SCHOOL OF GRADUATE STUDIES

NETWORK TRAFFIC CLASSIFICATION OF SOFTWARE DEFINED NETWORK USING DEEP LEARNING

MSc Thesis

BETEMICHAEL KASSAYE MIHRETU

APRIL, 2024

WOLKITE, ETHIOPIA

Wolkite University
School of Graduate Studies

Network Traffic Classification of Software Defined Network Using Deep Learning

A Thesis Submitted to School of Graduate Studies, in Partial Fulfilment of the Requirements for the Degree of Master of Computer Science and Engineering (Specialization: Computer Science)

Betemichael Kassaye Mihretu

Major Advisor: Negalign Wake Hundera (Ph.D.)
Co-Advisor: Amanuel Tamirat Getachew

APRIL, 2024

WOLKITE, ETHIOPIA

APPROVAL SHEET

WOLKITE UNIVERSITY

SCHOOL OF GRADUATE STUDIES

We hereby certify that we have read and evaluated this Thesis titled "**Network Traffic Classification of Software Defined Network Using Deep Learning**" prepared under our guidance by **Betemichael Kassaye Mihretu**. We recommend that the Thesis shall be submitted as fulfilling the requirements for the award of a MSc. degree in Computer Science and Engineering.

1.	Negalign Wake Hundera (Ph.D.)	_____	April 13,2024
	Major Advisor	Signature	Date
2.	Amanuel Tamirat Getachew	_____	April 15,2024
	Co-Advisor	Signature	Date

As members of the Board of Examiners of the Master of Science Thesis open defense examination, we have read and evaluated this Thesis prepared by **Betemichael Kassaye Mihretu** and examined the candidate. we hereby certify that; the thesis is accepted for fulfilling the requirements for the award of the degree of Master of Science (M.Sc.) in Computer Science and Engineering.

1.	_____	_____	_____
	Name of External Examiner	Signature	Date
2.	_____	_____	_____
	Name of Internal Examiner	Signature	Date
3.	_____	_____	_____
	Name of Chairman	Signature	Date

Final approval and acceptance of the Thesis is contingent upon the submission of its final copy to the Council of Postgraduate Program (CPGS) through the candidate's department or school graduate committee (DGC or SGC)

DECLARATION

I hereby declare that this thesis is my own work and that it has not been submitted for any degree or examination at any other university. Wherever I have referred to the work of others, I have cited the source. No part of this thesis has been published or submitted for publication elsewhere. I understand that failure to comply with this requirement will result in the rejection of my thesis.

The thesis has been submitted in partial fulfillment of the requirements for the degree of Master of Science degree in Computer Science and Engineering.

With accurate acknowledgment of the source, users are free to use this thesis without special permission.

Betemichael Kassaye		April 15, 2024
Name	Signature	Date

As a thesis research advisor, I hereby approve that I have read and evaluated this thesis prepared, under my guidance, by Betemichael Kassaye, entitled “Network Traffic Classification of Software Defined Network Using Deep Learning”. I recommend it be submitted as fulfilling the thesis requirement.

Dr. Negalign Wake		April 13, 2024
Name of advisor	Signature	Date

ACKNOWLEDGMENTS

First and foremost, I would like to thank my research supervisor, Dr. Negalign Wake Hundera, for his guidance, support, and encouragement throughout the research process. His expertise and insights have greatly contributed to the quality of this study. I would also like to extend my heartfelt gratitude to my colleagues and friends who have provided me with valuable feedback, constructive criticism, and insights that have helped me improve this research. Their contributions have been valuable in shaping the direction and scope of this study.

I would also like to thank Wolkite University for providing me with the necessary resources, facilities, and funding that have enabled me to conduct this research. I am also grateful to all the participants who have generously given their time and effort to take part in this study.

And finally, I would like to thank my families for their unwavering support, love, and encouragement throughout this research. Their understanding, patience, and sacrifice have been instrumental in my success. Without the help and support of all these people, this research would not have been possible. I am truly grateful for everything they have done for me.

ABBREVATIONS

Adam	Adaptive Moment Estimation
AI	Artificial Intelligence
BERT	Bidirectional Encoder Representations Transformers
GPT	Generative Pretrained Transformer
CNN	Convolution Neural Network
DL	Deep Learning
DNN	Deep Neural Network
DNP	Distributed Network Protocol
DPI	Deep Packet Inspection
FTP	File Transfer Protocol
IDS	Intrusion Detection System
IoT	Internet of Things
IP	Internet Protocol
KNN	K-Nearest Neighbors
ISP	Internet Service Provider
LSTM	Long Short-Term Memory
ML	Machine Learning
NLP	Natural Language Processing
QoS	Quality of Service
ResNet	Residual Network
RNN	Recurrent Neural Network
SDN	Software-Defined Networking
SAE	Staked Auto Encoder
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UDP	User Data Protocol
VPN	Virtual Private Network
VAE	Variational Auto Encoder

Table of Content

List of tables

List of Figures

ABSTRACT

In order to cope with the monumental growth in network traffic, the field of networking is continuously progressing to accommodate this monumental growth in network traffic. As a matter of fact, a centralized control mechanism is provided by architectures such as Software Defined Networking (SDN) for the measurement, control, and prediction of network traffic, but the amount of information that the SDN controller receives is enormous. Recently, it has been suggested that machine learning (ML) is used to process that data. In fact, it is crucial to fine-grained network management, resource utilization, network security that, network traffic classification is used in a variety of network activities. To classify and analyze network traffic flows, the port-based approach, deep packet inspection, and ML are among the most widely used methods. Nevertheless, over the past several years, there has been an explosion in the number of users of the Internet, which has led to an explosive increase in Internet traffic. The exponential growth of Internet applications, which incur high computational costs, has made port-based, deep packet inspection (DPI), and ML approaches inefficient. It has been found that software-defined networking is redefining the network architecture by separating the control plane from the data plane and resulting in the creation of a centralized network controller that maintains a global view of the entire network. The aim of this paper is to propose a new deep learning model for software-defined networks able to accurately predict a wide range of traffic applications in a short time-frame to improve efficiency. In contrast to traditional ML approaches, the proposed model has been able to achieve better results in terms of accuracy, precision, recall, and F1-Score when compared with the traditional approaches. The performance metrics result from deep learning model indicates accuracy of 90.7%, F1-Score of 91%, Precision consistently of above 92%, Recall 88% and testing accuracy 92% respectively. It has been suggested that some further directions should be pursued to achieve future advances in this field based upon the results obtained.

KEYWORDS:

Traffic classification, Software-defined networking, Artificial Intelligence, Deep Neural Network, Staked Auto Encoder, Random Forest

CHAPTER 1

1. INTRODUCTION

1.1. Background

Software-defined networking (SDN) is a network architecture that allows network administrators to manage network services through abstraction of lower-level functionality [1]. SDN has revolutionized the way networks are managed, making it easier to deploy, manage, and secure networks. Network traffic classification is the process of identifying the type of traffic flowing through a network. It is essential for network management, security, and performance optimization. In recent years, deep learning (DL) has emerged as a powerful tool for network traffic classification. SDN allows logic and control to be decoupled from physical networking devices such as switches, routers, and gateways. SDN enables centralized network management, provisioning, and monitoring using a software controller, which configures and manages network devices through a programmable interface. SDN has gained popularity due to its ability to abstract the complexity of network management and simplify network management tasks. And it is a critical task in network management which involves identifying the type of traffic flowing through a network, such as web traffic, video traffic, or voice traffic. Network traffic classification is essential for several reasons. It helps network administrators to optimize network performance, detect and prevent network attacks, and prioritize network traffic. Traditional methods of network traffic classification rely on packet inspection, port-based classification, and statistical analysis [2]. However, these methods have limitations, such as low accuracy and inability to classify encrypted traffic.

The rapid development of Internet, communication devices and networking technologies has provided exponential growth of network traffic and has created bigger and more complicated network structures [3]. This complexity arises from adaption of highly powerful router, firewalls, servers, switches, etc. in networks which has introduced an overflow of vast amounts of traffic data and contributed to the challenges in network management and traffic optimization, including traffic measurement.

Distribution of traffic in an optimized way and managing large volume of heterogeneous devices is a complex task. Deployment of intelligence in network can solve the above issues. Knowledge plane had been introduced to incorporate intelligence, automation, and recommendation to the network through ML techniques. But distributed nature of traditional network systems had made the process complex. To reduce the complexity of learning, SDN paved a way for it. It is an innovative architecture which decouples control plane from data plane [4]. The centralized control plane is responsible for routing and management policies. The data plane forwards the packet only through protocols like Open-flow. However, the amount of data an SDN controller receives could be overwhelming. While the SDN controller itself is made scalable, for example by running it in a cloud. But still efficient algorithms like ML and DL are needed to extract the required measurements and information from the received data. Many of the traffic classification and traffic prediction issues is performed efficiency by various ML and DL algorithms, improving the system performance while maintaining relative simplicity in design.

In this paper, the proposed model is classifying SDN network traffic using DL techniques which is a branch of ML algorithms and is applied as a learning algorithm that automatically learns the features from the raw network traffic and outputs the classified labels directly. With the emerging paradigm of SDN as a clean-slate approach to redesign the network architecture [5], in which the control plane is decoupled from the data plane, the logically centralized controller maintains a global view of the entire network. This gives a great opportunity for ML and DL techniques to be applied and learn from the network traffic data. And take the advantage of a SDN architecture and propose a new framework that utilizes the DL towards identifying network traffic applications. DL is a promising approach in dealing with significant massive data towards extracting the hidden patterns, e.g., traffic features, for classification and prediction purposes. Our main goal is to build a SDN network model that capable of achieving high prediction accuracy in terms of network traffic identification and classification.

DL is a subset of ML that has gained popularity in recent years. It involves training artificial neural networks to learn from large datasets [6]. DL has been applied to several fields, including computer vision, NLP, and speech recognition. DL has also shown promise in network traffic classification. DL models can learn the features of network traffic automatically, without the need for handcrafted features. This makes DL models more accurate and robust than traditional methods. In recent years, DL has also been applied in

network traffic classification, where it is used to classify network traffic with high accuracy and efficiency. DL models are designed to learn from a large set of data, and network traffic data is no exception. With the increasing demand for DL in network traffic classification, SDN has been identified as an ideal platform for deploying DL models due to its flexibility and programmability. An SDN-based traffic classification system can utilize the programmable interface of the SDN controller to collect traffic data from network devices, process the data using a DL model, and then classify the traffic based on its characteristics [7].

One of the advantages of using DL in SDN-based traffic classification is its ability to handle large and complex datasets, which are difficult to classify using traditional methods [8]. Furthermore, DL models can adapt to changes in network traffic patterns, making them suitable for dynamic and unpredictable network environments. However, the effectiveness of DL depends on the quality, quantity, and diversity of the training data, which requires careful selection and preparation.

Overall, DL has immense potential for Network Traffic Classification in SDN environments [9]. Its ability to handle complex, large-scale datasets and to adapt to dynamic network environments makes it an ideal technology for accurate and efficient network traffic classification.

1.1.1. Network Traffic Classification and ML Concepts

Network traffic classification is the process of identifying the type of traffic flowing through a network. In its broadest definition, network traffic classification refers to a system in which a program assigns traffic flows to the sources (e.g., applications and protocols) that produce them [10]. Traffic classification has attracted ever-increasing interest over the years as a crucial step towards the network management process. Moreover, traffic classification covers a wide variety of applications in Quality of Service (QoS) purposes, pricing in Internet service providers (ISPs), anomaly detection, etc. Due to the continuing growth in Internet-based applications and the number of connected devices, applying efficient traffic classification methods is critically important. This classification is based on various parameters, such as port numbers, protocols, packet sizes, and so on. The importance of network traffic classification lies in its ability to help network administrators manage and optimize network resources [11]. By classifying network traffic, administrators can identify which applications are consuming network resources and take measures to

manage traffic flow more efficiently. One area where network traffic classification is particularly important is in SDN. SDN is an emerging technology that allows network administrators to centrally manage network resources through software instead of using traditional hardware-based methods. SDN is becoming increasingly popular due to its ability to improve network performance, security, and flexibility. The following section discusses an overview of machine learning, deep learning and transfer learning models and their application in the area of networking.

1.1.2. Machine Learning

A sub field of artificial intelligence (AI), where the human agent enters data and the expected results from the data, and then the learning model yields the rules. Then, these rules are applied to new data to achieve original results. ML systems are trainable rather than explicitly programmable [3]. This means a massive amount of data feed into these systems to find meaningful features in this data. Then, these features are used to produce rules for automating the task. For the classical statistical analysis, such as Bayesian analysis, it is almost impossible to handle such big datasets. ML models basically categorized into supervised learning, unsupervised learning, and reinforcement learning. ML based models are broadly divided into two categories: simple (i.e., single) and ensemble models. Ensemble models aim to combine heterogeneous or homogeneous models (commonly classifiers) to obtain a model that outperforms every one of them and overcome their limitations. A heterogeneous ensemble has been proposed to take advantage of the diversity of the models' strengths [12]. The blending and majority voting are heterogeneous models. Blending consists of two levels: base classifier and meta-classifier. Using a meta-classifier, that combines the base classifier, blending is a powerful ensemble technique. However, little attention has been paid to the application of the blending ensemble in network traffic classification. Furthermore, it is well-known that DL outperforms the other shallow ML models in several fields like route-optimization, load balancing and network resource management, and has shown success in network traffic classification. DL is a branch of ML that evolved from Neural Networks (NNs) and has a unique nature and features for solving complex problems. In addition, the DT algorithm is known due to its simplicity and is easily understood by human experts [13]. It is one of the most suitable learning algorithms for network traffic classification. Despite the success of DL models, ML models are usually applied when it comes to the analysis of network traffic

measurements. ML models rely on the definition of highly specialized, expert-handcrafted features to achieve proper results, and this is critical for ML. There are different problems when addressing network traffic measurements tasks using shallow ML approaches [14]. First, the lack of a consensual labeled raw traffic, full-packet capture dataset to train ML models (e.g., due to privacy policies in the data); second, the lack of a consensual set of input features to tackle specific targets, such as network security, anomaly detection, traffic classification, etc.; third, the continuous changing in network measurements statistics that may cause static handcrafted features to fail. The following section discuses categories of ML algorithms:

Machine learning models can be categorized into several types, each with its own approach to learning from data and making predictions or decisions:

1.1.2.1. Supervised learning

These models are trained on labeled datasets, which means they learn to predict outcomes based on input-output pairs. They are used for tasks like classification (predicting discrete labels) and regression (predicting continuous values). Examples include decision trees, support vector machines, and neural networks.

Decision trees are a type of ML algorithm that are commonly used for classification tasks. In the field of networking, decision trees is used for traffic classification and intrusion detection [15].

SVMs are a type of ML algorithm that are used for classification and regression tasks. In the field of networking, SVMs is used for traffic classification and intrusion detection [16] .

Random forests are a type of ensemble learning algorithm that combine multiple decision trees to improve classification accuracy [17]. In the field of networking, random forests are used for traffic classification and intrusion detection.

1.1.2.2. Unsupervised learning

These types of models work with unlabeled data and aim to find hidden patterns or intrinsic structures in the input data. They are often used for clustering, dimensionality reduction, and association rule learning. Common algorithms include k-means clustering, principal component analysis (PCA), and Priory algorithm.

KNN is a ML algorithm that is commonly used for classification and regression tasks [18]. In the field of networking, KNN is used for traffic classification and intrusion detection.

Adaptive Boosting (AdaBoost) It is a boosting technique that builds more accurate algorithms by creating a hybrid classifier out of weak classifiers [19].Linear Regression used for prediction and forecasting purposes, it tries to fit a model to data points based on independent variables [20].

1.1.2.3. Semi-supervised learning

This category combines elements of both supervised and unsupervised learning. Models are trained on a mix of labeled and unlabeled data, which is particularly useful when acquiring labeled data is expensive or time-consuming.

1.1.2.4. Reinforcement learning

In this type of learning, models learn to make decisions by taking actions in an environment to maximize some notion of cumulative reward. It's widely used in areas such as robotics, gaming, and navigation.

1.1.2.5. Self-Supervised Learning

A form of unsupervised learning where the data provides the supervision. The model is trained to predict any part of its input from any other part, effectively creating its own labels from the input data. Naive Bayes is a probabilistic ML algorithm that is commonly used for classification tasks. In the field of networking, Naive Bayes is used for traffic classification and intrusion detection [21]. Each type of machine learning model has its own set of algorithms and applications, and the choice of model depends on the specific problem and the nature of the data available.

Overall, ML algorithms offer a powerful tool for analyzing and processing large amounts of network traffic data. By leveraging the capabilities of these algorithms, network administrators can optimize network performance, detect security threats, and improve overall network efficiency.

1.1.3. Categories of Deep Learning

AI has attracted lots of interest in recent years for many use cases, such as self-driving cars, chat bots, virtual assistants, etc. AI's history goes back to the 1950s, when researchers tried to automate intellectual tasks that humans normally perform [1]. For a very long time, many experts were arguing that by formulating a large set of explicit rules for manipulating knowledge, they can realize human-like artificial intelligence [22]. DL is a specific sub-field of ML, in which Deep Neural Network (DNN) is used to find data representation at each

layer. Generally, DL algorithms represent two significant improvements upon the classical ML methods.

- They eliminate the need for the feature engineering phase via deploying automatic feature learning. Hence, some useful and meaningful features that might not be apparent to manual feature engineering approaches is obtained easily by DL algorithms.
- DL algorithms improve learning performance in terms of accuracy and loss through learning hidden and high-level patterns from data. It is achieved by feeding a huge volume of data into DL models.

Deep learning models can be broadly categorized into three main types:

Generative Models: are used to generate new data instances that are similar to the training data. They can capture the data distribution and are often used for tasks like image generation, text-to-image synthesis, and more. An example of a generative model is the Generative Adversarial Network (GAN).

Discriminative Models: These models are used for classification or regression tasks. They aim to distinguish between different categories or predict a continuous value. Convolutional Neural Networks (CNNs) for image recognition and Recurrent Neural Networks (RNNs) for sequence data like text or speech are examples of discriminative models.

Hybrid Models: Combine aspects of both generative and discriminative models. They can perform a variety of tasks, such as semi-supervised learning, where they leverage both labeled and unlabeled data for training.

Each of these models uses neural networks with multiple layers to process and learn from large amounts of data, allowing them to identify patterns and make predictions. Some of DL models are discussed:

1.1.3.1. Multi-layer Perceptron

A well-known category of a DL model is the feed-forward deep network or multi-layer perceptron (MLP). An MLP model is an artificial neural network mapping some examples of input data to target values. The network is formed by composing multiple simple layers [23].

1.1.3.2. Convolutional Networks

CNN is a multi-layer neural network which implements feature extraction and then applies classification. It consists of processing layer, convolution layer, pooling layer, and classification layer [5]. It uses MLP to do computational tasks and uses filters for learning. The activation function of this network is an instantiation parameter of a particular type of an entity.

1.1.3.3. Long short-term memory (LSTM)

LSTM contain memory cell which retains its information for long or short time based on the function of input value. It consists of three gates namely input gate, forget gate, and output gate. The simplified form of LSTM is gated recurrent unit which lacks output gate. The two gates are update and reset gates. Dynamic network routing based on LSTM predict the Internet traffic with high accuracy. LSTM models long range dependencies more accurately than RNN. In order to identify time related characteristics of the traffic, LSTM is applied [24]. It classifies the network traffic based on time features of the network traffic.

1.1.3.4. RNN

In RNN, output from the previous step is given as input to the next step. The feedback mechanism is provided by hidden layers, which remember some information about the sequences. Since RNN have memory, it remembers all the information calculated so far. It is less complex because it uses same parameters for each input or hidden layers to produce output [25]. RNN can use self-loops to store the gradient of recent input events for long duration. This is the core idea of LSTM model. This feature is potentially important for a wide spectrum of applications, such as speech recognition, handwriting recognition, machine translation, handwriting generation, image captioning and parsing.

1.1.3.5. Auto-Encoders

In the most general sense, an auto-encoder or AE is a neural network that is used to efficiently learn how to copy its inputs to its outputs. AE has a hidden layer, called h, which is responsible for describing a code that stands for the input [26].

1.1.3.6. Deep Belief Networks

DBN is a multi-layer neural network with training algorithm. In DBN, each pair of hidden layers is a Restricted Boltzmann machine (RBM). Hence DBN is represented as stack of RBMs. There are two phases of training namely unsupervised pretraining and supervised fine tuning [27].

1.1.4. Categories of Transfer learning

This section discusses transfer learning models and their applications for networking and enhancements brought to the field.

Transfer learning is an important technique in DL that allows the transfer of knowledge from one domain to another. In the context of network traffic classification, transfer learning can be used to leverage pre-trained models on large datasets such as ImageNet to improve the performance of models on smaller datasets. This is particularly useful when dealing with scarce and unbalanced datasets, which are common in network traffic classification. Transfer learning can also help reduce the training time and computational resources required to train DL models for network traffic classification [27]. By using pre-trained models as a starting point, transfer learning can help speed up the convergence of models and reduce the risk of over-fitting.

In addition, transfer learning can help improve the accuracy and robustness of DL models for network traffic classification by enabling them to learn more general features that are applicable across different domains. This can help reduce the risk of concept drift and improve the ability of models to generalize to new data.

1.1.4.1. Categories of transfer learning

Based on mechanisms that are used to categorize transfer learning models are “when to transfer", "what to transfer" and "how to transfer" are three approaches.

Feature-based transfer learning:

Feature-based transfer learning assumes the data of the source and target domain is mapped into a common feature space through a certain feature representation. It is crucial to learn a pair of mapping functions that project source and target domain data into the common feature space. There are three main methods to find these functions: first, reducing domain differences; second, finding the universal feature between domains; third, augmenting feature across domains through latent relationships [27].

Model-based transfer learning:

For model-based transfer learning, the source domain and the target domain can share models or model parameters. It assumes there is the universal knowledge of source domain task and target domain task. According to the assumptions behind transfer learning, it is

divided into two categories. One is sharing the parameter between domains and the other is using regularization.

Relation-based transfer learning:

The relation-based transfer learning approach is to find out the implied relationship between the source and target domains and transfer the source domain knowledge to the target domain according to certain rules. It assumes that there is a common relationship between the source and target domain data.

