M. Karanandi, "Data traffic classification in software defined networks (sdn) using supervised-learning," *Procedia Computer Science*, vol. 171, pp. 2750–2759, 2020.

[10] S. A. Repalle and V. R. Kolluru, "Intrusion detection system using ai and machine learning algorithm," *International Research Journal of Engineering and Technology (IRJET)*, vol. 4, no. 12, pp. 1709–1715, 2017.

[11] J. Guo, C.-K. Wen, M. Chen, and S. Jin, "Ai-enhanced codebook-based csi feedback in fdd massive mimo," in *2021 IEEE 94th Vehicular Technology Conference (VTC2021-Fall)*. IEEE, 2021, pp. 1–5.

[12] J. Van Lunteren and A. Engbersen, "Packet classification," Mar. 20 2007, uS Patent 7,193,997.

[13] P. Wang, F. Ye, X. Chen, and Y. Qian, "Datanet: Deep learning based encrypted network traffic classification in sdn home gateway," *IEEE Access*, vol. 6, pp. 55 380–55 391, 2018.

[14] M. Shafiq, X. Yu, A. A. Laghari, L. Yao, N. K. Karn, and F. Abdessamia, "Network traffic classification techniques and comparative analysis using machine learning algorithms," in *2016 2nd IEEE International Conference on Computer and Communications (ICCC)*, 2016, pp. 2451–2455.

[15] J. Zhang, X. Chen, Y. Xiang, W. Zhou, and J. Wu, "Robust network traffic classification," *IEEE/ACM Transactions on Networking*, vol. 23, no. 4, pp. 1257–1270, 2015.

[16] P. Wang, X. Chen, F. Ye, and Z. Sun, "A survey of techniques for mobile service encrypted traffic classification using deep learning," *IEEE Access*, vol. 7, pp. 54 024–54 033, 2019.

[17] J. Zhang, F. Li, H. Wu, and F. Ye, "Autonomous model update scheme for deep learning based network traffic classifiers," in *2019 IEEE Global Communications Conference (GLOBECOM)*, Dec 2019, pp. 1–6.

[18] R. Yuan, Z. Li, X. Guan, and L. Xu, "An svm-based machine learning method for accurate internet traffic classification," *Information Systems Frontiers*, vol. 12, no. 2, pp. 149–156, 2010.

[19] R. Yan and R. Liu, "Principal component analysis based network traffic classification." *J. Comput.*, vol. 9, no. 5, pp. 1234–1240, 2014.

[20] R. Yuan, Z. Li, X. Guan, and L. Xu, "An svm-based machine learning method for accurate internet traffic classification," *Information Systems Frontiers*, vol. 12, pp. 149–156, 2010.

[21] W. Li, P. yi, Y. Wu, L. Pan, and J. Li, "A new intrusion detection system based on knn classification algorithm in wireless sensor network," *Journal of Electrical and Computer Engineering*, vol. 2014, pp. 1–8, 06 2014.

[22] W. Li, X. Li, H. Li, and G. Xie, "Cutsplit: A decision-tree combining cutting and splitting for scalable packet classification," in *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, 2018, pp. 2645–2653.

[23] G. Sun, T. Chen, Y. Su, and C. Li, "Internet traffic classification based on incremental support vector machines," *Mobile Networks and Applications*, vol. 23, no. 4, pp. 789–796, 2018.

[24] S. Santiago Lopes Pereira, J. L. De Castro e Silva, and J. E. Bessa Maia, "Ntcs: A real time flow-based network traffic classification system," in *10th International Conference on Network and Service Management (CNSM) and Workshop*, 2014, pp. 368–371.

[25] Z. Wang, Y. Dong, S. Mao, and X. Wang, "Internet multimedia traffic classification from qos perspective using semi-supervised dictionary learning models," *China Communications*, vol. 14, no. 10, pp. 202–218, 2017.

[26] N. Hubballi and M. Swarnkar, "*bitcoding*: Network traffic classification through encoded bit level signatures," *IEEE/ACM Transactions on Networking*, vol. 26, no. 5, pp. 2334–2346, 2018.

[27] C. Yin, Y. Zhu, J. Fei, and X. He, "A deep learning approach for intrusion detection using recurrent neural networks," *IEEE Access*, vol. 5, pp. 21 954–21 961, 2017.