In the following section some of the most common transfer learning models and their applications are discussed.

1.1.4.2. VGG16 and VGG19

VGG is a convolutional neural network architecture designed for image classification tasks. VGG16 consists of 16 convolutional layers followed by 3 fully connected layers, while VGG19 has 19 convolutional layers [28]. These models are known for their simplicity and uniformity, using small 3x3 convolutional filters throughout the network. VGG models have achieved state-of-the-art results in various image classification challenges.

1.1.4.3. ResNet

ResNet is a DNN architecture known for its ability to train very deep networks effectively. It introduces skip connections or shortcuts that allow the model to bypass previous layers, addressing the vanishing gradient problem. ResNet typically consists of residual blocks, which contain multiple convolutional layers and identity mappings. By using these skip connections, ResNet can train networks with hundreds or even thousands of layers.

1.1.4.4. Inception

Inception, also known as GoogLeNet, is a convolutional neural network architecture designed to be computationally efficient. It introduces inception modules, which are composed of multiple convolutions with different filter sizes and pooling operations. Inception modules help capture features at different scales by performing different operations in parallel. Inception has been successful in various computer vision tasks and has won the ImageNet Large Scale Visual Recognition Challenge.

1.1.4.5. MobileNet

MobileNet is a convolutional neural network architecture designed for running on mobile and embedded devices. It focuses on reducing the computational complexity and model

size while maintaining reasonable accuracy. MobileNet achieves this by using depth wise separable convolutions, separating the spatial and channel-wise convolutions. This architecture allows for significantly fewer parameters and lower computational requirements while preserving performance.

1.1.4.6. BERT

BERT is a pre-trained model developed by Google. It is based on the Transformer architecture and is designed to understand the context of words in a sentence [29]. Unlike traditional language models that read text sequentially, BERT uses a masked language modeling objective that randomly masks words in a sentence and learns to predict those words. This allows BERT to capture bidirectional context and understand the meaning of words based on their surrounding context. BERT has demonstrated state-of-the-art performance in various NLP tasks, such as sentiment analysis, named entity recognition, question answering, and text classification.

1.1.4.7. GPT-2

GPT-2, developed by OpenAI, is a widely known example of a generative language model. It is trained to predict the next word in a sentence given the previous words. GPT-2 uses a Transformer architecture comprising stacked encoder-decoder layers [30]. It is trained on a large corpus of text from the internet, enabling it to generate coherent, human-like text. GPT-2 is fine-tuned on specific tasks or be used to generate creative text like stories, poems, and even code. It has been influential in several NLP applications but has also raised concerns about the potential for fabricated or misleading content.

1.1.4.8. AlexNet

AlexNet is a CNN architecture designed for image classification. It was developed by Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton and won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in 2012 [31]. AlexNet consists of eight layers, including five convolutional layers and three fully connected layers. It utilizes techniques like ReLU activation, local response normalization, and dropout regularization.

1.2. SDN Concepts

1.2.1. Architecture of SDN

SDN is an architecture that separates the network control plane from the data plane, allowing for centralized control and programmability of the network [19]. SDN architecture, consists of three planes namely data plane, control plane and application plane. As per Open Networking Foundation (ONF), control plane is decoupled from data plane, network automation and intelligence are specified to centralized control plane and provides abstraction of network infrastructure to the application [32]. The architecture component of each plane is shown in Figure 1.

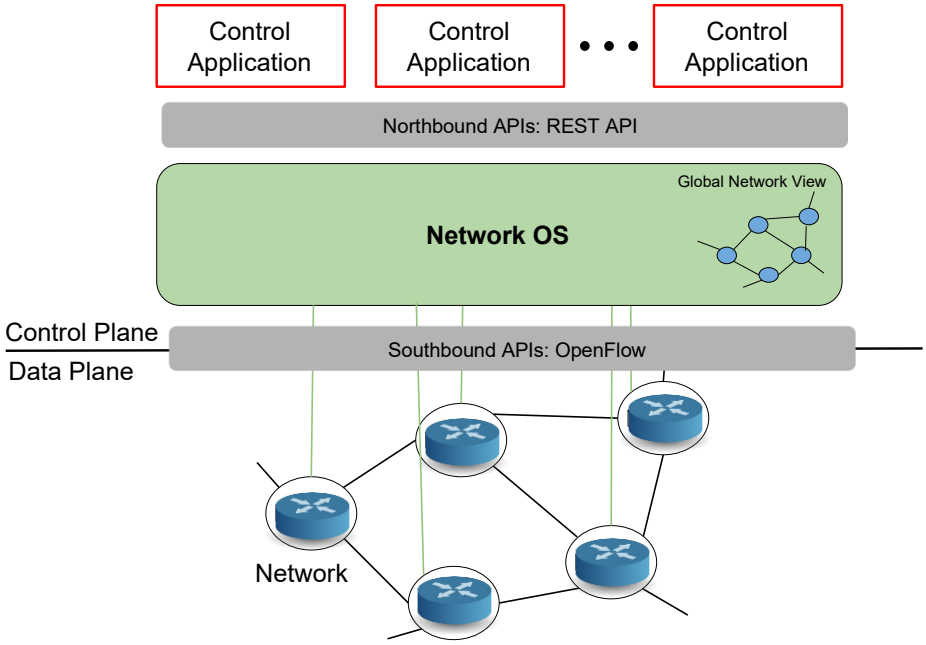

Figure 1: SDN architecture

1.2.2. Data plane

Data plane is the lowest layer in SDN architecture. It consists of network forwarding elements like physical switches and logical switches (virtual). Software switches are virtual switches which run on common operating systems like Linux. Some of the virtual switch implementations are Indigo, Open vSwitch and Pantou. The complete features of SDN protocol are supported by virtual switches whereas physical switches lack flexibility and feature completeness. The major responsibility of switches in data plane are forward, drop and modify packets based on flow rules received from the controller in the control plane. The communication protocol is Open-flow.

1.2.3. Control plane

The brain of the SDN architecture is the SDN controller present in the control plane. It is centralized controller provides the dynamic programming of network resources, updating of flow rules and make network administration flexible and agile. The information about network state of data plane is provided to application plane by the control plane. The application requirements are translated into policies and distributed to forwarding devices through control plane. The controller also provides functionalities like routing, network topology storage, device configuration, state information and so on. The various controllers available are NOX, POX, Ryu, Open-daylight, Floodlight, Beacon, etc.[33]. Through Open-flow protocol, the centralized control plane communicates with the network devices. Flow tables which contain flow entries in the Open Flow switches provide the forwarding path for the flows. The fields of the flow table are match fields and associated actions. The native features of flow like source IP, Destination IP, and header data. And includes other statistical features like number of bytes, duration and so on. SDN realizes traffic classification and feature selection. The SDN controller provides the global view of the network through which classification of traffic is performed. The effective matching of incoming packet is based on rule set.

1.2.4. Application plane

The layer in the SDN architecture composed of business application is called application plane. The services in the application plane provide business management and optimization. The business services for applications are provided by the controller [34]. Some of the SDN applications are traffic engineering, security, distributed denial of services attack, fault management and so on. SDN has been deployed in many networks like transport network, optical network, wireless sensor network, Internet of things (IoT), edge computing, Wide Area Network, cloud computing and Network Function Virtualization.

1.2.5. Features of SDN Architecture

As SDN networks become increasingly popular, there is a growing need to classify network traffic to optimize network performance, detect and prevent security threats, and manage network resources efficiently. The following section deals some of the features that make SDN suitable for modern network infrastructure:

Adaptive Network Configurations architecture revolutionizes traditional networking by providing an unprecedented level of flexibility. This innovative approach allows network administrators to dynamically adjust and configure network settings in real-time to meet the ever-evolving demands of the digital landscape. With SDN, network resources can be allocated, modified, or decommissioned with ease, ensuring that the network remains responsive and efficient.

Effortless Network Expansion: The highly scalable nature of SDN architecture means that networks can grow or shrink in size without the complexities traditionally associated with such changes. Network administrators can effortlessly integrate additional devices or retire obsolete ones, maintaining optimal network performance without service disruptions. This scalability is crucial for organizations that experience fluctuating network loads or rapid growth.

Robust Network Security: SDN architecture enhances the security posture of the network infrastructure. By centralizing network control, administrators gain a holistic view of the network, enabling them to identify and mitigate security threats swiftly [35].The programmable nature of SDN facilitates the implementation of advanced security protocols and real-time updates to security policies, ensuring that the network remains resilient against both known and emerging threats.

Programmable Networking and Performance Optimization: At its core, SDN architecture offers a programmable framework that empowers network administrators to tailor the network to the organization's specific needs. This level of customization allows for the optimization of network performance, ensuring that data flows efficiently and reliably. Moreover, the ability to automate network management tasks reduces the potential for human error and frees up valuable time for IT staff to focus on strategic initiatives.

Traditional methods of traffic classification, such as signature-based detection and port-based filtering, are becoming outdated and inadequate due to the exponential growth of network traffic and the increasing sophistication of network attacks. ML usually struggles with big and sophisticated datasets, such as image datasets with thousands or even millions of instances [9]. For the classical statistical analysis, such as Bayesian analysis, it is almost impossible to handle such big datasets

In SDN, the centralized control and programmability of the network provide an opportunity to implement new techniques for efficient and accurate network traffic classification.

However, the performance of existing methods in SDN is limited due to the high volume of data and the dynamic nature of the network. Therefore, there is a need to explore the suitability of DL algorithms in network traffic classification of SDN systems.

DL uses neural networks to learn patterns in data, has shown promising results in network traffic classification for traditional networks. However, there is a need to explore the effectiveness of DL in traffic classification of SDN networks, which offer greater flexibility and programmability. The main problem to be addressed in this research is how to develop an effective DL-based traffic classification model for SDN networks that can accurately identify different types of traffic flows, including those that are encrypted and those that use non-standard protocols.

1.3. Motivation

The motivation lies in the potential to enhance network security, improve network management, optimize traffic engineering, enable anomaly detection, and achieve automation and real-time decision making in SDN networks. The motivation to design a DL model for network traffic classification of SDN network for a multi-label network traffic dataset is driven by several reasons:

Improved Network Security: Network traffic classification can help in identifying and mitigating potential security threats such as DDoS attacks, malwareearn normal network traffic behavior and detect anomalies or unusual patterns, unauthorized access attempts. By accurately classifying network traffic, the model can enable proactive security measures and timely response to network anomalies.

Efficient Network Management: SDN networks require efficient management and resource allocation. By classifying network traffic, the model can assist in optimizing network resources, traffic prioritization, and QoS enforcement. This can lead to improved network performance, reduced latency, and better user experience [36].

Traffic Engineering and Optimization: DL models can provide insights into network traffic patterns, allowing network administrators to make informed decisions about network design, capacity planning, and traffic engineering [37]. This can result in more efficient network architectures and better utilization of network resources.

Scalability and Adaptability: DL models,have shown their effectiveness in image classification tasks. By adapting these models to network traffic classification, it is possible to leverage their scalability and adaptability to handle large-scale network traffic datasets and real-time classification requirements.

Automation and Real-time Decision Making: By automating the network traffic classification process, DL models can relieve the burden on network administrators, allowing them to focus on other critical tasks. Additionally, real-time decision making based on accurate traffic classification can facilitate dynamic network management and response to changing traffic patterns.

1.4. Statement of Problem

The increasing complexity and volume of network traffic pose significant challenges in accurately classifying and analyzing network data for various applications such as network security QoS, and resource management. Traditional methods for network traffic classification often rely on manual feature extraction and rule-based approaches, which are time-consuming, labor-intensive, and limited in their ability to handle dynamic and evolving network environments.

To address these limitations, this research aims to investigate the effectiveness of DL techniques in network traffic classification. DL learning models have shown remarkable success in various domains, including computer vision and NLP. However, their application to network traffic classification is relatively unexplored.

The primary objective of this research is to design and implement a DL model for network traffic classification in an SDN network. The model should be capable of effectively classifying different types of network traffic into different categories, such as web browsing, streaming, file transfer, and VoIP, DDoS attacks, malware, and unauthorized access attempts. The proposed model demonstrates high classification accuracy, scalability, and adaptability to handle the dynamic nature of SDN networks. Additionally, the model is expected be adapted and generalize well to different network scenarios, varying network conditions, and evolving applications and protocols.

To achieve these objectives, this research will address the following key research questions:

1. What are the optimal hyper-parameters and settings for the DL models to achieve the best classification performance under different network scenarios?
2. What features of network traffic data affect the classification performance of DL models for network traffic classification?
3. Which type network traffic categories and feature are most frequent and dominant based on the prediction of the DL model proposed for network traffic classification?
4. How do different DL models perform in terms of classification accuracy, scalability, and adaptability for network traffic classification under different network scenarios?

By addressing these research questions, this study aims to contribute to the advancement of network traffic classification techniques, providing valuable insights and practical solutions for efficient and accurate network management, security, and resource allocation in complex and dynamic network environments.

1.5. Objectives of the Research

1.5.1. General Objectives

The objective of this study is to analyze the effectiveness of DL algorithms in the classification of network traffic in SDN systems.

1.5.2. Specific Objectives

The research will specifically focus on network traffic classification to help deal with network operations such as security monitoring, traffic engineering, fault detection, network resource utilization, billing and QoS implementation and identification of parameters of the various network services for network traffic behavior prediction.

To mention some of specific goals that will be achieved by the proposed method:

1. To design and implement a DL model for multi-label network traffic classification that achieves high classification accuracy.
2. To investigate the effectiveness of different ML and DL models for network traffic classification under different network scenarios and emerging applications and protocols.
3. To explore characteristics of network traffic features that affect the performance of the proposed model and identify the most frequent traffic classes in the network.

4. To evaluate the effectiveness of different performance metrics, such as accuracy, precision, recall, and F1-score, for evaluating the classification performance of DL models for network traffic classification.

1.6. Scope and Limitation of the Study

1.6.1. Scope of the Study

The scope of this study includes the investigation of various DL algorithms and their applicability in network traffic classification of SDN. The study will focus on analyzing the accuracy, efficiency, and scalability of the algorithms. The research will also examine the impact of DL models, by leveraging benefits from their knowledge, feature extraction capabilities, and data augmentation techniques to improve the performance and efficiency of DL models for network traffic classification in SDN networks.

1.6.2. Limitation of the Study

The main drawback and challenges that are identified and have been hindering adoption of ML and DL for networking are described as follows:

Availability and quality of labeled data: DL models often require a large amount of accurately labeled data for training. Obtaining such labeled data for network traffic classification is challenging, as it may require significant resources and expertise. The availability and quality of labeled data may limit the generalizability and performance of the proposed DL model.

Interoperability and explainability: DL models are often considered black boxes, making it difficult to interpret and explain the reasoning behind their predictions. This limitation can hinder the understanding of how the model classifies network traffic and may raise concerns in certain applications where interoperability is crucial, such as network security.

Lack of standardized and representative datasets: Different from other DL/ML related domains, where well established, publicly available datasets are available for testing, evaluation and bench marking purposes (e.g., ImageNet in image processing), it is very difficult to find appropriate public datasets to assess ML/DL [38]. While one of the main reasons for this lack clearly arises from the data's sensitive nature, including end-user

privacy, other limitations come from the efforts required to build proper and representative datasets in networking.

Ethical considerations: The use of DL models for network traffic classification raises ethical concerns related to privacy and data protection [39]. The study should address these concerns and ensure compliance with relevant regulations and ethical guidelines, particularly when dealing with sensitive or personal data.

Data dynamics: This brings us to an even more critical issue in networking; networking data is non-static, and generally comes in the form of data streams, which by nature are difficult to analyze.

1.7. Significance of the Study

In today's world, networks play a crucial role in almost every aspect of our lives from communication to finance, transportation to healthcare, networks are the backbone of modern society. However, with the rise of the IoT and the proliferation of connected devices, networks are becoming increasingly complex [40]. This complexity has made it difficult to manage network traffic, leading to a need for more advanced techniques for network traffic classification.

Improved accuracy: By using DL techniques, it is possible to achieve higher accuracy in network traffic classification compared to traditional methods. This will help network administrators, network operators, service providers and network device vendors better understand and control their networks, identify security threats more quickly, and optimize network performance. SDN offers a scalable architecture that separates control and data planes, which can simplify the deployment and management of network devices. DL algorithms is trained on large datasets and can handle large volumes of data with high speed and accuracy. This means that the use of DL in SDN can enable efficient processing of network traffic in large-scale networks.

Flexibility: The use of DL models in SDN can provide a more flexible approach to network traffic classification. Traditional rule-based methods have been unable to capture the complexity and variability of network traffic. With DL, it is possible to build models that are capable of learning from raw data and adapt to changing network conditions.

Security: Ensuring the security of network traffic is a critical challenge in SDN. DL models is trained to detect anomalies in network traffic patterns and identify potential security threats [41].

Innovation: DL is an emerging field of AI that has the potential to revolutionize many areas of technology. By applying DL techniques to network traffic classification in SDN, researchers and engineers can explore new approaches and solutions to network management and security. This could lead to the development of new tools, applications, and technologies that can further enhance the capabilities of SDN.

In summary, DL offers significant advantages over traditional ML in network traffic classification of SDN due to its ability to automatically extract relevant features, handle large and complex datasets, learn complex patterns, and continually improve accuracy over time.

1.8. Contributions

As the use of SDN continues to grow, the need for efficient and accurate network traffic classification becomes increasingly important. Traditional methods of network traffic classification, such as port-based classification, are no longer sufficient in the dynamic and complex environments of SDN networks. Therefore, the use of DL techniques for network traffic classification has become an area of active research. Here are some of research contributions that will be achieved in this area:

1. Development of DL architectures for network traffic classification: One of our main contributions in this area of research is the development of DL architectures that are specifically designed for network traffic classification. While DL has shown promise in this area, existing architectures may not be optimized for the unique features of network traffic.
2. Investigation of the impact of different network parameters on classification accuracy: Another contribution is the investigation of the impact of different network parameters on the accuracy of network traffic classification. Network traffic parameters affecting the performance of DL algorithms are identified.
3. Improve the efficiency and accuracy of network traffic classification task, making it possible to manage SDN networks more effectively.

CHAPTER 2

2. LITERATURE REVIEW

2.1. Applications of Network Traffic Classification Techniques in Networking

The rapid growth of network technology has led to an exponential increase in the volume and diversity of traffic data. Efficiently identifying and classifying traffic flows within a network is crucial for network security, management, and resource optimization. It serves as the foundation for dynamic access control, content-based billing, anomaly detection, and malware identification.

Traditionally, network traffic classification relied on methods such as port-based analysis and deep packet inspection. However, these approaches are becoming less effective due to the widespread adoption of full traffic encryption. In response, ML techniques have emerged as the prevailing approach for traffic classification. Researchers have focused on efficient feature extraction and optimal classification networks, yielding promising results.

Despite these advancements, two significant challenges persist. First, network traffic exhibits an inherent imbalance in distribution, with varying amounts of traffic generated by different protocols and applications. Second, most ML algorithms prioritize overall accuracy, often neglecting minority classes. To address these issues, recent research explores integrating DL with SDN and leveraging transfer learning techniques.

In this literature review, the intersection of networking, ML, DL, and transfer learning, emphasizing their practical applications in traffic classification are explored. By understanding these techniques, we can enhance network performance, QoS, and security.

2.1.1. Port-based Network Traffic Classification

These techniques simply associate services/applications to registered port numbers, e.g., HTTP port, and categorize the traffic according to the used port number. Port-based techniques are among the earliest traffic classification methods [42]. Despite the advantages of port-based techniques such as simplicity on implementation, deploying new communication methods such as tunneling and random ports assignments techniques cause serious difficulties and affect the performance and applicability of them.

2.1.2. Payload-based Network Traffic Classification

Payload-based methods, also known DPI, closely investigate the content of the captured packet, especially the application layer-related information, in order to associate the packet to a specific service/application [8]. To make a prediction, this methodology usually leverages predefined signatures or patterns for each communication protocol, and then discover these patterns to differentiate the traffic flows from each other.

2.1.3. ML And DL Based Network Traffic Classification

ML and DL have the potential to significantly improve network performance optimization by providing more accurate traffic management, predictive analytics, dynamic adaptation, network security, and application optimization. However, it is important to note that DL algorithms require large amounts of data to learn effectively and must be carefully monitored to avoid false positives and other errors.

2.1.3.1. Network and cyber security

ML algorithms are applied for intrusion detection, optimized spectrum utilization, obtaining power efficiency and network traffic management. DL is a subset of ML that uses neural networks to learn and make predictions based on large amounts of data. It has been used in various fields such as image recognition, speech recognition, NLP, and more. In recent years, DL has also been applied to network and cybersecurity to improve threat detection and response [43].

Intrusion detection: DL algorithms is used to identify patterns of abnormal behavior in network traffic that could indicate a potential security threat. By analyzing large amounts of data, DL algorithms can learn to differentiate between normal and abnormal network activity and flag any suspicious activity for further investigation [44].

Malware detection: DL algorithms trained to recognize patterns of malicious code in software and identify potential malware threats. This help detect and prevent malware infections before they can cause damage to a system or network [45].

Threat intelligence: DL algorithms to analyze large amounts of data from various sources, such as social media, news articles, and security reports, to identify potential threats to a network or system. This can help security teams stay ahead of emerging threats and take proactive measures to prevent attacks [46].

User behavior analytics: DL algorithms is able to monitor user activity on a network and

identify patterns of behavior that could indicate a potential security threat. By analyzing user activity logs and network traffic, DL algorithms can learn to differentiate between normal and abnormal behavior and flag any suspicious activity for further investigation. This indicates DL has the potential to significantly improve network and cyber security by providing more accurate threat detection and faster response times [47]. However, it is important to note that DL algorithms require large amounts of data to learn effectively and must be carefully monitored to avoid false positives and other errors.

2.1.3.2. ML and DL for QoS

QoS is a technique used to manage network resources and ensure that critical applications receive adequate bandwidth and priority over less critical applications. DL is applied to QoS to improve network traffic management and ensure that network resources are allocated efficiently. DL algorithms is used to classify network traffic based on the type of application or service that it represents. By analyzing traffic patterns and characteristics, DL algorithms can learn to differentiate between different types of traffic and assign priority levels based on their importance [49]. In addition, DL algorithms is used to allocate bandwidth to different types of traffic based on their priority levels. By analyzing network traffic patterns and resource utilization, DL algorithms can learn to allocate bandwidth dynamically to ensure that critical applications receive adequate resources.

Predictive analytic: DL algorithms is useful to predict network traffic patterns and resource utilization in advance. By using historical data and trends, DL algorithms can learn to predict future network traffic and allocate resources accordingly.

Dynamic adaptation: DL algorithms is used to adapt to changing network conditions and adjust QoS settings in real-time [50] [51]. By analyzing network traffic patterns and resource utilization, DL algorithms can learn to adjust QoS settings dynamically to ensure that critical applications receive adequate resources [51]. Therefore, DL has the potential to significantly improve QoS by providing more accurate traffic classification, dynamic resource allocation, and predictive analytic.

Network traffic management: DL algorithms can analyze network traffic patterns to identify bottlenecks and congestion points in the network. By analyzing packet loss, latency, and other performance metrics, DL algorithms can learn to optimize network settings to reduce congestion and improve overall network performance [52].

Application optimization: DL algorithms is used to optimize network settings for specific applications. Network traffic patterns can be analyzed and resource utilization for specific applications by DL algorithms to allocate resources and optimize network settings to improve application performance [53].

2.1.3.3. Traffic engineering

Traffic engineering is the process of managing network traffic to ensure that it flows efficiently and meets the needs of users.

DL is also crucial for traffic engineering to improve network performance, reduce congestion, and ensure that critical applications receive the necessary resources [54].

Resource allocation: DL algorithms is used to allocate resources dynamically based on traffic patterns and priority levels. DL algorithms are capable of learning traffic patterns to allocate resources to ensure that critical applications receive the necessary resources [55].

Load balancing: DL algorithms is used to balance network traffic across multiple paths to reduce congestion and improve performance by balancing traffic across different paths to ensure that resources are used efficiently.

2.1.3.4. Application identification

Neural network model: DL techniques is used to create a neural network model that can identify certain characteristics of an application. This model is trained on a large dataset of applications to recognize different signatures of different applications. For example, some applications may have a particular type of code, whereas other applications may have specific data flow patterns [56].

Unsupervised learning: DL techniques can also be used for unsupervised learning. This technique involves feeding in a large volume of raw data and allowing the system to learn and recognize patterns on its own. In the case of application identification, an unsupervised DL model can analyze the raw data to recognize patterns that are unique to different applications and identify them accordingly [57].

NLP: DL is also useful in NLP to analyze the text from the application's descriptions, reviews, and feedback in the app store or website. Neural networks can analyze various linguistic features allowing the algorithm to discover the tone of the text. Applying NLP to application identification can estimate the sentiment and intent of the application's users to assist in identification [54].

2.1.3.5. Fault identification

DL models are particularly good at identifying patterns and trends in large datasets, and this makes them ideal for fault identification. The first step in using DL for fault identification is to collect large amounts of data about the system being studied. This might include data from sensors, historical maintenance records, and other sources. The data should be well-organized and stored in a way that makes it easy to access and analyze. Next, a DL model needs to be trained on the data. This involves feeding the model a subset of the data and allowing it to learn the patterns and relationships that exist within that data. The model can then be tested on a separate subset of the data to see how well it can identify faults or abnormalities. Once the DL model has been trained and tested, it is used for fault identification [58]. It is trained on different types of data and is updated as new information becomes available. This makes it well-suited to complex systems that may have many different types of faults and failure modes.

2.1.3.6. User identification

DL involves training artificial neural networks to perform complex tasks such as image recognition, NLP, and user identification.

Facial recognition: Facial recognition is one of the most popular applications of DL for user identification. DL algorithms is trained to detect and identify key facial features such as eyes, nose, mouth, and facial structure. Once the algorithm has identified these features, it can quickly match them with a known database of faces to confirm the user's identity. This technology is widely used in security systems, social media platforms, and mobile devices to provide secure and efficient identity verification.

Voice recognition: Voice recognition is another way DL algorithms is used for user identification. By analyzing speech patterns, tone, and other unique voice characteristics, DL algorithms can quickly verify a user's identity. This technology is especially useful in mobile banking applications where users can use their voice to authenticate transactions.

Biometric authentication: DL algorithms is trained to recognize a wide range of biometric data such as fingerprints, iris scans, and palm prints. These unique identifiers are used to quickly authenticate users for various applications such as banking, e-commerce, and healthcare. This technology is widely used in high-security environments such as government agencies and military organizations.

Behavioral analysis: DL algorithms is trained to analyze a user's behavior patterns to identify unusual activity. This technology is widely used in fraud detection systems where abnormal behavioral patterns such as unusual purchasing activity, location, and time of access can indicate fraudulent activity. By analyzing large amounts of data, DL algorithms can quickly identify these behavioral patterns and alert security or fraud detection teams.

Network traffic analysis: DL models is trained on large datasets of network traffic from different operating systems to identify unique patterns associated with each OS. These patterns could include features such as packet sizes, network protocols, or timing characteristics. The model can then use these patterns to predict the OS of an unknown device based on its network traffic [59].

Host-based analysis: DL is used to analyze host-based data, such as system call traces or file system metadata, to identify unique patterns associated with different operating systems [58]. These patterns could include details such as the types of system calls made, the file paths accessed, or the user accounts used to run processes. The model could then use these patterns to predict the OS of an unknown device based on its host-based data.

2.2. Transfer Learning for Traffic Classification

Researches of traffic classification related to transfer learning are mainly focused on ML based traffic classification. The experimental results of the Moore dataset demonstrate that it can improve the accuracy and precision, and reduce the computing consumption. To solve the point-to-point traffic recognition under variable network environment and overcome the problem of inconsistent data distribution between training samples and predicting samples. An instance-based transfer learning strategy to find a part of the relevant data effective for training the target domain task by analyzing the source domain dataset.

Although those studies are all transfer methods based on ML, they have proved the potential of transfer learning in the field of traffic classification. In reality, large amounts of data are difficult to obtain for long-term storage due to the scarcity of storage resources, which means that packet-level transfer learning methods are difficult to apply into most scenarios.

Facing the problems of scarce labeled data and insufficient computation power is used for many unlabeled data to train the model by semi-supervised learning to extract the statistical

features of the time series, and transferred the trained parameters to the target domain neural network, and then adopted a small number of labeled data to train the new network to obtain the final classification model. Their experimental results show that this method can achieve a similar effect to large-scale labeled data training on a specific dataset. However, they also stated that the research essentially relies on the time-series feature extraction classification method, and doesn't work well on some datasets that cannot be classified by this feature such as the ISCX dataset [60]. On the other hand, this scheme still essentially relies on temporal features extracted by humans for classification and is effective on some datasets, while it still suffers from the "concept drift". And cannot support the end-to-end classification effect. From the above analysis, transfer learning as an effective method to solve the scarce data problem will play an important role in network traffic classification. For network traffic classification using a multilabel network traffic dataset, several transfer learning models are considered.

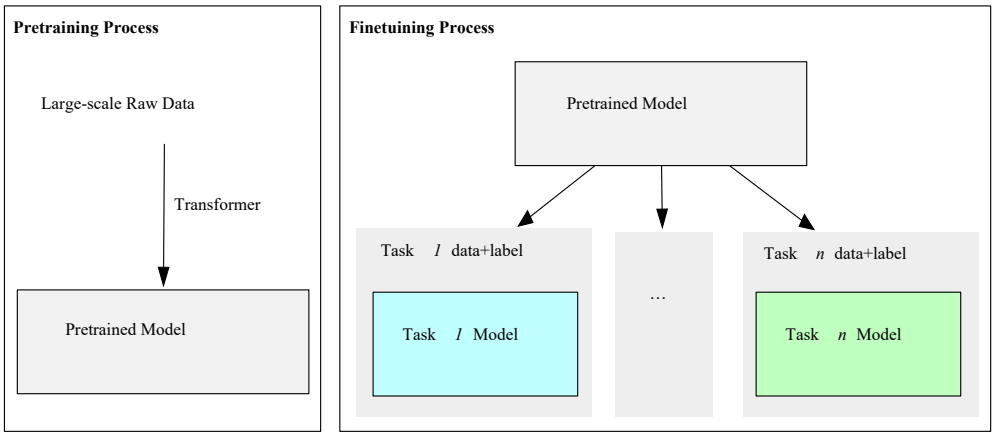

Figure 2: Pretrained model design

2.2.1.1. Deep transfer learning with fine-tuning

This approach involves using a pre-trained neural network and fine-tuning it for a specific dataset. Starting with a pre-trained model that has been trained on a large-scale dataset, remove the last classification layer(s) of the pre-trained model, add new classification layers specific for multilabel traffic classification problem. And finally train the modified model on the dataset while keeping the pre-trained weights frozen for the initial layers. Fine-tune the entire model by updating the weights of all layers using a smaller learning rate. This method leverages the knowledge learned from the pre-trained model and adapts it to specific task.

2.2.1.2. DA-Transfer, Data Adaptation and Model Transfer

DA-Transfer combines data adaptation and model transfer adaptation to improve the performance of small-sample classification models. It's particularly useful when dealing with scarce labeled data. The approach aims to enhance the transferability of features learned from one domain to another.

2.2.1.3. Ensemble of deep learning architectures

This method combines multiple deep learning architectures for traffic classification which trains an ensemble model that combines predictions from different architectures. This approach can improve classification accuracy by leveraging diverse representations.

2.2.1.4. Feature-based deep learning

This approach focuses in extracting relevant features from network traffic data packet-level, payload, inter-arrival time, etc. And train deep learning models on these features.

2.2.2. Applications of Transfer Learning in Other Domain

Transfer learning is now widely used in image recognition, NLP, text mining, speech classification, and other application fields. In terms of image recognition, has given a selective learning algorithm to solve the problem of distance transfer learning, such as using face images to identify aircraft images. A convolutional neural network has been applied for corn disease image recognition model based on transfer learning to realize the recognition of corn disease images under the complex field background of small data samples, which is intelligent recognition with rust images. In NLP and text mining, transfer denoising auto encoder (AE) and a transfer deep network have been proposed, both of which are more accurate in complex text classification, message classification, and news comment classification [61].

2.3. ML and DL Applications in SDN

SDN is an emerging technology that enables network administrators to manage network resources centrally through software, rather than traditional hardware-based methods. SDN is gaining popularity due to its ability to enhance network performance, security, and flexibility. However, managing network traffic in an SDN environment is challenging due to the dynamic nature of the network. By training DL models on network traffic data, administrators can classify traffic in real-time and take appropriate actions to optimize network performance [62]. The use of DL for network traffic classification in SDN networks

has several benefits. Firstly, it can provide administrators with a more granular view of network traffic, allowing them to identify which applications are consuming network resources. This information can then be used to optimize network performance by allocating resources more efficiently. Secondly, DL-based classification can help improve network security by identifying anomalous traffic patterns that may indicate a security breach. Finally, DL-based classification can help reduce the workload of network administrators by automating the classification process. DL has been shown to be more effective than traditional ML in network traffic classification of SDN.

2.3.1. **Deep learning And SDN Integration**

The SDN controller represents the network's brain. It is responsible for routing updates, constructing the forwarding rules, optimize the network performance and classify the network traffic based on the learned traffic information. There are different types of SDN controllers such as POX, NOX, Open Daylight with different specifications and characteristics. We employ ODL controller as it facilitates fast prototyping. The standard Open-flow protocol is used as a southbound API for establishing the communication between the data and control planes, whereas the set of ODL APIs is used on the northbound interface for developing various network control applications.

Open-flow is one of the most dominated SDN communication protocols. By using the software programming, Open-flow supports the control and communication of the rules signaling and flows forwarding from the centralized controller southbound to the SDN switch. The northbound APIs from the control plan to the management plan, on the other hand, is defined to be used for application service and operation management. Several consoles, such as ODL, Floodlight, and Open Day Light, have been developed to support Open-flow. We apply the Open-daylight controller which supports Open-flow to define the framework of the SDN networks. Open-daylight, being a component-based SDN framework, provides software components with well-defined API, which is easy for developers to create new control applications and network management. It also supports various communication protocols for network devices management, such as Open-flow and OF-config. Open-flow is one of the most dominant communication protocols in SDN. By using the software programming, Open-flow allows for the control and deployment of network rules and policies from a centralized controller to an SDN switch. As for the northbound APIs, Open-flow supports open APIs from both the controller and management

applications to be used as communication transports from applications to the underlying infrastructure. Open-flow has become a popular communication protocol in the world of software-defined networking. The protocol supports the control and communication of rules that are sent from the central controller to the SDN switch, as well as the flow forwarding controls. The southbound API on the software stack, on the other hand, is used by applications to manage network operations and services.

The DL model is deployed at various points in an SDN network depending on the requirements and objectives. Some potential locations to deploy the model include:

2.3.2. Edge of the Network

This would involve deploying the model on the switches closest to the end-users, allowing traffic to be classified in real-time as it enters the network. This approach would be particularly useful for traffic management and QoS applications.

2.3.3. Data Center

Deploying the model in the data center would allow the classification of traffic between different applications, identifying which services are consuming the most resources. This could be used to optimize the allocation of resources within the network and improve overall network performance.

2.3.4. Cloud

For cloud-based services, deploying the model within the cloud would allow traffic to be classified and monitored in real-time, providing insights into user behavior and network performance. This would be particularly useful for cloud-based applications that rely on low-latency connectivity. Ultimately, the choice of deployment location will depend on the specific requirements and goals of the SDN network.

In this section the most recent and significant studies in the field of DL in network traffic classification of SDN for various applications are discussed.

2.3.5. Deep learning in SDN

Software-defined networking has several main features, including the global network view, separation of the data plane from the control plane, and network programmability. These

features provide a new opportunity for DL algorithms to be applied to network traffic management.

In this section, different application cases in the SDN environment have been presented that use DL algorithms in detail. The existing application cases is divided into three categories based on different application scenarios: traffic classification, traffic prediction, and IDS. Traffic classification refers to the process of identifying the type of traffic that is flowing through the network. Traffic prediction involves forecasting the future traffic patterns based on historical data. IDS are used to detect malicious traffic that may be attempting to compromise the network [63]. DL algorithms can be used to improve the accuracy and efficiency of these tasks by automating the classification process and reducing the workload of network administrators. By leveraging the granular view of network traffic provided by DL-based classification, administrators can optimize network performance by allocating resources more efficiently and improve network security by identifying anomalous traffic patterns that may indicate a security breach.

2.3.6. Traffic Classification

The number of smart devices connected to the internet has increased rapidly, creating more network traffic and new kinds of data [51]. To optimize the internet access and user experience, it is important to classify the network traffic and understand what kind of data it is. For example, prioritize some data over others, depending on how important or urgent they are. We can also allocate the bandwidth more efficiently and avoid wasting resources. Traffic classification is the process of identifying the source or type of the network data. It can tell us the name of the application (such as YouTube, Netflix, Twitter, etc.) or the category of the data (such as streaming, web browsing, etc.). Traffic classification is useful for many network-related tasks, such as security monitoring (e.g., detecting malicious attacks), QoS provisioning, and long-term planning and management of network resources. For instance, knowing the application name can help us to manage the bandwidth resources better. Moreover, network operators need to know what kind of data is flowing over their networks quickly so they can respond to their business goals effectively.

As DL has been emerged as a promising approach for network traffic classification in SDN. SDN is an architecture that allows network administrators to centrally manage network traffic flows by separating the data and control planes of the network.

Raikar et al. [32] proposed a framework for the classification of applications in traffic classes using three machine learning models: Support Vector Machine (SVM), Nearest Centroid (NC), and Naive Bayes (NB). The network traffic datasets were captured and flows features are generated, which is sent to the classifier for prediction. The accuracy obtained for SVM is 92.3%, NB is 96.79% and the nearest centroid is 91.02%. The challenges faced are in the live network data traffic capture and classification of the applications in the SDN platform.

Ghosh et al. [42] a hybrid approach that combined DL with traditional classification techniques for SDN-based network traffic classification has been proposed. The authors used a decision tree algorithm to preselect relevant features from network traffic data before feeding the data into a deep CNN for classification.

Zhang et al. [51] a DNN-based traffic classification method composed of SAE and a Softmax regression layer to identify applications in classes. The proposed DL-based system leverages a VAE to extract high-level representations of network traffic data before applying a hierarchical classification approach and achieved better classification accuracy than DL-based methods. The authors evaluated their proposed method on a public dataset and reported an accuracy of 90% or above in identifying previously trained in-app activities and an average accuracy of 79% in identifying previously untrained in-app activity traffic as unknown data when this framework is employed.

Malik et al. [64] proposed a new deep learning model for SDN that accurately identify a wide range of traffic applications in a short time, called Deep-SDN. The proposed model is based on a deep neural network architecture that uses a combination of convolutional and recurrent neural networks to extract features from the network traffic data. The proposed model was evaluated using a real-world dataset, and the results showed that it outperformed several state-of-the-art traffic classification methods in terms of accuracy, precision, recall, and f-measure. The authors have reported an overall accuracy of 96% for their proposed model.

Fan and Liu [65] have conducted a study on machine learning-based traffic classification techniques and how the models' fit and feature selection affect their performance. The authors have proposed two machine learning algorithms, namely supervised Support Vector Machine (SVM) and unsupervised K-means clustering, for traffic classification. The study has found that an overall accuracy of over 95% can be achieved using these algorithms. The authors have also investigated the impact of model tuning and feature

selection on the system performance. They have found that the system performance can be further improved with model tuning and feature selection.

Chen et al.[66] proposed a novel deep neural network that combines both the convolutional network and the recurrent network to improve the accuracy of the classification results. The proposed model extracts packet features for a single packet using a convolutional network and picks out the flow features based on the inputs of the packet features of any three consecutive packets in a flow using a recurrent network. This model provides more flexibility in real practice and surpasses the existing studies which ask for the first packets of a flow.

Deep Packet-Based Traffic Classification in SDN with Attention Mechanism by Junliang Chen, Jian Yang, and Shijie Zhou proposes a deep packet-based traffic classification method for SDN that uses an attention mechanism to focus on important features in network traffic data [67]. The proposed method achieved high accuracy in the classification of network traffic with a low computational overhead.

2.3.7. Traffic Prediction

The objective of network traffic prediction is to forecast the amount of traffic expected based on historical or real-time data to avoid future congestion and maintain high network quality while decreasing communication latency. Predicting network congestion is essential to provide and maintain high-quality network communication. Based on the outcome of the analyzed traffic data, the SDN controller can direct the flows to the less congested links, which helps to optimize network performance and reduce communication latency [68].

Wang et al. [69] proposed a framework for traffic classification into QoS classes using semi-supervised machine learning in SDN. The proposed model focuses on classifying multimedia traffic based on QoS characteristics from a QoS class perspective. The model has been trained on datasets captured popular Internet multimedia traffic from a campus network. The datasets include 26 natural categories of multimedia traffic, which are divided into seven QoS classes. The paper a modified K-Singular Value Decomposition (K-SVD) method for multimedia identification.

2.3.8. Network Security

Despite the many advantages of SDN, its security remains a challenge [63] . IDS are one of the most important network security tools due to their potential in detecting novel attacks. Intrusions are defined as "attempts to compromise the confidentiality, integrity, or availability of a computer or network, or to bypass the security mechanisms of a computer or network." IDS can help detect such attempts and alert network administrators to take appropriate actions to prevent or mitigate the damage caused by the attacks.

Guoyan Tian and Hongtao Du.a DL approaches for network traffic anomaly detection in SDN. This paper presents a DL-based approach for network traffic anomaly detection in an SDN environment. The proposed method uses DBN to capture the complex relationships between network traffic features and detect anomalies with high accuracy [70].

In summary, the literature review shows that DL techniques hold great potential for network traffic classification in software-defined networking environments. These approaches capable to help reduce computational overhead, increase accuracy, and enable real-time application-aware traffic management to provide better network control. However, the proposed techniques still require more research and testing to overcome challenges such as data scarcity, model complexity, and network heterogeneity.

Table 1: Literature reviews on DL in SDN network traffic classification

Ref.	Area of the study	Model used	Number of features and approaches used.	Model Output	Accuracy	Limitations
[32]	Framework for the classification applications	SVM, NB	14 flow features	3 classes	NB: 96.79% SVM: 92.3% NC: 91.02%	• No information provided on the size of the dataset used
[42]	DL with traditional classification techniques	Deep CNN for classification	DT preselect relevant features	UNSW-NB15 dataset & 9 network attacks classes		• Limited dataset and attack types,
[51]	DNN-based traffic classification	SAE	Automatic by algorithm	10 classes	SAE: Training accuracy: 90% Testing: 79%	• High computational cost, is not be suitable for real-time

	composed of SAE					traffic classification tasks
[64]	Deep-SDN	DNN	Automatic by algorithm	10 classes	96.00%	• Limited number of features
[65]	ML-based traffic classification techniques	SVM and K-Means	30 flow features	8 classes	SVM: 98.7% K-M: 88.0%	• The study considers limited number of features •
[66]	Combined novel deep neural network	CNN,RNN	Automatic by algorithm	6 applications	99.0%	• Limited number of applications and protocols.
[69]	Aframework for traffic classification into QoS classes	Semi-supervised machine learning	26 features	47classes	90.0%	• Limited number of features which to cover real-world network traffic.
[71]	DL for Network Traffic Classification with SDN in Cloud	CNN	Not mentioned	5 classes	97.6%	• Features which don't cover real-world network traffic
[70]	DL approaches for anomaly detection in SDN	DBN	Not mentioned	Captures network traffic features and detect anomalies		Don't represent the diversity and complexity of real-world network traffic and cyber attacks
The proposed model	DL approaches for network traffic classification of SDN	DNN	RF is used to preselect relevant features	208 Features 24 classes	92%	Lack of enough training data

CHAPTER 3

3. MATERIALS AND METHODS

The architecture for traffic classification has been proposed for SDN architecture which separates the data layer from the control layer. The data plane solely concentrates on network equipment and packet transmission while the control plane determines packet routing and QoS policy decision by an external controller. These two planes can communicate using Open-flow protocols. Moreover, open-source APIs are utilized to reduce dependence on network equipment manufacturers and enable development and usage of various network software.