[28] Y. Xiao, C. Xing, T. Zhang, and Z. Zhao, "An intrusion detection model based on feature reduction and convolutional neural networks," *IEEE Access*, vol. 7, pp. 42 210–42 219, 2019.

[29] K. Wu, Z. Chen, and W. Li, "A novel intrusion detection model for a massive network using convolutional neural networks," *IEEE Access*, vol. 6, pp. 50 850–50 859, 2018.

[30] V. H. C. de Albuquerque, A. R. de Alexandria, P. C. Cortez, and J. M. R. Tavares, "Evaluation of multilayer perceptron and self-organizing map neural network topologies applied on microstructure segmentation from metallographic images," *NDT & E International*, vol. 42, no. 7, pp. 644–651, 2009.

[31] B.-S. Hua, M.-K. Tran, and S.-K. Yeung, "Pointwise convolutional neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 984–993.

[32] Q. Lyu and X. Lu, "Effective media traffic classification using deep learning," in *Proceedings of the 2019 3rd International Conference on Compute and Data Analysis*, 2019, pp. 139–146.

[33] Z. Okonkwo, E. Foo, Q. Li, and Z. Hou, "A cnn based encrypted network traffic classifier," in *Australasian Computer Science Week 2022*, 2022, pp. 74–83.

[34] P. Wang, F. Ye, X. Chen, and Y. Qian, "Datanet: Deep learning based encrypted network traffic classification in sdn home gateway," *IEEE Access*, vol. 6, pp. 55 380–55 391, 2018.

[35] L. Wang, H. Q. Ngo, M. Elkashlan, T. Q. Duong, and K.-K. Wong, "Massive mimo in spectrum sharing networks: Achievable rate and power efficiency," *IEEE Systems Journal*, vol. 11, no. 1, pp. 20–31, 2017.

[36] P.-H. Kuo, H. T. Kung, and P.-A. Ting, "Compressive sensing based channel feedback protocols for spatially-correlated massive antenna arrays," in *2012 IEEE Wireless Communications and Networking Conference (WCNC)*, 2012, pp. 492–497.

[37] C.-K. Wen, W.-T. Shih, and S. Jin, "Deep learning for massive mimo csi feedback," *IEEE Wireless Communications Letters*, vol. 7, no. 5, pp. 748–751, 2018.

[38] J. Guo, C.-K. Wen, S. Jin, and G. Y. Li, "Convolutional neural network-based multiple-rate compressive sensing for massive mimo csi feedback: Design, simulation, and analysis," *IEEE Transactions on Wireless Communications*, vol. 19, no. 4, pp. 2827–2840, 2020.

[39] Z. Cao, W.-T. Shih, J. Guo, C.-K. Wen, and S. Jin, "Lightweight convolutional neural networks for csi feedback in massive mimo," *IEEE Communications Letters*, vol. 25, no. 8, pp. 2624–2628, 2021.

[40] M. Gao, T. Liao, and Y. Lu, "Fully connected feedforward neural networks based csi feedback algorithm," *China Communications*, vol. 18, no. 1, pp. 43–48, 2021.

[41] B. Zhang, H. Li, X. Liang, X. Gu, and L. Zhang, "Fully connected layer-shared network architecture for massive mimo csi feedback," *Electronics Letters*, vol. 58, no. 6, pp. 255–257, 2022.

[42] Y. Jang, G. Kong, M. Jung, S. Choi, and I.-M. Kim, "Deep autoencoder based csi feedback with feedback errors and feedback delay in fdd massive mimo systems," *IEEE Wireless Communications Letters*, vol. 8, no. 3, pp. 833–836, 2019.

[43] P. Ray, S. S. Reddy, and T. Banerjee, "Various dimension reduction techniques for high dimensional data analysis: a review," *Artificial Intelligence Review*, pp. 1–43, 2021.

[44] I. Guyon, S. Gunn, M. Nikravesh, and L. A. Zadeh, *Feature extraction: foundations and applications*. Springer, 2008, vol. 207.

[45] M. A. Hall *et al.*, "Correlation-based feature selection for machine learning," 1999.