The adoption of DL into SDN architecture for traffic classification further enhances its efficiency by segregating data and control layers while also enabling communication between them through Open-flow protocols. Additionally, open-source APIs offer flexibility regarding software development and usage while reducing reliance on specific manufacturers.

3.1. Materials

To effectively implement and evaluate a DL-based model for traffic classification in software-defined networks, contributing to advancements in network management, security, and resource allocation in SDN environments it is crucial to select the right resources and materials for the right proposed model including software packages, modules, and hardware.

3.1.1. Network Traffic Dataset

A suitable dataset is required for training and evaluating the DL model. This dataset should consist of network traffic data captured from SDN environment. The dataset should be representative of real-world SDN traffic, covering a diverse range of applications, protocols, and network conditions. It is obtained by deploying packet capturing tools or leveraging existing network monitoring systems. The dataset should contain labeled network traffic data with each class representing a specific type of traffic, such as web traffic, video traffic, voice traffic, etc.

3.1.2. Hardware Resources

The implementation phase may require suitable hardware resources depending on the complexity of the DL model and the size of the dataset. This includes high-performance CPUs, sufficient memory, and storage capacity. The hardware resources should be capable of handling the computational requirements of training and evaluating the DL model effectively. For running the proposed model Satellite S55t-A with Processor Intel(R) Core (TM) i7-4700MQ CPU @ 2.40GHz, 2401 Mhz, 4 Core(s), 8 Logical Processor(s) and Installed Physical Memory (RAM) of 16.0 GB platform, is used.

3.1.3. Deep Learning Framework and Libraries

To train and implement the DL model, popular frameworks has been used such as TensorFlow and Keras. These frameworks make it easy to develop and train the model by providing various DL layers, models, and optimization algorithms. TensorFlow: TensorFlow is a software tool that enables developers to build and train ML models. It was developed by Google and is widely used in AI. TensorFlow lets programmers create programs that can learn from data. TensorFlow is a powerful tool for creating intelligent applications that can learn from data. It is versatile and easy to use, and it is one of the most popular frameworks in ML today [37].

Keras: Keras is a DL API based on Python that works on the TensorFlow ML platform. It is designed to solve complex problems in both DL and ML. Keras is a valuable tool for engineers and researchers, as it gives them access to all the features of TensorFlow. Keras models can also be deployed easily on browsers and mobile devices using TPU or GUP clusters. Besides the DL framework, also various software and libraries for data preprocessing, feature extraction, and analysis is used. This includes Python programming language, and libraries such as NumPy and Pandas for data manipulation, Scikit-learn for feature extraction, and Matplotlib or Seaborn for data visualization.

SciPy: SciPy is a library for scientific computing in Python. It relies heavily on NumPy, which enables efficient manipulation of N-dimensional arrays. SciPy can work with NumPy arrays and can run on any operating system. SciPy is an open-source software that can be easily installed and used.

Table 2: Hardware and software tools

DL Frameworks and tools	Available Interface
TensorFlow	Python
Keras (higher level library for TensorFlow)	Python
Anaconda Navigator	Anaconda 3
Spyder IDE	Python
ML models	RF, SAE, DNN

3.2. Data Collection and Preparation

In this phase a suitable dataset for training and evaluation is collected. This includes selecting appropriate network traffic datasets in the context of SDN. The data is obtained from existing traffic traces, simulators, and SDN testbeds. In addition it is important to make ensure that the CSV dataset contains the network traffic data along with their corresponding labels.The features are represented as columns, and the labels is presented in the separate column.

A reliable open-source dataset is an essential foundation for researching DL algorithms. Using open-source datasets helps to ensure the credibility of research results and facilitates the reproduction and comparison of research work by other researchers.

3.2.1. Collect Network Traffic

It is important to note that capturing network traffic from a network may have legal and ethical considerations, and it is important to obtain permission from the network owner before doing so. Additionally, the captured traffic may contain sensitive information, so it is important to handle it securely and appropriately. To collect and capture traffic data network traffic network monitoring tools and network traffic capture tools like Wireshark are used to capture and record network traffic data. These tools allow to analyze packets flowing through the network and extract relevant features for training the DL model.

The network traffic dataset should include a wide variety of network traffic samples representing different application protocols and activities within a software-defined

network in addition the packets captured from the network traffic, contains information about the different protocols, headers, and payload data.

There are relatively few public labeled datasets in academia, and many studies have been conducted on self-collected private datasets, which limits the comparison of different schemes in some aspects. In the end-to-end network traffic classification study, most researches adopt the dataset provided which contains 10 kinds of normal traffic collected by IXIA BPS equipment including normal applications. And it is important to ensure that the dataset is representative of real-world SDN traffic, covering a diverse range of applications, protocols, and network conditions.

3.2.2. Unicauca-dataset-April-June-2019-Network-flows Dataset

Several datasets suffer from some shortcomings. For example, the KDD-Cup99 dataset contains much redundant and duplicate traffic, while the UNSW-NB15 dataset contains outdated and unreliable traffic. Compared with the first two datasets, the traffic samples in the USTC-TFC dataset do not have too many redundant and duplicate traffic samples. The traffic in this dataset is encrypted using a Transport Layer Security (TLS) protocol, which more closely matches the real network situation. Therefore, the Unicauca-dataset-April-June-2019-Network-flows dataset has been chosen.Unicauca-dataset-April-June-2019-Network-flows is a network traffic dataset that contains network flow data collected from a university campus network in Colombia between April and June 2019 The dataset consists of 2,325,372 network flows, each of which is described by 85 features, including packet size, inter-arrival time, source and destination IP addresses, source and destination ports, and protocol type . The dataset is labeled with 114 different applications, including web browsing, email, file transfer, and video streaming. The dataset is available in CSV and other formats and have been used for network traffic classification, intrusion detection, and other network security applications

3.2.3. Preprocess Network Traffic Data

Convert the packets (in PCAP format) into a suitable format for training.

3.2.3.1. Input Preprocessing

The data type for each input layer is determined based on the data type of the corresponding column in train_features. Numeric inputs are concatenated and normalized using a Normalization layer, for categorical features (non-float data types), a String Lookup layer

is used to map string values to integers, followed by a Category Encoding layer. A functional preprocessing model is created to generate preprocessed inputs. This model can be used to preprocess new data before feeding it into trained model. A visual representation of the train_preproccessing model is generated. A dictionary of features_dict is created to store arrays of feature values. Overall, this pipe line created input features using TensorFlow and Keras which is helpful in preparing the data and the model is used for a machine learning and deep learning tasks, ensuring that the input data is in the correct format for training the neural network.

3.2.4. Feature Engineering and Reshaping

In traditional ML, feature engineering involves an important step to identify relevant features that can describe and differentiate network traffic effectively. These features is statistical, packet-level, or flow-level information. It is crucial to design features that capture key characteristics of the traffic, such as payload, packet size, flow duration, or inter-arrival time. The common techniques that help improve the performance of the proposed model, to avoid overfitting, and improve the generalization of the proposed model, there are several techniques implemented:

Feature Extraction: Extract relevant features from the network traffic data. Common features include packet size, flow duration, and payload characteristics.

3.2.4.1. Feature selection and extraction

This is a method of selecting the most relevant and informative features for the problem and discarding the redundant or noisy features. Which enables us to reduces the dimensionality and complexity of the data and the model. This involves removing any irrelevant or noisy data from the dataset. Clean the dataset to remove any incomplete, irrelevant, or erroneous data. This is done by removing duplicate data, removing outliers, and imputing missing values and packets that do not belong to the network traffic of interest. To select the most relevant features from the dataset,the proposed model used RF. This has been done by selecting features based on network traffic datasets that can represent real-world network traffic and features that affect the efficiency of the model.

3.2.4.2. Feature scaling

This involves scaling the data to a common range to ensure that all features have equal importance and to ensure that all the features have similar scales. This is a common preprocessing step in machine learning as it can help improve the performance and

convergence of many algorithms. However, it's important to note that typically, the mean and standard deviation from the training set should be used to standardize the test and validation sets to ensure consistency across all data transformations, ensuring that the training, testing, and validation data are all on the same scale, which is crucial for many machine learning models to perform well. This process helps avoid bias towards features with larger scales and improves the convergence of algorithms during training.

Normalization: DL algorithm trains and develop a model then expected output compared with actual output in hopping the difference is minimum (minimum loss function). Gradient descent techniques scale the magnitude of parameters. During searching optimum values for hyper parameters small incremental update results in converging on the optimal value on the other hand drastic update results in divergent behavior.

Missing value and duplicate values handling: There are three methods to handle missing values in data science. Ignoring method, imputation method and last one is using modeling method.

Data dimensional reduction: Many features increase computation complexity, increase resource usage, and time consumption is high. To solve the mentioned drawback finding relevant and significant features of large network traffic is needed besides it increases the accuracy of the neural network. It includes the training data to be converted into a 2D array where the first dimension is the number of samples and the second dimension is the number of features. And finally, the dataset will be normalized, scaled and this helps with the convergence of the training process.

One-hot-encoding: One-hot encoded representation converts the class vectors (integers) in training, testing and validation to binary class matrices using the to_categorical function. One-hot encoding is a common preprocessing step for handling categorical data in machine learning. It converts categorical variables into a form that could be provided to ML algorithms to do a better job in prediction. Each unique category value is transformed into a binary column which is either 0 or 1 to denote the presence or absence of the feature. This process is repeated for each categorical feature in the dataset. The result is a data frame ready for use in machine learning models that require numerical input.

Data Splitting: Split the dataset into training, validation, and testing sets. The training set is used to train the DL model, the validation set is used to optimize the model hyper-parameters, and the testing set is used to evaluate the model's performance. Split data into

training and testing sets: The data set needs to be divided into training and testing sets to evaluate the performance of the DL models. Usually, 70-80% of the data is used for training, and the remaining 20-30% is kept for testing the models. Split the preprocessed dataset into training and validation sets using a stratified sampling technique. The training set should be used to train the DL model, while the validation set should be used to evaluate the model's performance.

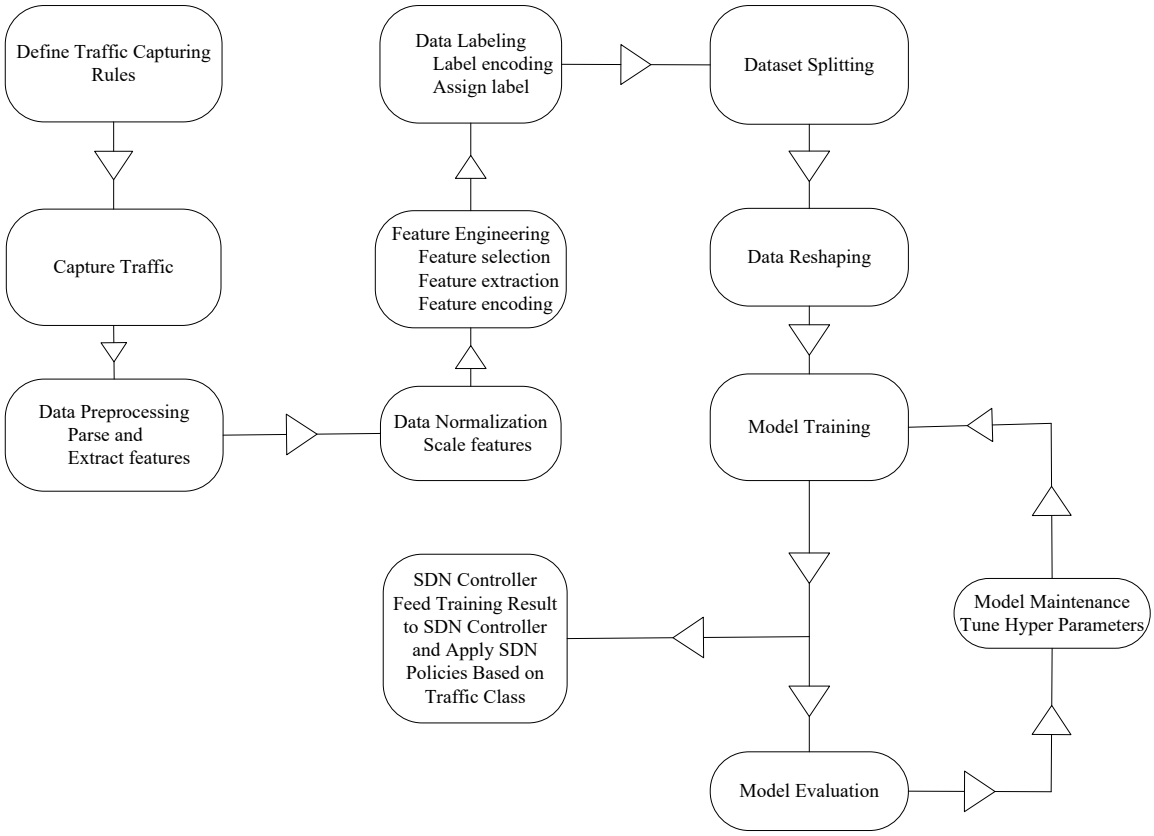

Figure 3: Data processing life cycle in DL-SDN integration

3.3. **Model Development**

The incorporation of AI techniques has led to notable improvements in numerous network traffic tasks, including traffic engineering, attack detection, resource scheduling, protocol analysis, traffic generation, etc. [72]. However, they always customize specific models for specific network tasks, which may suffer limitations from small sample sizes for specific tasks, insufficient model training under small samples, and high costs to develop customized models.

The following figure shows DL models used to provide high-quality network services as well as ensure data privacy, there are plenty of network tasks for diverse network service demands.

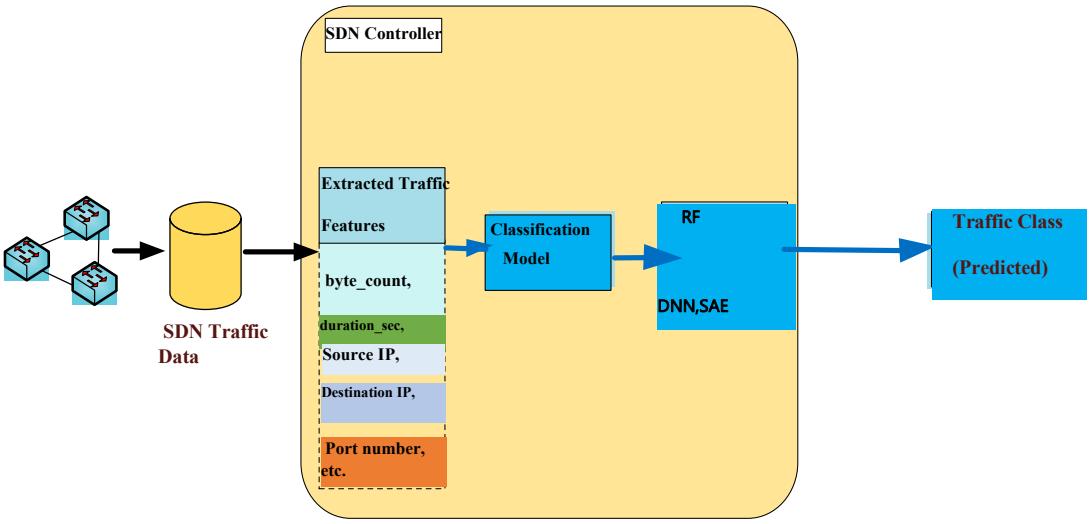

Figure 4: Diverse traffic classification tasks with different implementation

Developing a DNN model for network traffic classification in SDN involves several key steps:

3.3.1. The Proposed Model Architecture

The proposed model has been developed after series tunning hyper parameters, enables the selection of the optimal numbers and types layers, types of activation functions, types of optimizers, batch-size, dropout-rates learning rate etc.

The model is a type of feed forward neural network, where connections between the nodes do not form a cycle. The architecture is a DNN using TensorFlow, a popular machine learning library.

The model consists of the following major DL components:

Input Layer: This is where the model receives the input data. The shape is determined by the number of features in the training dataset.

Hidden Layers: These are intermediate dense layers between the input and output layers. They consist of neurons that apply weights to the inputs and pass them through an activation function to derive some output. The relu activation function is used here, which stands for rectified linear unit. It's a piece wise linear function that will output the input directly if it is positive, otherwise, it will output zero.

Normalization layer: This layer is inserted after each dense layers and normalize features for the next dense layer.

Activation function: After each convolutional layer, a rectified linear unit (ReLU) activation function is applied. ReLU introduces non-linearity to the network and helps in capturing complex patterns.

Dropout: You can use dropout regularization to randomly drop out some of the neurons in the model during training. This prevents over-fitting and improve the generalization of the model.

Output Layer: The final layer that provides the output of the model. The softmax activation function is used for multi-label classification problems. It converts the output to a probability distribution over the target classes.

Activation Function: A neural network consists of multiple neurons that are clustered into several layers, neurons in the previous layers connected to the neurons of the next layer, each connection have weight W, bias b, and activation function corresponding to it. To perform complex tasks neural network needs an activation function. A neural network without activation function is a linear regression model. E.g. ReLU Activation Function

Loss function: Binary cross-entropy is used as loss function and as the optimizer Adam has been used in the proposed model.

Parameters: indicates the number of trainable parameters in each layer, which are the weights and biases that the network will learn during training.

3.3.2. Model Evaluation

Model evaluation refers to the process of assessing how well a model performs in predicting outcomes or making decisions based on given data. It is an important step in ML and other predictive modeling techniques, as it helps determine if the model is accurate and reliable enough for practical use. The performance of the DL model will be evaluated using metrics such as accuracy, precision, recall, and F1 score. The performance of the DL model will be compared with traditional methods. Evaluate the trained model's performance on the validation set using appropriate evaluation metrics such as accuracy, precision, recall, F1-score, and confusion matrix. Analyze the results obtained and observe any trends or patterns. Here are metrics used to evaluate the proposed model

3.3.2.1. Accuracy

Accuracy metrics for DL refer to the measures used to evaluate how well a DL algorithm performs in correctly predicting outcomes. These metrics are important because they allow developers and researchers to assess the effectiveness of their models and make improvements where necessary. This is the most used metric for evaluating classification models. It measures the proportion of correctly classified samples among all samples. However, accuracy alone may not always be a reliable metric, especially in imbalanced datasets where the number of samples in each class is significantly different. And the Mathematical formula to calculate accuracy is stated below:

$$Accuracy = (TP + TN)/(TP + TN + FP + FN),$$

Where:

TP = True Positives

TN = True Negatives

FP = False Positives

FN = False Negatives

3.3.2.2. Precision

Precision metrics for DL refer to the measures used to evaluate the accuracy and reliability of ML models that use complex algorithms to analyze large amounts of data. Precision measures the proportion of true positives (correctly classified traffic belonging to the class) among all positives (both correctly classified and misclassified traffic classified as the given class). These precision metrics are critical in assessing the performance and effectiveness of DL models. Compute Precision using the formula below:

$$Precision = TP/(TP + FP)$$

3.3.2.3. Recall

Refers to the way that measure how well a ML model is able to correctly identify all relevant instances of a particular class or category. In other words, it measures how many true positives (correctly identified instances) there are out of all possible positive cases. This metric is important because it helps us understand whether our model is accurately identifying and categorizing information. Recall measures the proportion of true positives

among all actual positive samples (correctly classified and misclassified traffic belonging to the given class). Recall computed using the mathematical equation below:

$$Recall = TP(TP + FN)$$

3.3.2.4. F1 score

F1 Score is a measure of how well a classification model can correctly identify positive and negative instances. It considers both precision (the proportion of true positives out of all predicted positives) and recall (the proportion of true positives out of all actual positives). The F1 score is the harmonic mean between precision and recall, giving an overall performance metric for the model. Essentially, it tells us how accurate our model is at identifying relevant information while minimizing false results. F1 score is the harmonic average of precision and recall, combining both metrics into a single score. It ranges from 0 to 1, where a score of 1 indicates perfect precision and recall. F1-Score is given below:

$$F1\_Score = (2 * Precision * Recall)/(Precision + Recall)$$

3.3.2.5. ROC curve

Receivers Operating Characteristics is a visualization tool for binary classification tasks. It plots the true positive rate against the false positive rate as the classification threshold varies. The area under ROC curve is used as a metric for the model's ability to distinguish between classes.

3.3.2.6. Confusion Matrix

A confusion matrix is a tool used in ML to evaluate the accuracy of a prediction model. It is essentially a table that compares the predicted outcomes of a model with the actual outcomes. The matrix displays four different values: true positives, false positives, true negatives, and false negatives. True positives refer to cases where the model correctly predicts positive outcomes. False positives occur when the model incorrectly predicts positive outcomes. True negatives are cases where the model correctly predicts negative outcomes, while false negatives occur when it incorrectly predicts negative outcomes. By analyzing these values in relation to each other, researchers can determine how well their prediction models are performing and identify areas for improvement. For example, if there are many false positives or false negatives in the results, adjustments may need to be made to improve accuracy. It is an effective way to visualize model performance and identify where the model is making mistakes.

3.3.2.7. Area under curve

AUC is a metric used to evaluate the performance of a classification model. It represents the probability that a randomly chosen positive instance is ranked higher than a randomly chosen negative instance. It provides an aggregate measure of performance across all possible classification thresholds.

Calculate AUC for Each Class: In a multilabel setting, calculate the AUC for each class separately, treating it as a binary classification problem where one class is positive and all others are negative.

Aggregate AUC Scores: Aggregate the individual AUC scores using one of the following methods:

Micro-average: Calculate the AUC based on the aggregation of all individual true positives, false positives, false negatives, and true negatives.

Macro-average: Calculate the AUC for each class and then take the average.

Weighted-average: the proposed model has used these approaches and calculate the AUC for each class, weighted by the number of true instances for each class.

Interpret the Results so that a higher AUC value indicates better model performance. An AUC of 0.5 suggests no discrimination (equivalent to random guessing), while an AUC of 1.0 indicates perfect classification.

3.3.2.8. Probability Distribution of Prediction

It's a tool that helps to balance the trade-off between precision and recall, ensuring that the model's predictions are as accurate and useful as possible. The probability distribution is crucial for understanding the model's confidence in its predictions and for making decisions based on the model's output. It's especially important in multi-label classification, where the complexity of the data can lead to a single input being associated with multiple classes.

3.3.3. Resource Utilization

To measure the resource consumption of the model, a profiler tool has been used that can track the time spent on each operation and layer of the model,CPU and memory consumption from the start to end time of each training step and calculate the total consumption.

3.3.3.1. Training time of model

Training time is a period that DL spent to learn dataset and identify network traffic from normal network flow. The training time of DL models depends on various factors such as the size of the dataset, the complexity of the model, and the hardware you are using for training. Therefore, training step time is an important factor to consider when designing and evaluating machine learning models, as it affects the speed and efficiency of the training process. A shorter training step time means that the model can learn faster and use less computational resources. However, a shorter training step time may also imply a smaller batch size, which may affect the stability and accuracy of the gradient update. Therefore, there is a trade-off between the training step time and the quality of the model. To reduce the training time the proposed model, the following strategies have been applied.

Increased the batch size: multiple batch size have been implemented to process more examples in parallel and reduce the number of gradient updates which speeds up the training process and reduce the communication overhead. .

Used early stopping: early stopping has been applied to monitor the validation loss and stop the training process when the loss stops decreasing or starts increasing. This also prevent overfitting and save unnecessary training time.

3.3.3.2. CPU utilization

To measure the CPU and memory utilization of the model, the resources that the proposed model consumes is monitored during the training or inference process. A profiler tool has been used, which is suitable to measure the CPU and memory usage of a model in a more detailed and accurate way.

3.3.3.3. Classification report

A classification report summarizes the model's performance on each class, including precision, recall, F1 score, and support. It helps in identifying which classes the model is performing well on and where it is struggling.

3.3.4. DL SDN Integration

SDN provides a programmable network infrastructure, allowing us to dynamically control network behavior. Deep learning models can be employed to classify network traffic based on various features. Once the model is trained, integrate it with the SDN controller to classify network traffic in real-time. During the implementation phase, for effectively

developing and evaluating a DL-based model for traffic classification in software-defined networks, contributing to advancements in network management, security, and resource allocation in SDN environments the following are the most important approaches.

The following pseudo algorithm outlines the steps to integrate a deep learning model with an SDN environment for the purpose of classifying network traffic. The model is trained to identify different types of traffic, which the SDN controller use to manage the network more effectively.

Algorithm: SDN_Deep Learning_Traffic_Classification

Input: Network traffic data

Output: Traffic classification results

1: Initialize the SDN environment

2: Deploy SDN controllers and configure network topology

3: Collect network traffic data

4: Preprocess the traffic data for deep learning

Feature extraction and Normalization

5: Split the preprocessed data into training and testing sets

6: Define the deep learning model architecture

Input layer with size equal to the number of features

Hidden layers

Output layer with size equal to the number of traffic classes

7: Train the DL model using the training set

Choose a loss function and an optimizer

Validate the model on the testing set

8: Integrate the trained model with the SDN controller

Develop a module for the controller to input traffic data into the model

Use the model's predictions to classify network traffic

9: Define network policies based on traffic classification

Quality of Service rules

Security measures (firewall rules)

10: Monitor network traffic in real-time using SDN switches

11: Classify incoming traffic using the deep learning model

12: Dynamically apply network policies based on classification

End Algorithm

SDN controller follows the following measures based on network traffic classification results from a deep learning model.

Flow Classification:

The deep learning model classifies network traffic into different categories (e.g., video streaming, web browsing, VoIP).

Each flow is labeled based on its predicted application or service.

Traffic Policy Mapping: The SDN controller maps the flow labels to predefined traffic policies.

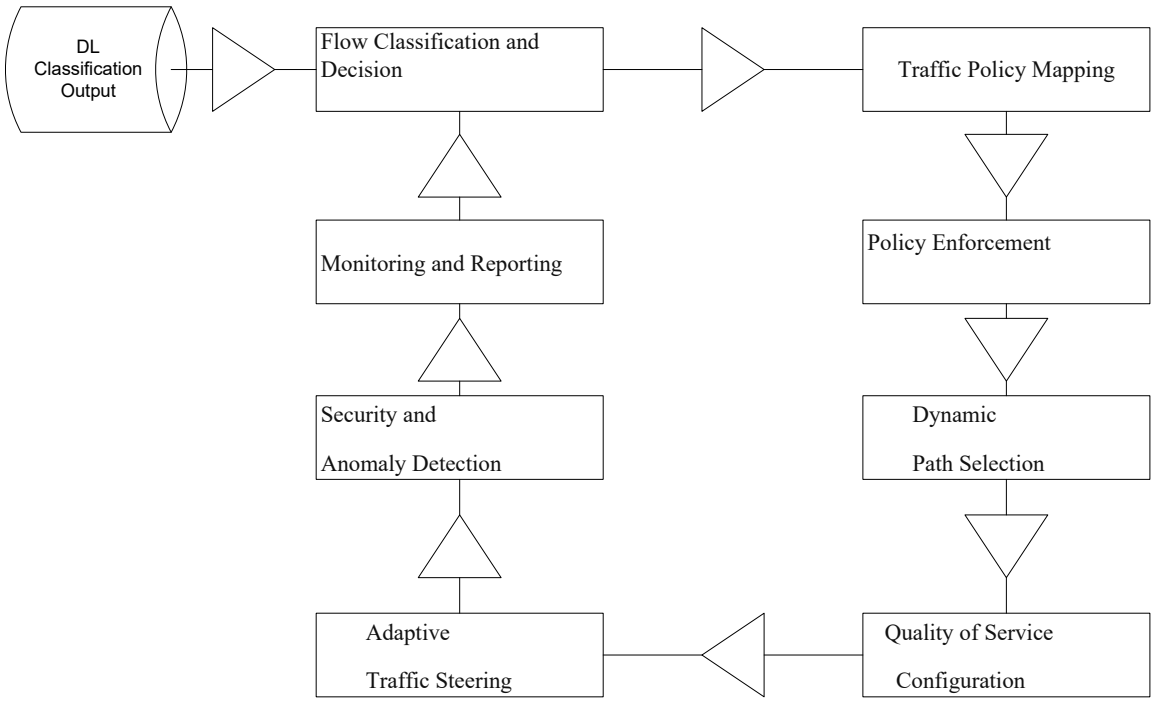

Figure 5: DL-SDN integration

Policies define how specific types of traffic should be handled (e.g., prioritization, bandwidth allocation).

Policy Enforcement:

The controller enforces the defined policies across the network.

It instructs switches and routers on how to handle traffic based on the flow labels.

Dynamic Path Selection:

Using the deep learning results, the controller dynamically selects optimal paths for different traffic types.

It considers factors like latency, congestion, and available bandwidth.

Quality of Service QoS Configuration:

The controller configures QoS parameters for each flow category.

It ensures that critical applications receive the necessary resources.

Adaptive Traffic Steering:

Based on real-time traffic conditions, the controller adjusts flow paths.

It reroutes traffic away from congested links or faulty devices.

Security and Anomaly Detection:

The controller collaborates with security modules.

It identifies anomalies (e.g., DDoS attacks) based on traffic patterns.

Monitoring and Reporting:

The controller continuously monitors traffic behavior.

It generates reports on network performance, compliance, and policy violations.

CHAPTER 4

4. RESULTS AND DISCUSION

The research work is conducted based on classifying the network traffic. The framework of classifying the network traffic classification is executed in diverse phases in which the data is pre-processed, more relevant features are selected, hyper parameters are tuned DL model is selected, the multilabel network traffic data is classified into multiple classes.

4.1. Unicauca-dataset-April-June-2019-Network-flows Overview

The dataset used for the model training is Unicauca-dataset-April-June-2019-Network-flows dataset. The dataset has the 209 attributes and target set which contain multiple classes of different network traffic category. The Unicauca-dataset-April-June-2019-Network-flows is a valuable dataset used for network traffic analysis. The followings are some key details:

Dataset Overview:

Name: Unicauca-dataset-April-June-2019-Network-flows

Purpose: It serves as a resource for studying network traffic behavior and developing traffic classification models.

Size: Contains approximately 2 million rows of network flow data.

Features:

The dataset includes various features extracted from network flows:

- Timestamp (ts): The time when the flow occurred.
- Duration: Flow duration in seconds.
- Source IP (src_ip) and Destination IP (dst_ip): IP addresses involved in the flow.
- Source Port (src_port) and Destination Port (dst_port): Port numbers.
- Protocol (proto): Transport protocol (e.g., TCP, UDP).
- Packet Count (packets) and Total Bytes (bytes): Flow-level statistics.
- Packet Size (packet_size): Average packet size.

Multilabel Labels:

The dataset provides labels for different network applications (e.g., web browsing, video streaming, file transfer). These labels allow researchers to perform multilabel traffic classification.

Use Cases:

Researchers and practitioners use this dataset for:

- ✓ Developing machine learning models for traffic classification.
- ✓ Evaluating the effectiveness of deep learning approaches.
- ✓ Investigating network anomalies, security threats, and DDoS attacks.

Availability:

This dataset is accessible and publicly available on Hugging Face Datasets. And additionally, it's available on GitHub and Kaggle.

4.1.1. Data Processing and Analysis

These steps are essential to ensure that the deep learning model has high-quality data to learn from, which is crucial for accurate network traffic classification in an SDN environment. Each step contributes to the model's ability to learn patterns of network traffic. This step starts with preprocessing the collected dataset by filtering out irrelevant traffic, removing duplicates, and encoding numerical and categorical features. Tools like Scikit-learn and Pandas were used for data cleaning, feature extraction, normalization, and any transformations required for preparing the data.

4.1.1.1. Handle missing values and handle infinite values

This section cleans the DataFrame by removing rows with any missing values and columns that are completely empty, ensuring that the data is free from null entries that could interfere with the training dataset.

```
sdn_train = sdn_train.dropna()
with pd.option_context('mode.use_inf_as_na', True):
    sdn_train.dropna(inplace=True)
```

By cleaning the data in this way, and ensure that the DataFrame does not contain rows with missing or infinite values, which is important, as these affect the performance of the

classification model. And after droping drops rows with NaN values (which now includes rows with infinity values) from the sdn_train DataFrame and modifies it in place.

4.1.1.2. Remove duplicate values

Next the task of identifying and removing some duplicate values and inconsistent columns, that will have no impact on performance classification task.

```
# Remove duplicate rows

sdn_train = sdn_train_ds.drop_duplicates()
```

sdn_train_ds.drop_duplicates(): This function call to drop_duplicates() returns a copy of the sdn_train_ds DataFrame with duplicate rows removed, by considering all columns to identify duplicates.

This ensures that the training dataset (sdn_train) contains only unique rows, with any row that is an exact duplicate of another (considering all columns) being removed. The original dataset has 2704839 attributes and 50 columns,after removing duplicates it's attributes have been reduced to 2704829 and 50 columns.

4.1.1.3. Handling less relevant features

In this step certain features (columns) are deemed less relevant or redundant for the classification task at hand. By removing these columns, it is possible simplify the dataset and potentially improve the performance of the classification models by focusing on more significant features.

```
sdn_train = sdn_train.drop(['flow_key', 'src_ip', 'dst_ip'], axis=1)
```

There for values 'flow_key', 'src_ip', 'dst_ip' has been reduced and we have now training dataset of shape = (2704829, 47).

4.1.1.4. One-hot-encode categorical features

This section converts the categorical 'application_protocol' and 'web_service' column into a numerical format that deep learning algorithms can work with, by creating a separate binary column for each category.

```
# Encode categorical features

ohe = OneHotEncoder(sparse=False)

protocol_ohe = ohe.fit_transform(sdn_train['application_protocol']. values.reshape(-1, 1))

protocol_cols = ohe.get_feature_names_out(['application_protocol'])
```

protocol_df = pd.DataFrame(protocol_ohe, columns=protocol_cols)

Concatenate the one-hot encoded protocol columns to the original dataframe

sdn_train = pd.concat([sdn_train, protocol_df], axis=1)

Drop the original protocol column

sdn_train = sdn_train.drop('application_protocol', axis=1),

This phase is repeated for 'web_service' column

Then these key columns that are categorical and converted into numerical values. Next these values are concatinated to the original columns. After all these steps we have a dataset of shape = (2704829, 210).

Data visualization:

The graph below represents the distribution of samples across different categories in the training dataset.

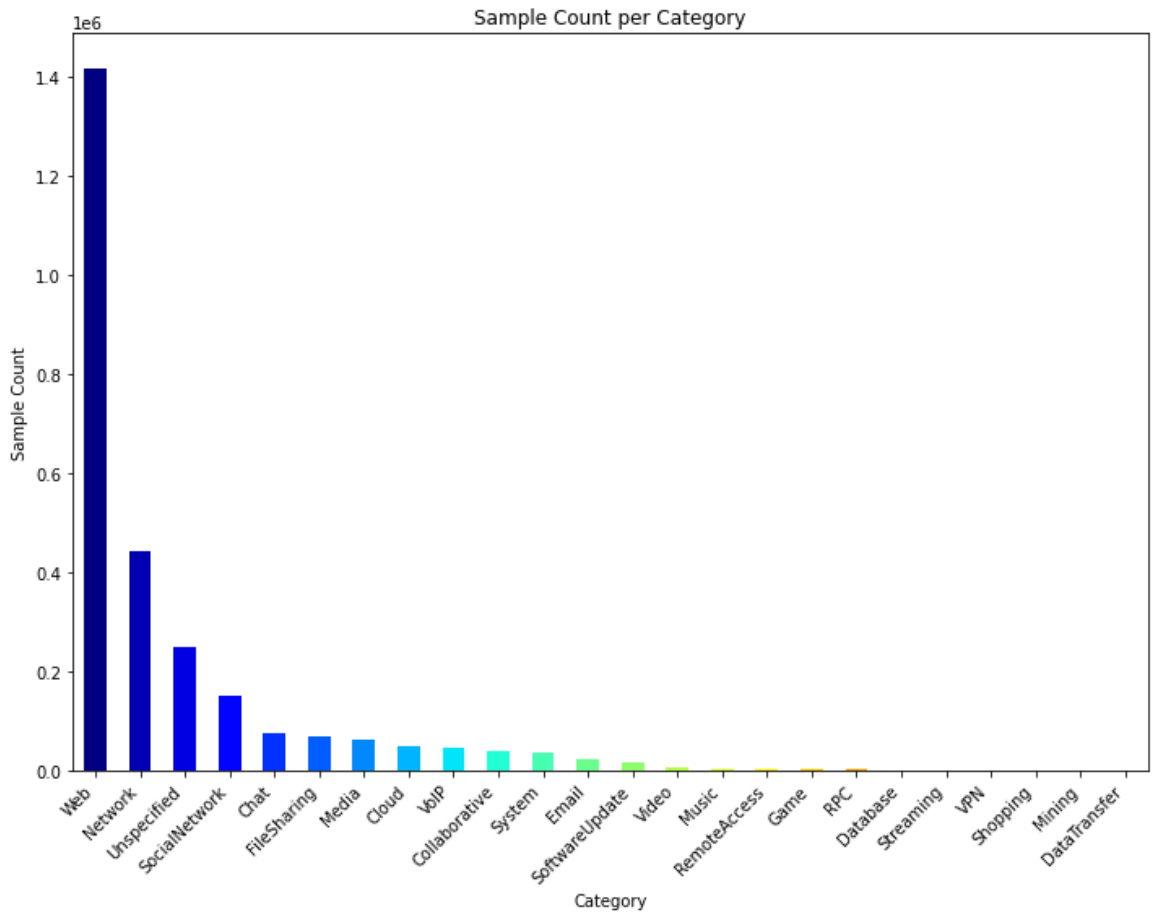

Figure 6: Visualize the dataset

Each bar corresponds to a category, and its height reflects the number of samples in that category. This visualization helps identify which categories are more prevalent and which are underrepresented.

train_features = sdn_train.copy()

sdn_train DataFrame is made to create a separate DataFrame for features.

train_labels = train_features.pop('category')

The 'category' column is removed from train_features and stored in train_labels, effectively separating the features from the target variable.

Value Counts of Target Variable:

print(sdn_train['category'].value_counts())

Table 3: Sample Traffic features

R/N	Column Names	Descriptions
1	src_ip_numeric	The IP address of the source device in numeric format
2	src_port	The port number of the source device
3	dst_port	The port number of the destination device
4	proto	The protocol used for communication
5	pktTotalCount	The total number of packets in the flow
6	octetTotalCount	The total number of bytes in the flow
7	min_ps	The minimum packet size in the flow
8	max_ps	The maximum packet size in the flow
9	avg_ps	The average packet size in the flow
10	std_dev_ps	The standard deviation of packet size in the flow
11	flowStart	The start time of the flow
12	flowEnd	The end time of the flow
13	flowDuration	The duration of the flow
14	f_flowStart	The start time of the forward flow
15	f_flowEnd	The end time of the forward flow
16	f_flowDuration	The duration of the forward flow
17	f_min_piat	The minimum inter-arrival time between packets in the forward direction of the flow
18	f_max_piat	The maximum inter-arrival time between packets in the forward direction of the flow

19	f_avg_piat	The average inter-arrival time between packets in the forward direction of the flow
20	b_std_dev_piat	The standard deviation of inter-arrival time between packets in the backward direction of the flow
21	Flow End Reason	The reason for the flow ending
22	category	The category of the flow.
23	application_protocol	The application protocol used for communication
24	web_service	The web service used for communication

4.1.2. **Feature Engineering**

The network traffic dataset contains the following traffic categories:

Table 4: Traffic category lists(target)

R/N	Category of traffic	Descriptions
1	Web	This category includes traffic from web browsers, web servers, and web applications. Web traffic is usually unicast, and is also one of the most common types of traffic on the Internet and includes things like web browsing, email, and file transfers.
2	Network	This includes traffic from network devices, such as routers, switches, firewalls, and VPNs. Network traffic can be used for various purposes, such as routing, switching, security, and communication.
3	Unspecified	This is a traffic that does not fit into any of the other categories. It may be due to errors in the data collection or classification process, or it may be due to unknown or new applications or services that are not yet identified by the network devices
4	Social-network	Social network traffic is usually unicast or multicast as well. Social network traffic can be used for various purposes, such as sharing content, chatting with friends
5	Chat	Chat traffic is usually unicast as well. Chat traffic can be used for various purposes, such as voice and video calls, text messages, file sharing, and group discussions.
6	Download-File Transfer-FileSharing	Download-file-transfer-file-sharing (DFTFS) traffic is usually unicast or multicast depending on the source and destination addresses of the packets
7	Media	This category includes traffic from media streaming services (such as Netflix), video conferencing services (such as Zoom), online gaming platforms (such as

		Steam), music streaming services (such as Spotify), podcasting platforms (such as Spotify), radio stations (such as BBC Radio 4), etc.
8	Cloud	This is a traffic category from cloud computing services (such as Amazon Web Services), cloud storage services (such as Google Drive), cloud backup services (such as Carbonite), cloud gaming services (such as Nvidia GeForce Now), etc.
9	VoIP	This category includes traffic from Voice over Internet Protocol (VoIP) applications and services such as Skype Voice calls, WhatsApp Voice calls, Google Voice, etc.
10	Collaborative	Traffic from collaborative applications and services such as Google Docs, Microsoft Office 365, Dropbox Paper, Slack Channels, etc.
11	System	Traffic from system administration, monitoring, diagnostic, security tools, etc.
12	Email	Traffic from email applications and services such as Gmail, Outlook.com, Yahoo Mail, etc.
13	Software Update	Traffic from software update applications and services such as Windows Update, Mac
14	Video	This category includes traffic from video streaming services (such as Netflix), video conferencing services (such as Zoom), online gaming platforms (such as Steam), etc.
15	Music	Music traffic is usually unicast or multicast depending on the source and destination addresses of the packets. Music traffic can be used for various purposes such as listening to music online, downloading podcasts, tuning in to radio stations, etc.
16	Remote Access	Includes traffic from remote access applications and services such as TeamViewer, AnyDesk, LogMeIn, etc.
17	Game	Includes traffic from game applications and services such as Steam, Epic Games Store, Origin, etc.
18	RPC	This is a traffic from remote procedure call (RPC) applications and services such as Google Cloud Functions, AWS Lambda, Azure Functions, etc.
19	Database	Database Traffic can be used for various purposes such as storing data, querying data, updating data, backing up data online, etc.
20	Streaming	This is a traffic from streaming applications and services such as YouTube, Twitch, Facebook Live, Instagram Live, etc.
21	VPN	This includes Traffic from virtual private network

		(VPN) applications and services such as NordVPN, ExpressVPN, ProtonVPN etc.
22	Shopping	Shopping traffic is usually unicast or multicast depending on the source and destination addresses of the packets.
23	Mining	Traffic from mining applications and services, such as Bitcoin mining, Ethereum mining, etc. Mining traffic can be used for various purposes such as generating cryptocurrency, validating transactions, securing networks, etc.
24	Data Transfer	This category includes traffic from data transfer applications and services, such as FTPS (FTP over SSL/TLS), P2P (Peer-to-Peer), BitTorrent (BT), etc.

The following section describes input data preperoccesing model that can be re-used for retraining with other models.

The training features dictionary contains input features.

For each feature:If the data type is numeric (float32), it is added to numeric inputs.If the data type is non-numeric (object), it is processed further.

Normalization and Concatenation:

Numeric features are concatenated into a single tensor X.

All numeric inputs are normalized using the norm layer.

For each non-numeric feature:

A string lookup layer (lookup) is created with unique vocabulary.

A one-hot encoding layer (one_hot) is applied to the lookup results.

The processed non-numeric input is appended to preprocessed_inputs.

Concatenation of All Preprocessed Inputs:

All preprocessed inputs (numeric and one-hot encoded) are concatenated into a single tensor preprocessed_inputs_cat.

Create a Preprocessing Model:

A Keras model (train_preproccessing) is defined with the input features and the concatenated preprocessed inputs. The model maps raw input features to the preprocessed features.

Create a dictionary of features:

A dictionary (features_dict) is created with a single example of each feature.

Apply preprocessing to the input raw data:

The train_preproccessing model is applied to the training data (features_dictionary). The model will be used repeatedly without re-running data processing tasks, which defines a neural network model, preprocesses input features both numeric and non-numeric, and visualizes the preprocessing architecture.