[46] L. Yu and H. Liu, "Feature selection for high-dimensional data: A fast correlation-based filter solution," in *Proceedings of the 20th international conference on machine learning (ICML-03)*, 2003, pp. 856–863.

[47] K. Kira, L. A. Rendell *et al.*, "The feature selection problem: Traditional methods and a new algorithm," in *Aaai*, vol. 2, no. 1992a, 1992, pp. 129–134.

[48] Y. Jiang and J. Ren, "Eigenvalue sensitive feature selection," in *ICML*, 2011.

[49] E. Rashedi, H. Nezamabadi-Pour, and S. Saryazdi, "Gsa: a gravitational search algorithm," *Information sciences*, vol. 179, no. 13, pp. 2232–2248, 2009.

[50] M. Radovic, M. Ghalwash, N. Filipovic, and Z. Obradovic, "Minimum redundancy maximum relevance feature selection approach for temporal gene expression data," *BMC bioinformatics*, vol. 18, no. 1, pp. 1–14, 2017.

[51] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *Computer Vision – ECCV 2016*, B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds. Cham: Springer International Publishing, 2016, pp. 525–542.

[52] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or -1," 2016.

[53] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," 2015.

[54] J.-H. Luo, J. Wu, and W. Lin, "Thinet: A filter level pruning method for deep neural network compression," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 5058–5066.

[55] G. Mantena and K. C. Sim, "Entropy-based pruning of hidden units to reduce dnn parameters," in *2016 IEEE Spoken Language Technology Workshop (SLT)*, 2016, pp. 672–679.

[56] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *European conference on computer vision*. Springer, 2016, pp. 525–542.

[57] J. Yim, D. Joo, J. Bae, and J. Kim, "A gift from knowledge distillation: Fast optimization, network minimization and transfer learning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 4133–4141.

[58] Y. LeCun, J. S. Denker, and S. A. Solla, "Optimal brain damage," in *Advances in neural information processing systems*, 1990, pp. 598–605.

[59] J. Zhang, Y. Xiang, Y. Wang, W. Zhou, Y. Xiang, and Y. Guan, "Network traffic classification using correlation information," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 1, pp. 104–117, 2013.

[60] Y. Zeng, H. Gu, W. Wei, and Y. Guo, "$deep - full - range$: A deep learning based network encrypted traffic classification and intrusion detection framework," *IEEE Access*, vol. 7, pp. 45 182–45 190, 2019.

[61] Z. Bu, B. Zhou, P. Cheng, K. Zhang, and Z.-H. Ling, "Encrypted network traffic classification using deep and parallel network-in-network models," *IEEE Access*, vol. 8, pp. 132 950–132 959, 2020.

[62] H. Lim, J. Kim, J. Heo, K. Kim, Y. Hong, and Y. Han, "Packet-based network traffic classification using deep learning," in *2019 International Conference on Artificial Intelligence in Information and Communication (ICAIIC)*, 2019, pp. 046–051.

[63] J. Zhang, F. Li, H. Wu, and F. Ye, "Autonomous model update scheme for deep learning based network traffic classifiers," in *2019 IEEE Global Communications Conference (GLOBECOM)*, 2019, pp. 1–6.

[64] G. D'Angelo and F. Palmieri, "Knowledge elicitation based on genetic programming for non destructive testing of critical aerospace systems," *Future Generation Computer Systems*, vol. 102, pp. 633–642, 2020.

[65] A. Bate, M. Lindquist, I. R. Edwards, S. Olsson, R. Orre, A. Lansner, and R. M. De Freitas, "A bayesian neural network method for adverse drug reaction signal generation," *European journal of clinical pharmacology*, vol. 54, no. 4, pp. 315–321, 1998.

[66] A. Vehtari and J. Lampinen, "Bayesian mlp neural networks for image analysis," *Pattern Recognition Letters*, vol. 21, no. 13-14, pp. 1183–1191, 2000.

[67] J. Zhang, F. Li, and F. Ye, "An ensemble-based network intrusion detection scheme with bayesian deep learning," in *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*, 2020, pp. 1–6.

[68] K. Shridhar, F. Laumann, and M. Liwicki, "A comprehensive guide to bayesian convolutional neural network with variational inference," *arXiv preprint arXiv:1901.02731*, 2019.