4.1.2.1. Feature selection

This involves to extract relevant features from the network traffic data that is used as input to the DL model. And select a subset of relevant features that are most useful for the classification task.

Random Forest Classifier is used to perform feature selection based on the importance of each feature. Random forests are an ensemble learning method that operates by constructing multiple decision trees during training and outputting the class that is the mode of the classes of the individual trees.

Using a RF Classifier to fit a model to the training data x and y. After training, it calculates the importance of each feature with respect to the target variable y. The importance reflects how much each feature contributes to the model's ability to make accurate predictions.

In this case, only a single least important feature is selected and removed as indicated by the next few lines of steps that removes the least important feature from the training dataset ‘x’ by deleting the column at the index ‘indices_to_remove’.

The first most step is to separate target and independent variables:

'x' is feature matrix and 'y' is the target vector

x= train_features

y = train_labels

Train the model

```
model = RandomForestClassifier()

model.fit(x, y)

# Get feature importances

importances = model.feature_importances_

print ("Feature importances:", importances)

# Sort the feature importances in ascending order and get the indices of the least important features

indices_to_remove = importances.argsort()[:1]

print ("Indices to remove:", indices_to_remove)

# Print the names of the least important features

print ("Names of the least important features:", [train_labels[i] for i in indices_to_remove])

# Drop the least important features from your dataset

X = np.delete(x, indices_to_remove, axis=1)
```

The np.delete function is used to remove the column at the index indices_to_remove from the feature matrix x. The axis=1 argument specifies that a column (feature) should be deleted rather than a row.

The result is a new feature matrix ‘X’ with the least important feature removed, which potentially improve the performance of the model if that feature was noisy or irrelevant. This process can be repeated to remove multiple unimportant features iteratively. In other words, the most relevant feature impacting the performance of the model has been identified.

After feature selection phase completed the traing dataset has a shape =(2704829, 208).

And from these features some of the top 20 most important features that impact the overall efficiency of ML models including RF, SAE and DNN are described below.

dst_port: The destination port number used by the receiver. It's important for identifying the type of service the traffic is intended for.

max_ps: The maximum packet size observed in the traffic. This can indicate the volume and type of data being transmitted.

f_max_ps: The maximum packet size observed in the forward direction (from source to destination).

web_service_DNP3: A binary feature indicating whether the Distributed Network Protocol 3 (used in industrial, power grid, and utility company communications) is detected.

b_max_ps: The maximum packet size observed in the backward direction (from destination to source).

min_ps: The minimum packet size observed in the traffic.

avg_ps: The average packet size across the entire traffic. This gives a general idea of the data flow's consistency.

application_protocol_TLS: A binary feature indicating the presence of TLS, which is used for secure communication.

std_dev_ps: The standard deviation of packet sizes, which shows the variability of packet sizes in the traffic.

web_service_Unencrypted_Jabber: A binary feature indicating the use of Jabber/XMPP protocol without encryption.

src_port: The source port number used by the sender. It can be used to identify specific applications or services.

web_service_FTP_DATA: A binary feature indicating the use of the File Transfer Protocol for data transfer.

src_ip_numeric: The numeric representation of the source IP address, which can be used for geolocation or network topology analysis.

f_flowStart: The timestamp of the start of the flow in the forward direction.

flowEnd: The timestamp of when the flow ended.

These features are crucial for RF, SAE and DNN classification models as they provide insights into the nature and behavior of network traffic, which has been used to classify network traffic effectively. Understanding these features helps in interpreting the model's decisions and improving its performance. The following figure shows the top 20 most important network traffic features that are capable of influencing classification models' performance of the proposed model, the base-line models compared (i.e. RF, and SAE).

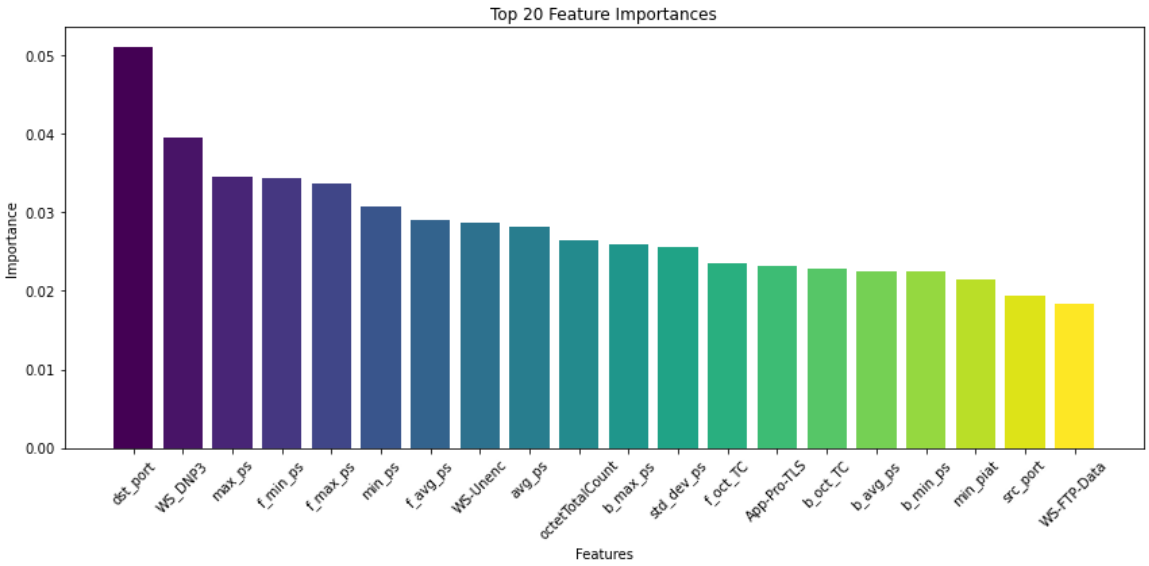

Figure 7: The top 20 most important features

4.1.2.2. Data splitting

The dataset is split into training, validation, and test sets, with the validation set being stratified based on the labels in y_train. This task is preparing the data for a DL workflow by creating distinct sets for training, validation, and testing. This allows for the training of the model, tuning of hyperparameters, and finally, evaluation of the model's performance on unseen data. The use of a validation set helps in fine-tuning the model without touching the test set, which should only be used for the final assessment.

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.15, random_state=0)

This splits the features X and the labels y into training (X_train, y_train) and testing (X_test, y_test) sets.

A test size=0.15 means that 15% of the data will be set aside for the testing set, and the remaining 85% will be used for training. A random_state=0 ensures the same split each time the code run with the same data.

X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.15, stratify=y_train, random_state=0)

This further split the training set from the first line into a smaller training set (X_train, y_train) and a validation set (X_val, y_val).

A test_size=0.15 is used, which means 15% of the original training set is now allocated to the validation set. The stratify=y_train parameter ensures that the validation set has the

same proportion of class labels as the original training set, which is important for maintaining a representative subset.

4.1.2.3. One-Hot encoding

Each categorical value (the type of network traffic) is represented by a binary vector. This is necessary for multi-label classification problems where the model should predict the probability of each class for a given sample.The labels are converted to one-hot encoded vectors, which is a common format for classification tasks in neural networks.

4.1.2.4. Label encoding

In this technique, each categorical value is assigned a unique numerical value. For example, if there are 3 types of traffic, the first type may be assigned the value 0, the second 1, and the third 2. This encoding is also commonly used for classification problems.

```
#label encoding
from sklearn.preprocessing import LabelEncoder
label_enc = LabelEncoder()
label_enc.fit(y)
y = label_enc.fit_transform(y)
classes=label_enc.classes_
```

4.1.2.5. Categorical encoding

In this technique, each categorical value is replaced with the mean of the target variable (i.e., the class label) for that category. This encoding is useful in situations where there are many categories with many occurrences.

```
#Categorical Features
label_enc = LabelEncoder()
label_enc.fit(y)
y = label_enc.fit_transform(y)
```

The to_categorical function is used to create a one-hot encoded representation of the labels, which is a standard approach when working with categorical data in neural networks.

num_classes is the total number of classes,This ensures that the binary matrix has columns equal to the number of classes.

```
classes=label_enc.classes_
num_classes = len(np.unique(y_train))
y_train = tf.keras.utils.to_categorical(y_train, num_classes)
y_test = tf.keras.utils.to_categorical(y_test, num_classes)
```

y_val = tf.keras.utils.to_categorical(y_val, num_classes)

4.1.2.6. Feature scaling

Feature scaling, is conducted specifically standardization, on the training, testing, and validation datasets which calculates the mean and standard deviation of the training, testing and validation dataset and then standardizes it by subtracting the mean and dividing by the standard deviation.

```
X_train = (X_train - np.mean(X_train)) / np.std(X_train)

X_test = (X_test - np.mean(X_test)) / np.std(X_test)

X_val = (X_val - np.mean(X_val)) / np.std(X_val)
```

4.1.2.7. Reshape the features

```
# Reshape the features to match the input shape

input_shape = (X_train.shape[1],)

X_train = X_train.reshape(-1, *input_shape)

X_test = X_test.reshape(-1, *input_shape)

X_val = X_val.reshape(-1, *input_shape)
```

4.1.3. Define the Proposed Model

This model used a CSV dataset of network traffic, where each row represents a traffic instance and each column a feature. After training, the model would be able to classify new network traffic data into one of the 24 categories based on the learned patterns during training. The effectiveness of this model would depend on the quality and quantity of the data, the chosen architecture, and the hyper-parameters used during training.

4.1.3.1. Hyper parameter tunning

Import: the necessary libraries, including TensorFlow and required dependencies.

In this step the most efficient parameters have been selected using a model developed to handle the hyper parameter tunning task, which has generated all optimal parameters that are going to fit into the proposed model for the classification problem at hand. The following sample of code shows the hyper parameter model.

```
# Define a function to create the model with hyperparameters
```

```
def build_model(hp):
    model=tf.keras.models.Sequential()
    model.add(tf.keras.layers.Dense(hp.Int('units_input',min_value=32,max_value=51
    2, step=32), activation='relu', input_dim=208))
    # …….. (rest of code) .....
    return model
    # Create a tuner
    tuner = RandomSearch(
        build_model,
        objective='val_acc',
        max_trials=10,
        executions_per_trial=1,
        directory='my_dir',
        project_name='hparam_tuning'
    )
# Perform hyperparameter tuning
    tuner.search(X_train, y_train, epochs=5, validation_data=(X_val, y_val))
# Get the optimal hyperparameters
    best_hps = tuner.get_best_hyperparameters(num_trials=1)[0]
# Build the model with the optimal hyperparameters and train it
    model = build_model(best_hps)-
```

Load the DL model: Load the DL model with the tunned parameters that will be trained on a multilabel dataset, in our case DNN. The proposed model is designed for multilabel classification of network traffic, use the sigmoid activation function in the output layer which is suitable for multilabel dataset and the binary_crossentropy loss function during compilation. The following section briefly describes the structure of the model:

Sequential Model: This is a linear stack of layers where each layer has exactly one input tensor and one output tensor.

Dense Layers: These are fully connected neural network layers. Each layer's output is computed from the input by a matrix multiplication (with weights) and a bias offset, followed by an activation function.

Activation Function 'relu': The rectified linear unit (ReLU) activation function is used in all hidden layers. It introduces non-linearity to the model, allowing it to learn more complex patterns.

$$f(x) = \backslash max(0, x)$$

This means that if the input (x) is positive, the output is (x). If the input (x) is negative, the output is (0).

Output Layer with 'sigmoid': The final layer has 24 units, corresponding to the number of classes in the dataset. The sigmoid function is used to output a probability distribution over the 24 classes.

Compilation: The model is compiled with the Adam optimizer, a popular choice for deep learning models due to its efficiency. The learning rate is set to 0.001. The loss function is binary_crossentropy, suitable for multilabel classification problems. The metrics include accuracy (acc), F1 score (f1_m), precision (precision_m), and recall (recall_m), which are important for evaluating the performance of classification models.

The following table summarizes parameters that has been tuned for the proposed model:

Table 5: Depth and parameters of the proposed model

Layer Type	Neurons	Activation	Input/Output Shape	Dropout Rate	Description
Dense	512	ReLU	Input: 208	-	Fully connected layer
Batch Normalization	-	-	Output: 512	-	Normalization layer
Dropout	-	-	Output: 512	0.1	Dropout layer

Dense	256	ReLU	Output: 256	-	Fully connected
Batch Normalization	-	-	Output: 256	-	Normalization layer
Dropout	-	-	Output: 256	0.1	Dropout
Dense	128	ReLU	Output: 128	-	Fully connected l
Batch Normalization	-	-	Output: 128	-	Normalization
Dropout	-	-	Output: 128	0.1	Dropout
Dense	64	ReLU	Output: 64	-	Fully connected
Batch Normalization	-	-	Output: 64	-	Normalization
Dropout	-	-	Output: 64	0.1	Dropout
Dense	32	ReLU	Output: 32	-	Fully connected
Batch Normalization	-	-	Output: 32	-	Normalization layer
Dropout	-	-	Output: 32	0.1	Dropout
Dense	16	ReLU	Output: 16	-	Fully connected
Batch Normalization	-	-	Output: 16	-	Normalization
Dropout	-	-	Output: 16	0.1	Dropout
Dense (Output Layer)	24	Sigmoid	Output: 1	-	Binary classification layer

Dropout layers suggest that they were considered for the model to prevent overfitting by randomly setting a fraction of input units to 0 at each update during training time, which helps prevent overfitting.

Optimizer: ‘Adam optimizer’ is used which is widely used in most deep learning applications. It leverages the power of adaptive learning rates methods to find individual learning rates for each parameter.

Learning rate: The learning rate is the default learning rate of the ‘Adam Optimizer’ which is 0.001.

Loss: ‘Binary crossentropy’ is used. It is used for multi label categorization

i.e. when each instance can belong to only one class.

Epochs: The model was trained through 100 epochs. An epoch represents a complete training over the training dataset.

Metrics: Metric used is Accuracy. Accuracy indicates the number of correctly classified instances.

Table 6 shows the summary of a SAE model the number of parameters, type of layesr, the shape of the output it produces, which has been trained on the same dataset as the proposed dataset.

Table 6: Depth and parameters of the SAE model

Layer (type)	Output Shape	#Parameters
input_3 (InputLayer)	(None, 208)	0
dense_12 (Dense)	(None, 128)	26752
dense_13 (Dense)	(None, 64)	8256
dense_14 (Dense)	(None, 32)	2080
dense_15 (Dense)	(None, 16)	528
dense_16 (Dense)	(None, 8)	136
dense_17 (Dense)	(None, 16)	144
dense_18 (Dense)	(None, 32)	544
dense_19 (Dense)	(None, 64)	2112
dense_20 (Dense)	(None, 128)	8320
dense_21 (Dense)	(None, 24)	3096

4.1.4. Training the Classification Model

The result from the data processing phase has been used to train three separate models, that includes RF, SAE, and the proposed model. And their performance to classify network traffic into different categories and recognizing different types of network traffic, utilizing the patterns and features it has learned from the data has been examined in the following section.

Train the RF model:

```
rfc = RandomForestClassifier()

rfc.fit(X_train, y_train)
```

Train the SAE and the proposed DNN model:

```
history=model.fit (X_train, y_train, batch_size=128, epochs=100, validation_data=(X_val, y_val))
```

4.1.5. Model Evaluation

The performance metrics from deep learning model indicate that the proposed model is performing quite well.

Accuracy (acc): The accuracy is over 90%, indicating that the model correctly predicts the target class most of the time.

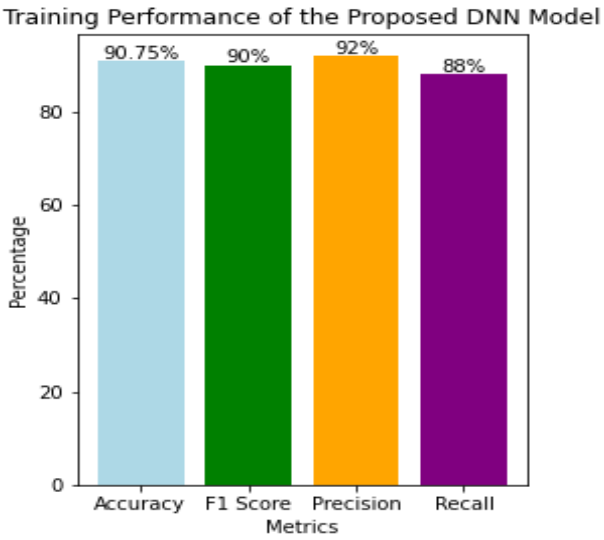

Figure 8: Performance metrics summary

4.1.5.1. Training Performance of the Proposed Model

Training accuracy

Figure 8 depicts the percentage of correct predictions that the model makes on the training data, that indicates how well the model fits the training data, but it does not necessarily indicate how well the model will generalize to new data. The deep learning model's training and validation performance over 100 epochs is summarized as follows:

Training Accuracy: 78.27% at 1st epoch and improved to 90.75%.

The model shows consistent improvement in accuracy, precision, recall, and F1 score, both in training and validation phases, indicating effective learning and generalization to the validation data. The final validation accuracy of 91 suggests the model is highly accurate in evaluating the validation dataset.

Training loss: The loss values are quite low (around 0.020), which suggests that the model's predictions are close to the actual targets.

Figure 9 shows training loss of 0.02 which is the value of the cost function that the model tries to minimize during training. It indicates how much error the model makes on the training data.

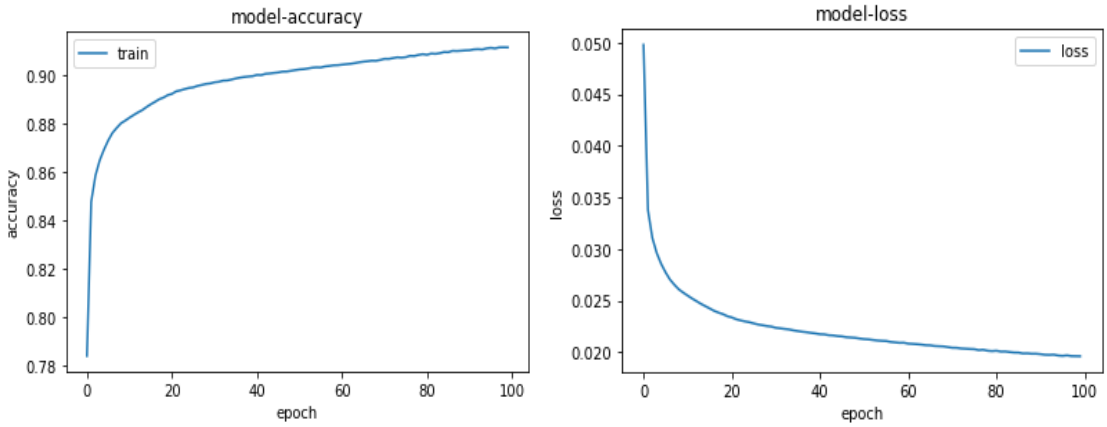

Figure 9: Model training & Training Loss

Validation accuracy:

Figure 10 indicates the percentage of correct predictions that the model makes on the validation data, which is used to monitor the model's learning progress and choose the best model configuration. Validation Accuracy: 90.78%

Validation loss: Figure 10 shows the variation between the predicted outputs and the true outputs for the validation data which indicates how much error the model makes on the validation data. Validation loss: 0.025

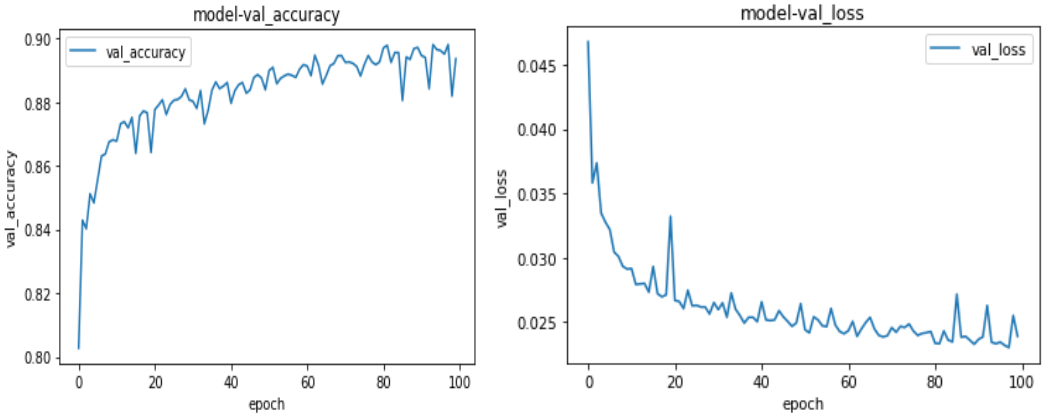

Figure 10: Validation accuracy and Validation loss

F1 Score (f1_m): The F1 score, which balances precision and recall, is also over 90%, showing a good balance between the precision and recall of the model.

Precision (precision_m): Precision is consistently above 92%, meaning when the model predicts a class, it is correct most of the time.

Recall (recall_m): Recall is slightly lower than precision but still above 88%, indicating that the model is capable to identify most of the relevant paterns. In addition, the validation metrics closely follow the training metrics, which is a good sign that the model is generalizing well and not overfitting.

4.1.5.2. Testing performance

It is often used to evaluate the model's final performance and to compare different models or methods. The overall testing performance of the deep learning model based on the output provided is excellent.

Test Accuracy: An accuracy of 92 % shows that the model correctly predicts the outcomes on the test dataset with high reliability. The performance metrics indicate that the model is performing consistently well on both the training and test datasets, with high accuracy and other metrics. The time taken for the last step (146 seconds for 61070 steps) suggests that the model is relatively efficient in making predictions. The overall accuracy of 90.75% on the training data is a good sign that the model has learned the patterns in the data effectively.

Model Prediction**:** In this phase of a deep learning (DL) model trained on a multi-label network traffic dataset with an input dimension of 208 and 24 classes, the model applies what it has learned during training to new, unseen data.

The model receives new network traffic data as testing input data. This data is a vector with 208 features, which represents different characteristics of the network traffic. The model processes the input data through its learned neural network architecture. It uses the weights and biases adjusted during the training phase to make sense of the input.