[69] M.-H. Chun, S.-J. Han, and N.-I. Tak, "An uncertainty importance measure using a distance metric for the change in a cumulative distribution function," *Reliability Engineering & System Safety*, vol. 70, no. 3, pp. 313–321, 2000.

[70] M. Lotfollahi, M. J. Siavoshani, R. S. H. Zade, and M. Saberian, "Deep packet: A novel approach for encrypted traffic classification using deep learning," *Soft Computing*, vol. 24, no. 3, pp. 1999–2012, 2020.

[71] Wei Wang, Ming Zhu, Xuewen Zeng, Xiaozhou Ye, and Yiqiang Sheng, "Malware traffic classification using convolutional neural network for representation learning," in *2017 International Conference on Information Networking (ICOIN)*, 2017, pp. 712–717.

[72] M. Lopez-Martin, B. Carro, A. Sanchez-Esguevillas, and J. Lloret, "Network traffic classifier with convolutional and recurrent neural networks for internet of things," *IEEE Access*, vol. 5, pp. 18 042–18 050, 2017.

[73] H. Yao, P. Gao, J. Wang, P. Zhang, C. Jiang, and Z. Han, "Capsule network assisted iot traffic classification mechanism for smart cities," *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 7515–7525, 2019.

[74] A. O. Salau and S. Jain, "Feature extraction: A survey of the types, techniques, applications," in *2019 International Conference on Signal Processing and Communication (ICSC)*, 2019, pp. 158–164.

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

[76] D. Arivudainambi, V. K. KA, P. Visu *et al.*, "Malware traffic classification using principal component analysis and artificial neural network for extreme surveillance," *Computer Communications*, vol. 147, pp. 50–57, 2019.

[77] S. Kumar, S. Dharmapurikar, F. Yu, P. Crowley, and J. Turner, "Algorithms to accelerate multiple regular expressions matching for deep packet inspection," *ACM SIGCOMM Computer Communication Review*, vol. 36, no. 4, pp. 339–350, 2006.

[78] Z. Yuan and C. Wang, "An improved network traffic classification algorithm based on hadoop decision tree," in *2016 IEEE International Conference of Online Analysis and Computing Science (ICOACS)*, 2016, pp. 53–56.

[79] S. Patro and K. K. Sahu, "Normalization: A preprocessing stage," *arXiv preprint arXiv:1503.06462*, 2015.

[80] Hao Zheng, Zhanlei Yang, Wenju Liu, Jizhong Liang, and Yanpeng Li, "Improving deep neural networks using softplus units," in *2015 International Joint Conference on Neural Networks (IJCNN)*, 2015, pp. 1–4.

[81] B.-N. Jiang, X.-Q. Ding, L.-T. Ma, Y. He, T. Wang, and W.-W. Xie, "A hybrid feature selection algorithm: Combination of symmetrical uncertainty and genetic algorithms," in *The second international symposium on optimization and systems biology*. Citeseer, 2008, pp. 152–157.

[82] S. S. Kannan and N. Ramaraj, "A novel hybrid feature selection via symmetrical uncertainty ranking based local memetic search algorithm," *Knowledge-Based Systems*, vol. 23, no. 6, pp. 580–585, 2010.

[83] L. A. Kuznar, "Evolutionary applications of risk sensitivity models to socially stratified species: comparison of sigmoid, concave, and linear functions," *Evolution and Human Behavior*, vol. 23, no. 4, pp. 265–280, 2002.

[84] Z. Boger and H. Guterman, "Knowledge extraction from artificial neural network models," in *1997 IEEE International Conference on Systems, Man, and Cybernetics. Computational Cybernetics and Simulation*, vol. 4. IEEE, 1997, pp. 3030–3035.

[85] S. Targ, D. Almeida, and K. Lyman, "Resnet in resnet: Generalizing residual architectures," *arXiv preprint arXiv:1603.08029*, 2016.

[86] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," 2017.

[87] J. Zhang, F. Li, F. Ye, and H. Wu, "Autonomous unknown-application filtering and labeling for dl-based traffic classifier update," in *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*, 2020, pp. 397–405.

[88] T. van Erven and P. Harremos, "Rényi divergence and kullback-leibler divergence," *IEEE Transactions on Information Theory*, vol. 60, no. 7, pp. 3797–3820, 2014.

[89] J. S. Hunter, "The exponentially weighted moving average," *Journal of quality technology*, vol. 18, no. 4, pp. 203–210, 1986.

[90] S. Karsoliya, "Approximating number of hidden layer neurons in multiple hidden layer bpnn architecture," *International Journal of Engineering Trends and Technology*, vol. 3, no. 6, pp. 714–717, 2012.

[91] B. Senliol, G. Gulgezen, L. Yu, and Z. Cataltepe, "Fast correlation based filter (fcbf) with a different search strategy," in *2008 23rd international symposium on computer and information sciences*. IEEE, 2008, pp. 1–4.

[92] Y. Zhang, C. Ding, and T. Li, "Gene selection algorithm by combining relieff and mrmr," *BMC genomics*, vol. 9, no. 2, pp. 1–10, 2008.

[93] N. Rachburee and W. Punlumjeak, "A comparison of feature selection approach between greedy, ig-ratio, chi-square, and mrmr in educational mining," in *2015 7th International Conference on Information Technology and Electrical Engineering (ICITEE)*, 2015, pp. 420–424.

[94] A. Zanella, N. Bui, A. Castellani, L. Vangelista, and M. Zorzi, "Internet of things for smart cities," *IEEE Internet of Things Journal*, vol. 1, no. 1, pp. 22–32, 2014.

[95] L. Chettri and R. Bera, "A comprehensive survey on internet of things (iot) toward 5g wireless systems," *IEEE Internet of Things Journal*, vol. 7, no. 1, pp. 16–32, 2020.

[96] Y. Wan, K. Xu, F. Wang, and G. Xue, "Characterizing and mining traffic patterns of iot devices in edge networks," *IEEE Transactions on Network Science and Engineering*, pp. 1–1, 2020.

[97] H. Tahaei, F. Afifi, A. Asemi, F. Zaki, and N. B. Anuar, "The rise of traffic classification in iot networks: A survey," *Journal of Network and Computer Applications*, vol. 154, p. 102538, 2020.

[98] A. M. Ghosh and K. Grolinger, "Deep learning: Edge-cloud data analytics for iot," in *2019 IEEE Canadian Conference of Electrical and Computer Engineering (CCECE)*, 2019, pp. 1–7.

[99] U. Cisco, "Cisco annual internet report (2018–2023) white paper," 2020.

[100] Y. Meidan, M. Bohadana, A. Shabtai, J. D. Guarnizo, M. Ochoa, N. O. Tippenhauer, and Y. Elovici, "Profiliot: a machine learning approach for iot device identification based on network traffic analysis," in *Proceedings of the symposium on applied computing*, 2017, pp. 506–509.

[101] J. Van Lunteren and A. Engbersen, "Packet classification," Mar. 20 2007, uS Patent 7,193,997.

[102] J. Zhang, F. Ye, and Y. Qian, "A distributed network qoe measurement framework for smart networks in smart cities," in *2018 IEEE International Smart Cities Conference (ISC2)*, 2018, pp. 1–7.

[103] B. Indira, K. Valarmathi, and D. Devaraj, "An approach to enhance packet classification performance of software-defined network using deep learning," *Soft Computing*, vol. 23, no. 18, pp. 8609–8619, 2019.

[104] S. Hajiheidari, K. Wakil, M. Badri, and N. J. Navimipour, "Intrusion detection systems in the internet of things: A comprehensive investigation," *Computer Networks*, vol. 160, pp. 165–191, 2019.

[105] S. Dharmapurikar, P. Krishnamurthy, T. Sproull, and J. Lockwood, "Deep packet inspection using parallel bloom filters," in *11th Symposium on High Performance Interconnects, 2003. Proceedings.*, 2003, pp. 44–51.

[106] H. Liu, B. Lang, M. Liu, and H. Yan, "Cnn and rnn based payload classification methods for attack detection," *Knowledge-Based Systems*, vol. 163, pp. 332–341, 2019.

[107] W. Tang, G. Hua, and L. Wang, "How to train a compact binary neural network with high accuracy?" in *Thirty-First AAAI conference on artificial intelligence*, 2017.