Output Prediction: The model outputs a prediction vector with 24 elements, each corresponding to a different traffic class. Each element of the vector represents the model's confidence that the input data belongs to the respective class. Since it's a multi-label classification, multiple classes can be assigned to each input. A threshold is set (often 0.5 by default), and any class with a confidence score above this threshold is considered a predicted label for the input data. Figure 11 shows the final output as a list of traffic types that the model predicts the input data belongs to, based on the learned patterns from the target domain dataset. This process allows for the automated classification of network traffic, aiding in tasks such as anomaly detection, security monitoring, and traffic management.

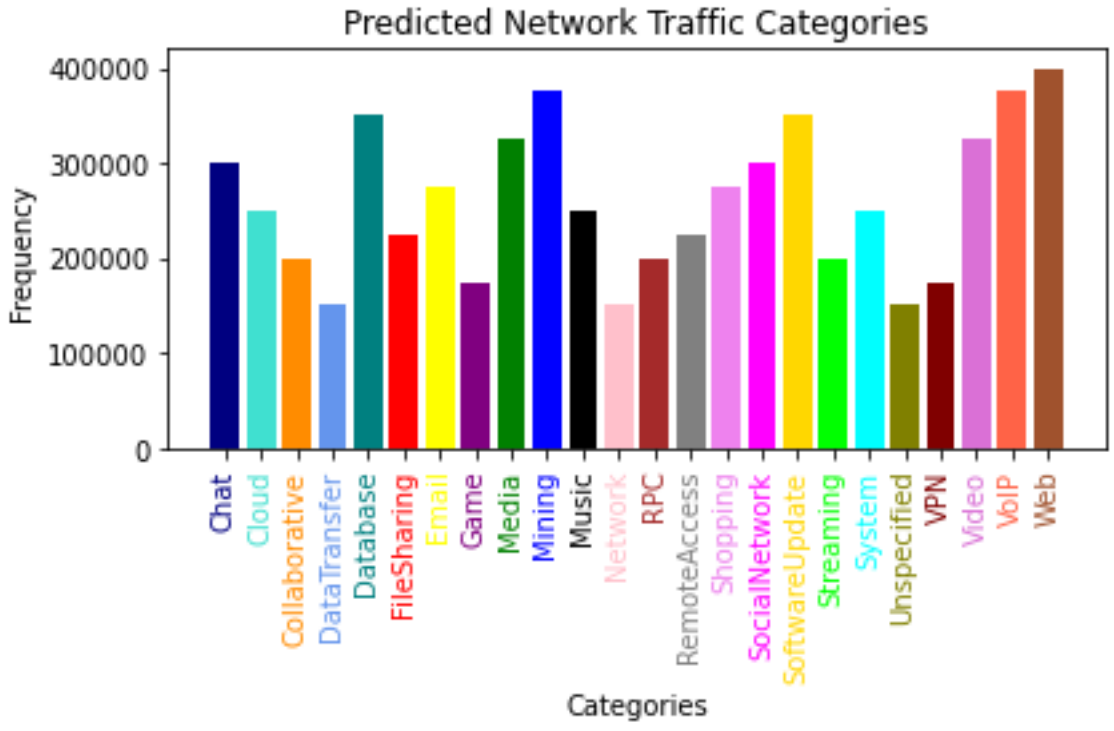

Figure 11: Prediction frequency of network traffic classes

Confusion matrix

It compares the actual labels with the predicted labels and counts how many times each pair occurs. The diagonal elements of the confusion matrix represent the true positives, while the off-diagonal elements represent the false positives and false negatives.

The following figure demonstrates and interprets the prediction accuracy of the proposed network traffic classification model:

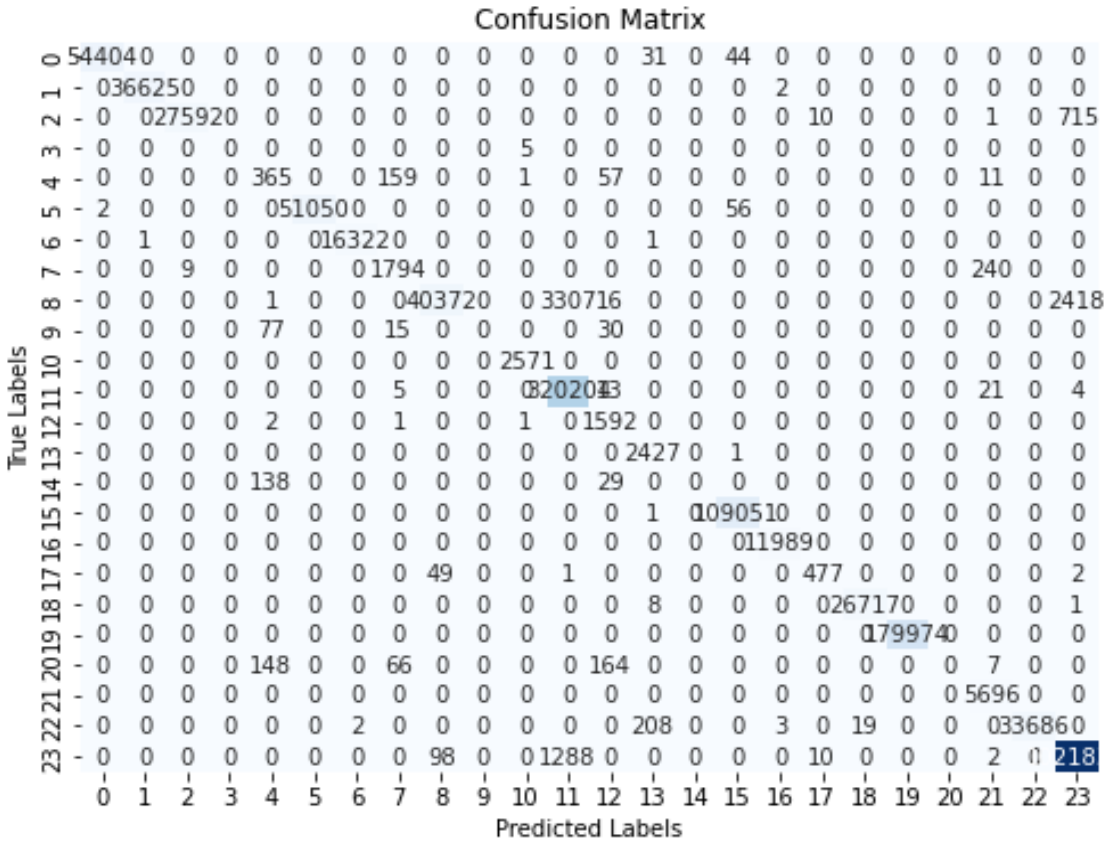

Figure 12: Confusion matrix

ROC curve: It plots the true positive rate (TPR) against the false positive rate (FPR) for each class, using different thresholds to decide the class membership. A model with perfect discrimination (no overlap between the two distributions) has a ROC curve that passes through the upper left corner (100% sensitivity, 0% FPR). The closer the ROC curve is to the upper left corner, the higher the overall accuracy of the test. Figure 12 depicts the ROC and AUC curves;

AUC: This is the area under the ROC curve. An AUC of 0.5 suggests no discrimination (equivalent to random chance), while an AUC of 1.0 indicates perfect discrimination.

Interpreting the Curve: An AUC curve near the diagonal line (AUC close to 0.5) indicates a poor model that has no discrimination capacity, essentially performing no better than random guessing. On the other hand, a curve that bows towards the upper left indicates a good model.

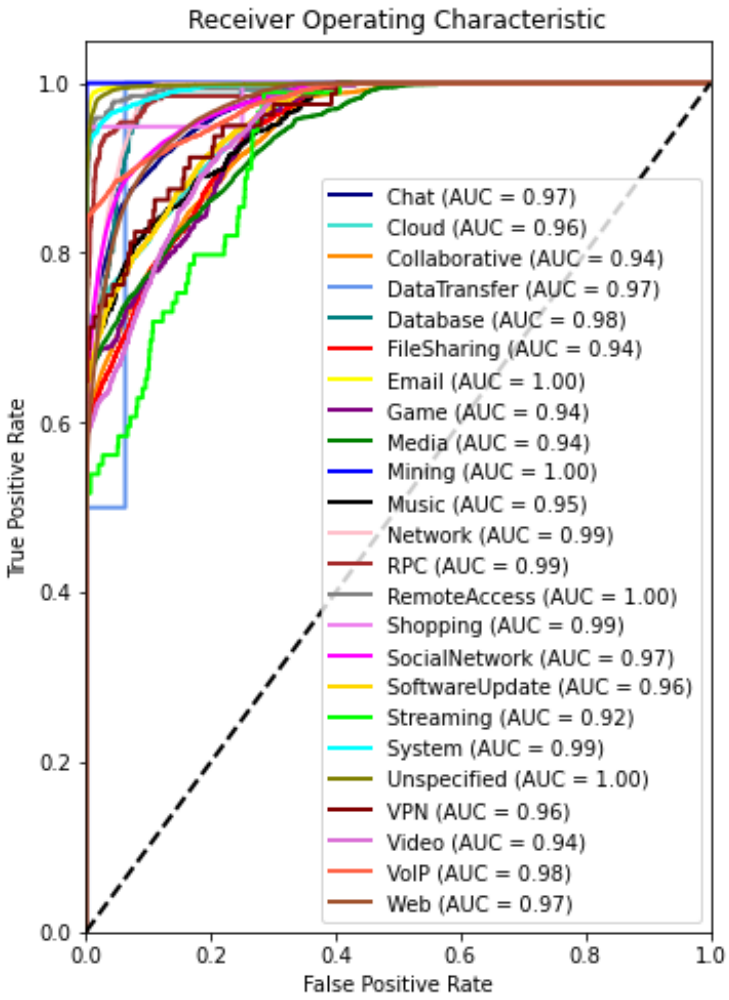

Figure 13: AUC & ROC curve

The larger the bow, the better the model is at distinguishing between the positive and negative classes. In addition, a curve that falls below the diagonal line indicates a model performing worse than random chance. This is often due to a problem with the model or the data.

4.1.6. Probability Distribution of Prediction

Based on probability distribution, the proposed model's effectiveness in providing accurate probability distributions depends on the quality of the training data and the architecture of the neural network.

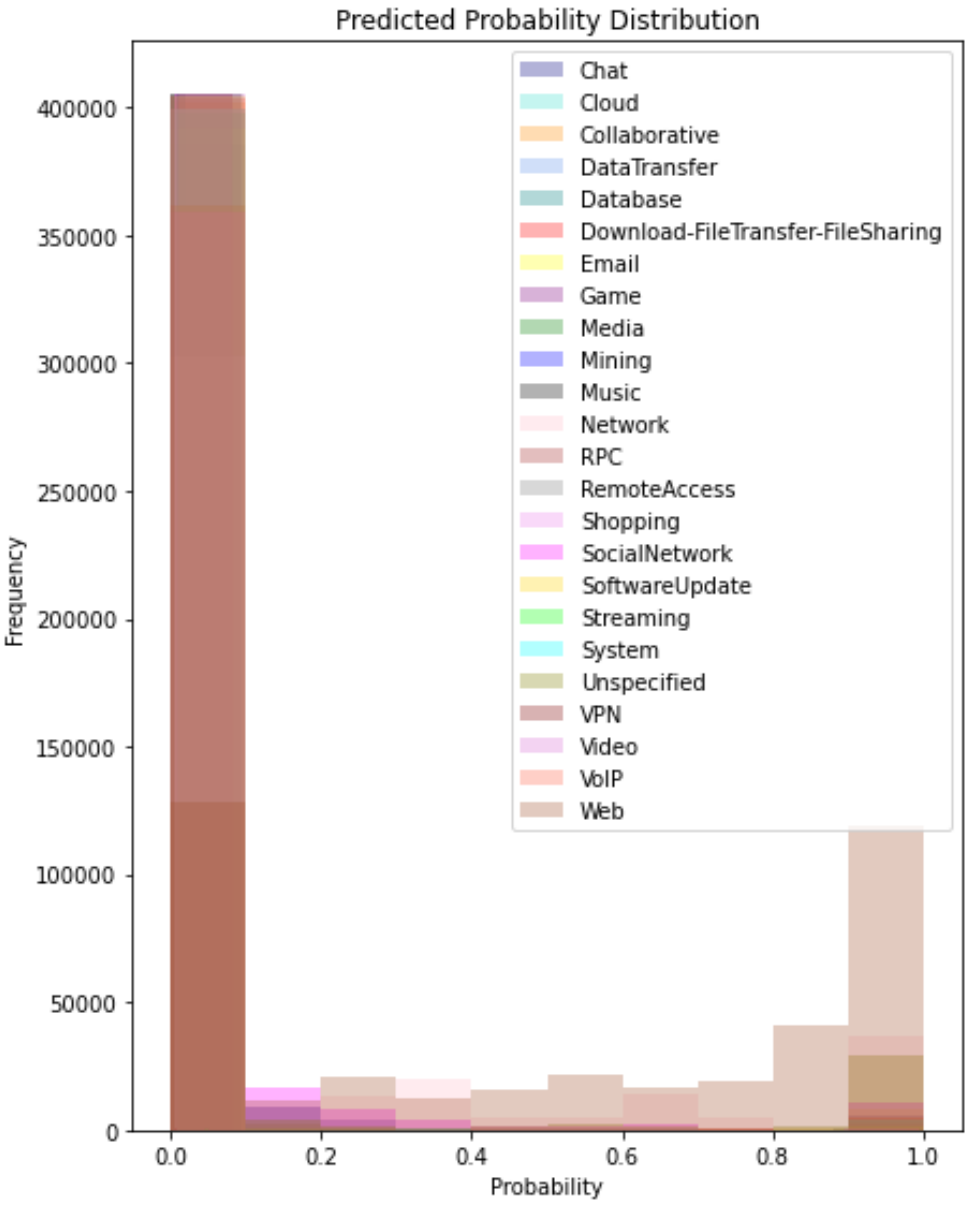

Figure 14: Prediction probability

The probability distribution helps to manage this complexity by providing a way tassess the likelihood of each potential label being correct.

Understanding this distribution is crucial for making informed decisions based on the model's output. If a model assigns high probabilities to several classes for an input, a

decision-maker might choose to associate the input with all those labels. In opposite, if the probabilities are low, they might decide to investigate further before assigning class labels to traffic classes.

4.1.7. Comparison with Baseline Methods

In this paper, the proposed traffic classification model is based on DL mechanisms, over SDN.To validate the effectiveness of the proposed DL model, it is important to compare its performance with baseline methods in which all these models are trained using the same dataset and all have gone the same preprocessing approaches. The dataset is divided into training, validation, and testing sets. The selected baseline models are Random Forest (RF), SAE. And performance analysis is conducted in terms of accuracy, precision, recall, and F1 indicators, and compared the results with two ML models. The training results have achieved more than 90% accuracy on identifying 24 categories of applications for all three different models. Furthermore, the model has achieved approximately 91% accuracy for testing prediction.

Table 7: Summary of baseline models' performances

Model	Training				Testing
	AC	F1 Score	Precision	Recall	
RF	87%	65%	89%	56%	86
SAE	89%	89%	91%	87%	88%
Proposed DNN model	90.75%	90%	92%	88%	92%

Tabel:7 depicts the comparison of the proposed technique with various classification models like Random Forest and SAE to classify the network traffic. The proposed model achieves maximum accuracy of 90.75 percent as compared to other models. The maximum accuracy indicates ability of the proposed model to classify network traffic with multiple features into multiple traffic classes or traffic category.

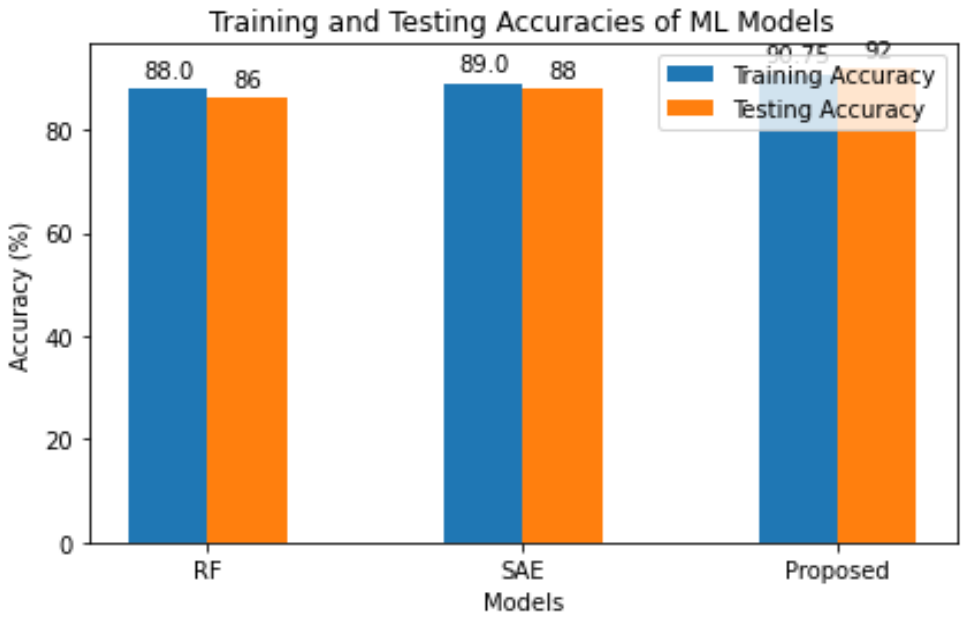

Figure 15: Performance metrics comparison

4.1.8. Training Time of the Proposed Model

Python time module have been applied which involves using the time module in Python to record the start and end time of the training process. This is a simple and straightforward way to measure the training time in seconds. The model took approximately 10244.62 seconds (about 2.84 hours) to train, which is reasonable for a deep learning model.

4.1.9. CPU and Memory Utilization

Based on the output from profiler tool which is installed and integrated with the proposed model's python code have been used to collect and display various metrics and information

CPU Utilization: The CPU usage was 42.1%, which suggests that the training process was computationally intensive but did not fully saturate the CPU.

Memory Utilization: The memory consumption of 4%, which is quite low and indicates that the model and the dataset fit comfortably in the available memory without causing any memory-related bottlenecks. Overall, the model shows good performance and efficiency in terms of resource utilization.

CHAPTER 5

5. CONCLUSSION AND RECOMMENDATION

5.1. Conclusion

In conclusion, accurate traffic classification is of fundamental importance to various network activities such as fine-grained network management and resource utilization. And our model combines accuracy, efficiency, and traffic identification, contributing to the advancement of network traffic monitoring, resource utilization and allocation, cyber security optimization, enhancing QoS and traffic prediction and driving force for flexible network management and operation. The experimental results reveal insights into the performance of the trained models. The key findings are discussed as follows:

Notably, the proposed model achieves remarkable accuracy 90.7% on training and 92% on test datasets, surpassing the base-line models (RF, SAE) compared in this study. The efficient training time, manageable memory footprint, and robust AUC position the model as a promising solution for multi-label network classification.

The loss of the developed model is also evaluated against the validation and test datasets. During the 100 epochs of training, the system's loss consistently diminishes, settling well below the threshold of 0.02.

The proposed model also shows good performance and efficiency in terms of resource utilization and demonstrates swift learning capabilities and efficiency to classify network traffic and identifies patterns makes it suitable for real-time environments.

5.2. Recommendation

Based on the findings from our experimental results, this study presents the following recommendations and a concise conclusion:

It is important to note that such models need to be deployed in real-world environments to monitor network traffic. Its swift learning capabilities make it suitable for instant classification and resource allocation.

Utilize the model's threat detection capabilities to proactively prevent security breaches. And allocate resources based on traffic behavior to safeguard network integrity. In addition, it is crucial to regularly assess the model's performance against evolving traffic patterns.

Fine-tune parameters as needed to maintain accuracy and adapt to changing network dynamics.

5.3. Future works

In the dynamic landscape of SDN, managing and classifying network traffic efficiently is paramount. The advent of transfer learning, particularly with pretrained models, offers a promising avenue for advancing traffic classification methods in SDN networks.

Pretrained models in transfer learning are akin to seasoned experts who bring a wealth of knowledge from previous experiences. These models have been trained on vast datasets, often in different but related fields, and have developed an ability to identify patterns and features that are not immediately apparent. Pretrained models are suitable for SDN traffic classification as pretrained models can be fine-tuned with SDN-specific data, allowing for rapid adaptation to the unique traffic patterns of a network. And leveraging the nuanced understanding from diverse datasets, pretrained models can offer improved accuracy in classifying traffic types, even with limited SDN data. Utilizing pretrained models can also reduce the need for extensive data collection and computation, leading to cost savings in deploying traffic classification systems.

As we look to the future, the synergy between pretrained transfer learning models and SDN networks holds the potential to revolutionize traffic classification, paving the way for smarter, more secure, and efficient networking solutions.

Future research could explore:

Tailoring pretrained models to better fit the specific characteristics of network traffic data.

Evaluating the performance of various pretrained models from different domains within the context of SDN traffic classification.

6. REFERENCES

[1] J. Jiang, C. Lin, G. Han, A. M. Abu-Mahfouz, S. B. H. Shah, and M. Martínez-García, "How AI-enabled SDN technologies improve the security and functionality of industrial IoT network: Architectures, enabling technologies, and opportunities," *Digit. Commun. Netw.*, p. S2352864822001420, Jul. 2022, doi: 10.1016/j.dcan.2022.07.001.

[2] M. Lotfollahi, R. S. H. Zade, M. J. Siavoshani, and M. Saberian, "Deep Packet: A Novel Approach For Encrypted Traffic Classification Using Deep Learning." arXiv, Jul. 04, 2018. Accessed: Jul. 14, 2023. [Online]. Available: http://arxiv.org/abs/1709.02656

[3] M. Reza, M. Javad, S. Raouf, and R. Javidan, "Network Traffic Classification using Machine Learning Techniques over Software Defined Networks," *Int. J. Adv. Comput. Sci. Appl.*, vol. 8, no. 7, 2017, doi: 10.14569/IJACSA.2017.080729.

[4] A. Malik, R. De Frein, M. Al-Zeyadi, and J. Andreu-Perez, "Intelligent SDN Traffic Classification Using Deep Learning: Deep-SDN," in *2020 2nd International Conference on Computer Communication and the Internet (ICCCI)*, Nagoya, Japan: IEEE, Jun. 2020, pp. 184–189. doi: 10.1109/ICCCI49374.2020.9145971.