[108] G. Draper-Gil, A. H. Lashkari, M. S. I. Mamun, and A. A. Ghorbani, "Characterization of encrypted and vpn traffic using time-related," in *Proceedings of the 2nd international conference on information systems security and privacy (ICISSP)*, 2016, pp. 407–414.

[109] R. Doshi, N. Apthorpe, and N. Feamster, "Machine learning ddos detection for consumer internet of things devices," in *2018 IEEE Security and Privacy Workshops (SPW)*, 2018, pp. 29–35.

[110] M. Lopez-Martin, B. Carro, A. Sanchez-Esguevillas, and J. Lloret, "Network traffic classifier with convolutional and recurrent neural networks for internet of things," *IEEE Access*, vol. 5, pp. 18 042–18 050, 2017.

[111] L.-H. Chang, T.-H. Lee, H.-C. Chu, and C. Su, "Application-based online traffic classification with deep learning models on sdn networks," *Adv. Technol. Innov*, vol. 5, pp. 216–229, 2020.

[112] P. Wang, F. Ye, X. Chen, and Y. Qian, "Datanet: Deep learning based encrypted network traffic classification in sdn home gateway," *IEEE Access*, vol. 6, pp. 55 380–55 391, 2018.

[113] M. J. Berry and G. S. Linoff, *Data mining techniques: for marketing, sales, and customer relationship management.* John Wiley & Sons, 2004.

[114] D. Johnson and S. Sinanovic, "Symmetrizing the kullback-leibler distance," *IEEE Transactions on Information Theory*, 2001.

[115] A. Kulmala, M. Alonso, S. Repo, H. Amaris, A. Moreno, J. Mehmedalic, and Z. Al-Jassim, "Hierarchical and distributed control concept for distribution network congestion management," *IET Generation, Transmission Distribution*, vol. 11, no. 3, pp. 665–675, 2017.

[116] X. Fan, J. Zhang, L. Guan, and X. Wang, "Qblue: A new congestion control algorithm based on queuing theory," in *2012 IEEE 14th International Conference on High Performance Computing and Communication 2012 IEEE 9th International Conference on Embedded Software and Systems*, 2012, pp. 1253–1257.

[117] G. Draper-Gil, A. H. Lashkari, M. S. I. Mamun, and A. A. Ghorbani, "Characterization of encrypted and vpn traffic using time-related," in *Proceedings of the 2nd international conference on information systems security and privacy (ICISSP)*, 2016, pp. 407–414.

[118] J. Zhang, F. Li, F. Ye, and H. Wu, "Autonomous unknown-application filtering and labeling for dl-based traffic classifier update," 2020.

[119] R. J. Little, "Modeling the drop-out mechanism in repeated-measures studies," *Journal of the american statistical association*, vol. 90, no. 431, pp. 1112–1121, 1995.

[120] H. Zhang, Y. Li, V. Stolpman, and N. Van Waes, "A reduced csi feedback approach for precoded mimo-ofdm systems," *IEEE Transactions on Wireless Communications*, vol. 6, no. 1, pp. 55–58, 2007.

[121] C.-K. Wen, W.-T. Shih, and S. Jin, "Deep learning for massive mimo csi feedback," *IEEE Wireless Communications Letters*, vol. 7, no. 5, pp. 748–751, 2018.

[122] Z. Lu, J. Wang, and J. Song, "Multi-resolution csi feedback with deep learning in massive mimo system," in *ICC 2020-2020 IEEE International Conference on Communications (ICC)*. IEEE, 2020, pp. 1–6.

[123] X. Yu, X. Li, H. Wu, and Y. Bai, "Ds-nlcsinet: Exploiting non-local neural networks for massive mimo csi feedback," *IEEE Communications Letters*, vol. 24, no. 12, pp. 2790–2794, 2020.

[124] J. Guo, C.-K. Wen, S. Jin, and G. Y. Li, "Convolutional neural network-based multiple-rate compressive sensing for massive mimo csi feedback: Design, simulation, and analysis," *IEEE Transactions on Wireless Communications*, vol. 19, no. 4, pp. 2827–2840, 2020.

[125] D. Tse and P. Viswanath, *Fundamentals of wireless communication*. Cambridge university press, 2005.