[5] N. Rust-Nguyen and M. Stamp, "Darknet Traffic Classification and Adversarial Attacks." arXiv, Jun. 12, 2022. Accessed: Jul. 14, 2023. [Online]. Available: http://arxiv.org/abs/2206.06371

[6] R. Babaria, S. C. Madanapalli, H. Kumar, and V. Sivaraman, "FlowFormers: Transformer-based Models for Real-time Network Flow Classification," in *2021 17th International Conference on Mobility, Sensing and Networking (MSN)*, Exeter, United Kingdom: IEEE, Dec. 2021, pp. 231–238. doi: 10.1109/MSN53354.2021.00046.

[7] A. Dawoud, "Deep Learning: Enhancing the Security of Software-Defined Networks".

[8] M. A. Siddiqi and W. Pak, "Optimizing Filter-Based Feature Selection Method Flow for Intrusion Detection System," *Electronics*, vol. 9, no. 12, p. 2114, Dec. 2020, doi: 10.3390/electronics9122114.

[9] D. Nunez-Agurto, W. Fuertes, L. Marrone, and M. Macas, "Machine Learning-Based Traffic Classification in Software-Defined Networking: A Systematic Literature Review, Challenges, and Future Research Directions," vol. 49, no. 4, 2022.

[10] M. Perera Jayasuriya Kuranage, K. Piamrat, and S. Hamma, "Network Traffic Classification Using Machine Learning for Software Defined Networks," in *Machine Learning for Networking*, vol. 12081, S. Boumerdassi, É. Renault, and P. Mühlethaler, Eds., in Lecture Notes in Computer Science, vol. 12081. , Cham: Springer International Publishing, 2020, pp. 28–39. doi: 10.1007/978-3-030-45778-5_3.

[11] C. Yu, J. Lan, J. Xie, and Y. Hu, "QoS-aware Traffic Classification Architecture Using Machine Learning and Deep Packet Inspection in SDNs," *Procedia Comput. Sci.*, vol. 131, pp. 1209–1216, 2018, doi: 10.1016/j.procs.2018.04.331.

[12] L. Yang, C. Beliard, and D. Rossi, "Heterogeneous Data-Aware Federated Learning." arXiv, Nov. 12, 2020. Accessed: Jul. 14, 2023. [Online]. Available: http://arxiv.org/abs/2011.06393

[13] J. Velasco-Mata, V. González-Castro, E. Fidalgo, and E. Alegre, "Efficient Detection of Botnet Traffic by features selection and Decision Trees." arXiv, Jun. 30, 2021. Accessed: May 16, 2023. [Online]. Available: http://arxiv.org/abs/2107.02896

[14] A. Rasteh, F. Delpech, C. Aguilar-Melchor, R. Zimmer, S. B. Shouraki, and T. Masquelier, "Encrypted Internet traffic classification using a supervised Spiking Neural Network," *Neurocomputing*, vol. 503, pp. 272–282, Sep. 2022, doi: 10.1016/j.neucom.2022.06.055.

[15] J. Velasco-Mata, V. González-Castro, E. Fidalgo, and E. Alegre, "Efficient Detection of Botnet Traffic by features selection and Decision Trees." arXiv, Jun. 30, 2021. Accessed: Jul. 14, 2023. [Online]. Available: http://arxiv.org/abs/2107.02896

[16] O. Belkadi, A. Vulpe, Y. Laaziz, and S. Halunga, "ML-Based Traffic Classification in an SDN-Enabled Cloud Environment," *Electronics*, vol. 12, no. 2, p. 269, Jan. 2023, doi: 10.3390/electronics12020269.

[17] S. Almuhammadi, A. Alnajim, and M. Ayub, "QUIC Network Traffic Classification Using Ensemble Machine Learning Techniques," *Appl. Sci.*, vol. 13, no. 8, p. 4725, Apr. 2023, doi: 10.3390/app13084725.

[18] R. Maarouf, D. Sattar, and A. Matrawy, "Evaluating Resilience of Encrypted Traffic Classification Against Adversarial Evasion Attacks." arXiv, May 30, 2021. Accessed: Jul. 14, 2023. [Online]. Available: http://arxiv.org/abs/2105.14564

[19] A. Yulianto, P. Sukarno, and N. A. Suwastika, "Improving AdaBoost-based Intrusion Detection System (IDS) Performance on CIC IDS 2017 Dataset," *J. Phys. Conf. Ser.*, vol. 1192, p. 012018, Mar. 2019, doi: 10.1088/1742-6596/1192/1/012018.

[20] X. Wang, S. Chen, and J. Su, “Real Network Traffic Collection and Deep Learning for Mobile App Identification,” *Wirel. Commun. Mob. Comput.*, vol. 2020, pp. 1–14, Feb. 2020, doi: 10.1155/2020/4707909.
[21] “Liu and Lang - 2019 - Machine Learning and Deep Learning Methods for Int.pdf.”
[22] P. Casas, “Two Decades of AI4NETS-AI/ML for Data Networks: Challenges & Research Directions.” arXiv, Mar. 02, 2020. Accessed: Jul. 14, 2023. [Online]. Available: http://arxiv.org/abs/2003.04080
[23] Y. Zheng, Z. Dang, C. Peng, C. Yang, and X. Gao, “Multi-view Multi-label Anomaly Network Traffic Classification based on MLP-Mixer Neural Network.” arXiv, Nov. 07, 2022. Accessed: Jul. 14, 2023. [Online]. Available: http://arxiv.org/abs/2210.16719
[24] R.-H. Hwang, M.-C. Peng, V.-L. Nguyen, and Y.-L. Chang, “An LSTM-Based Deep Learning Approach for Classifying Malicious Traffic at the Packet Level,” *Appl. Sci.*, vol. 9, no. 16, p. 3414, Aug. 2019, doi: 10.3390/app9163414.
[25] X. Wang, S. Chen, and J. Su, “Real Network Traffic Collection and Deep Learning for Mobile App Identification,” *Wirel. Commun. Mob. Comput.*, vol. 2020, pp. 1–14, Feb. 2020, doi: 10.1155/2020/4707909.
[26] L.-H. Chang, Tsung-Han Lee, Hung-Chi Chu, and Cheng-Wei Su, “Application-Based Online Traffic Classification with Deep Learning Models on SDN Networks,” *Adv. Technol. Innov.*, Jul. 2020, doi: 10.46604/aiti.2020.4286.
[27] X. Meng, C. Lin, Y. Wang, and Y. Zhang, “NetGPT: Generative Pretrained Transformer for Network Traffic.” arXiv, May 17, 2023. Accessed: Jul. 14, 2023. [Online]. Available: http://arxiv.org/abs/2304.09513
[28] A. Lichy, O. Bader, R. Dubin, A. Dvir, and C. Hajaj, “When a RF Beats a CNN and GRU, Together -- A Comparison of Deep Learning and Classical Machine Learning Approaches for Encrypted Malware Traffic Classification.” arXiv, Jun. 16, 2022. Accessed: Jul. 14, 2023. [Online]. Available: http://arxiv.org/abs/2206.08004
[29] S. Bano, A. Machumilane, L. Valerio, P. Cassarà, and A. Gotta, “Federated Semi-Supervised Classification of Multimedia Flows for 3D Networks.” arXiv, May 01, 2022. Accessed: Jul. 14, 2023. [Online]. Available: http://arxiv.org/abs/2205.00550
[30] O. Aouedi, K. Piamrat, and B. Parrein, “Intelligent Traffic Management in Next-Generation Networks,” *Future Internet*, vol. 14, no. 2, p. 44, Jan. 2022, doi: 10.3390/fi14020044.
[31] Y. Zeng, Z. Qi, W. Chen, Y. Huang, X. Zheng, and H. Qiu, “TEST: an End-to-End Network Traffic Examination and Identification Framework Based on Spatio-Temporal Features Extraction.” arXiv, Aug. 26, 2019. Accessed: Jul. 14, 2023. [Online]. Available: http://arxiv.org/abs/1908.10271
[32] M. M. Raikar, M. S M, M. M. Mulla, N. S. Shetti, and M. Karanandi, “Data Traffic Classification in Software Defined Networks (SDN) using supervised-learning,” *Procedia Comput. Sci.*, vol. 171, pp. 2750–2759, 2020, doi: 10.1016/j.procs.2020.04.299.
[33] H. A. H. Ibrahim, O. R. A. L. Zuobi, A. M. Abaker, and M. B. Alzghoul, “A Hybrid Online Classifier System for Internet Traffic Based on Statistical Machine Learning Approach and Flow Port Number,” *Appl. Sci.*, vol. 11, no. 24, p. 12113, Dec. 2021, doi: 10.3390/app112412113.
[34] N. Ahuja, G. Singal, and D. Mukhopadhyay, “DLSDN: Deep Learning for DDOS attack detection in Software Defined Networking,” in *2021 11th International Conference on Cloud Computing, Data Science & Engineering (Confluence)*, Noida, India: IEEE, Jan. 2021, pp. 683–688. doi: 10.1109/Confluence51648.2021.9376879.
[35] H. Polat, M. Turkoglu, and O. Polat, “Deep network approach with stacked sparse autoencoders in detection of DDoS attacks on SDN-based VANET,” *IET Commun.*, vol. 14, no. 22, pp. 4089–4100, Dec. 2020, doi: 10.1049/iet-com.2020.0477.
[36] S. Madanapalli, “Enhancing User Experience by Extracting Application Intelligence from Network Traffic,” UNSW Sydney, 2022. doi: 10.26190/UNSWORKS/24314.
[37] J. Ren, Y. Zhang, Z. Wang, Y. Song, and Northeast Branch of State Gird Corporation of China, #1, Yingpan North Street, Shenyang, Liaoning, MO 110180, China, “Artificial intelligence-based network traffic analysis and automatic optimization technology,” *Math. Biosci. Eng.*, vol. 19, no. 2, pp. 1775–1785, 2021, doi: 10.3934/mbe.2022083.
[38] I. Guarino, C. Wang, A. Finamore, A. Pescape, and D. Rossi, “Many or Few Samples? Comparing Transfer, Contrastive and Meta-Learning in Encrypted Traffic Classification.” arXiv, Jun. 03, 2023. Accessed: Jul. 14, 2023. [Online]. Available: http://arxiv.org/abs/2305.12432
[39] M. H. Pathmaperuma, Y. Rahulamathavan, S. Dogan, and A. M. Kondoz, “Deep Learning for Encrypted Traffic Classification and Unknown Data Detection,” *Sensors*, vol. 22, no. 19, p. 7643, Oct. 2022, doi: 10.3390/s22197643.

[40] M. Abbasi, A. Shahraki, and A. Taherkordi, “Deep Learning for Network Traffic Monitoring and Analysis (NTMA): A Survey,” *Comput. Commun.*, vol. 170, pp. 19–41, Mar. 2021, doi: 10.1016/j.comcom.2021.01.021.
[41] N. Ahmed *et al.*, “Network Threat Detection Using Machine/Deep Learning in SDN-Based Platforms: A Comprehensive Analysis of State-of-the-Art Solutions, Discussion, Challenges, and Future Research Direction,” *Sensors*, vol. 22, no. 20, p. 7896, Oct. 2022, doi: 10.3390/s22207896.
[42] H. A. H. Ibrahim, O. R. A. L. Zuobi, A. M. Abaker, and M. B. Alzghoul, “A Hybrid Online Classifier System for Internet Traffic Based on Statistical Machine Learning Approach and Flow Port Number,” *Appl. Sci.*, vol. 11, no. 24, p. 12113, Dec. 2021, doi: 10.3390/app112412113.
[43] A. P. Veiga, “Applications of Artificial Intelligence (AI) to Network Security”.
[44] M. Ring, S. Wunderlich, D. Scheuring, D. Landes, and A. Hotho, “A Survey of Network-based Intrusion Detection Data Sets,” *Comput. Secur.*, vol. 86, pp. 147–167, Sep. 2019, doi: 10.1016/j.cose.2019.06.005.
[45] G. Marín, P. Casas, and G. Capdehourat, “DeepMAL -- Deep Learning Models for Malware Traffic Detection and Classification.” arXiv, Mar. 10, 2020. Accessed: Jul. 14, 2023. [Online]. Available: http://arxiv.org/abs/2003.04079
[46] K. Prasad, S. Ghosh, G. Cormode, I. Mironov, A. Yousefpour, and P. Stock, “Reconciling Security and Communication Efficiency in Federated Learning.” arXiv, Jul. 26, 2022. Accessed: Jul. 14, 2023. [Online]. Available: http://arxiv.org/abs/2207.12779
[47] T. Ongun, T. Sakharaov, S. Boboila, A. Oprea, and T. Eliassi-Rad, “On Designing Machine Learning Models for Malicious Network Traffic Classification.” arXiv, Jul. 10, 2019. Accessed: Jul. 14, 2023. [Online]. Available: http://arxiv.org/abs/1907.04846
[48] M. V. Kiran and N. B, “Network Status Aware Congestion Control (NSACC) Algorithm for Wireless Body Area Network,” *Procedia Comput. Sci.*, vol. 171, pp. 42–51, 2020, doi: 10.1016/j.procs.2020.04.005.
[49] K. M. Sivalingam, “Applications of Artificial Intelligence, Machine Learning and related techniques for Computer Networking Systems.” arXiv, Apr. 21, 2021. Accessed: Jul. 14, 2023. [Online]. Available: http://arxiv.org/abs/2105.15103
[50] H. Dhillon and A. Haque, “Towards Network Traffic Monitoring Using Deep Transfer Learning,” in *2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, Guangzhou, China: IEEE, Dec. 2020, pp. 1089–1096. doi: 10.1109/TrustCom50675.2020.00144.
[51] M. Zhang, W. Sun, J. Tian, X. Zheng, and S. Guan, “An internet traffic classification method based on echo state network and improved salp swarm algorithm,” *PeerJ Comput. Sci.*, vol. 8, p. e860, Feb. 2022, doi: 10.7717/peerj-cs.860.
[52] M. Abbasi, A. Shahraki, and A. Taherkordi, “Deep Learning for Network Traffic Monitoring and Analysis (NTMA): A Survey,” *Comput. Commun.*, vol. 170, pp. 19–41, Mar. 2021, doi: 10.1016/j.comcom.2021.01.021.
[53] J. Ren, Y. Zhang, Z. Wang, Y. Song, and Northeast Branch of State Gird Corporation of China, #1, Yingpan North Street, Shenyang, Liaoning, MO 110180, China, “Artificial intelligence-based network traffic analysis and automatic optimization technology,” *Math. Biosci. Eng.*, vol. 19, no. 2, pp. 1775–1785, 2021, doi: 10.3934/mbe.2022083.
[54] P. Singhal, R. Mathur, and H. Vyas, “State of the Art Review of Network Traffic Classification based on Machine Learning Approach,” *Int. J. Comput. Appl.*.
[55] K. Qiu *et al.*, “Traffic Analytics Development Kits (TADK): Enable Real-Time AI Inference in Networking Apps,” in *2022 Thirteenth International Conference on Ubiquitous and Future Networks (ICUFN)*, Jul. 2022, pp. 392–398. doi: 10.1109/ICUFN55119.2022.9829628.
[56] P. Singhal, R. Mathur, and H. Vyas, “State of the Art Review of Network Traffic Classification based on Machine Learning Approach,” *Int. J. Comput. Appl.*.
[57] H. D. Trinh, A. F. Gambin, L. Giupponi, M. Rossi, and P. Dini, “Mobile Traffic Classification through Physical Channel Fingerprinting: a Deep Learning Approach.” arXiv, Feb. 07, 2020. Accessed: Jul. 14, 2023. [Online]. Available: http://arxiv.org/abs/1910.11617
[58] J. H. Ring, C. M. Van Oort, S. Durst, V. White, J. P. Near, and C. Skalka, “Methods for Host-based Intrusion Detection with Deep Learning,” *Digit. Threats Res. Pract.*, vol. 2, no. 4, pp. 1–29, Dec. 2021, doi: 10.1145/3461462.
[59] J. Pešek, D. Soukup, and T. Čejka, “Active Learning Framework to Automate NetworkTraffic Classification.” arXiv, Oct. 26, 2022. Accessed: Jul. 14, 2023. [Online]. Available: http://arxiv.org/abs/2211.08399

[60] A. Parchekani, S. Nouri, V. Shah-Mansouri, and S. P. Shariatpanahi, “Classification of Traffic Using Neural Networks by Rejecting: a Novel Approach in Classifying VPN Traffic.” arXiv, Dec. 10, 2021. Accessed: Jul. 14, 2023. [Online]. Available: http://arxiv.org/abs/2001.03665

[61] O. Aouedi, K. Piamrat, and D. Bagadthey, “A Semi-supervised Stacked Autoencoder Approach for Network Traffic Classification,” in *2020 IEEE 28th International Conference on Network Protocols (ICNP)*, Madrid, Spain: IEEE, Oct. 2020, pp. 1–6. doi: 10.1109/ICNP49622.2020.9259390.

[62] G. Bovenzi *et al.*, “A First Look at Class Incremental Learning in Deep Learning Mobile Traffic Classification.” arXiv, Jul. 09, 2021. Accessed: Jul. 14, 2023. [Online]. Available: http://arxiv.org/abs/2107.04464

[63] M. S. Elsayed, N.-A. Le-Khac, and A. D. Jurcut, “InSDN: A Novel SDN Intrusion Dataset,” *IEEE Access*, vol. 8, pp. 165263–165284, 2020, doi: 10.1109/ACCESS.2020.3022633.

[64] A. Malik, R. De Frein, M. Al-Zeyadi, and J. Andreu-Perez, “Intelligent SDN Traffic Classification Using Deep Learning: Deep-SDN,” in *2020 2nd International Conference on Computer Communication and the Internet (ICCCI)*, Nagoya, Japan: IEEE, Jun. 2020, pp. 184–189. doi: 10.1109/ICCCI49374.2020.9145971.

[65] Z. Fan and R. Liu, “Investigation of machine learning based network traffic classification,” in *2017 International Symposium on Wireless Communication Systems (ISWCS)*, Bologna: IEEE, Aug. 2017, pp. 1–6. doi: 10.1109/ISWCS.2017.8108090.

[66] J. Chen, J. Breen, J. M. Phillips, and J. Van der Merwe, “Practical and Configurable Network Traffic Classification Using Probabilistic Machine Learning,” *Clust. Comput.*, vol. 25, no. 4, pp. 2839–2853, Aug. 2022, doi: 10.1007/s10586-021-03393-2.

[67] M. Lotfollahi, R. S. H. Zade, M. J. Siavoshani, and M. Saberian, “Deep Packet: A Novel Approach For Encrypted Traffic Classification Using Deep Learning.” arXiv, Jul. 04, 2018. Accessed: May 16, 2023. [Online]. Available: http://arxiv.org/abs/1709.02656

[68] B. Pang, Y. Fu, S. Ren, Y. Wang, Q. Liao, and Y. Jia, “CGNN: Traffic Classification with Graph Neural Network.” arXiv, Oct. 19, 2021. Accessed: Jul. 14, 2023. [Online]. Available: http://arxiv.org/abs/2110.09726

[69] Z. Wang, Y. Dong, S. Mao, and X. Wang, “Internet multimedia traffic classification from QoS perspective using semi-supervised dictionary learning models,” *China Commun.*, vol. 14, no. 10, pp. 202–218, Oct. 2017, doi: 10.1109/CC.2017.8107644.

[70] H. Liu and B. Lang, “Machine Learning and Deep Learning Methods for Intrusion Detection Systems: A Survey,” *Appl. Sci.*, vol. 9, no. 20, p. 4396, Oct. 2019, doi: 10.3390/app9204396.

[71] A. Adamou Djergou, Y. Maleh, and S. Mounir, “Machine Learning Techniques for Intrusion Detection in SDN: A Survey,” in *Advances in Information, Communication and Cybersecurity*, vol. 357, Y. Maleh, M. Alazab, N. Gherabi, L. Tawalbeh, and A. A. Abd El-Latif, Eds., in Lecture Notes in Networks and Systems, vol. 357. , Cham: Springer International Publishing, 2022, pp. 460–473. doi: 10.1007/978-3-030-91738-8_42.

[72] S. Soleymanpour, H. Sadr, and H. Beheshti, “An Efficient Deep Learning Method for Encrypted Traffic Classification on the Web,” in *2020 6th International Conference on Web Research (ICWR)*, Tehran, Iran: IEEE, Apr. 2020, pp. 209–216. doi: 10.1109/ICWR49608.2020.9122299.

7. APPENDECES

Appendix A:

The proposed model:

```
# General utilities
import time
import logging
import pickle
# Data handling
import numpy as np
import pandas as pd
# Data visualization
from matplotlib import pyplot as plt
import matplotlib.pyplot as plt
import seaborn as sns
# Machine Learning
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, OneHotEncoder, StandardScaler, MinMaxScaler
from sklearn.metrics import roc_curve, auc, roc_auc_score,accuracy_score
from sklearn.metrics import precision_score, recall_score, f1_score, confusion_matrix
from sklearn.metrics import multilabel_confusion_matrix, classification_report
from sklearn.feature_selection import SelectKBest, chi2, RFE, mutual_info_classif
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.inspection import permutation_importance, plot_partial_dependence
from sklearn.preprocessing import label_binarize
```

```
# Deep Learning
import tensorflow as tf
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential, load_model, clone_model
from tensorflow.keras.layers import Dense, Flatten, Dropout, Conv2D, MaxPooling2D, BatchNormalization
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from tensorflow.keras.applications import ResNet50, DenseNet201
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras import backend as K
from tensorflow.keras.regularizers import l2
# Compatibility with older versions of Keras
import np_utils
# from sklearn.metrics import roc_curve, auc
from itertools import cycle
# System monitoring
import psutil
# File handling
import h5py
# Graph visualization
import graphviz
def recall_m(y_true, y_pred):
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
    recall = true_positives / (possible_positives + K.epsilon())
```

```
        return recall
    def precision_m(y_true, y_pred):
        true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
        predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
        precision = true_positives / (predicted_positives + K.epsilon())
        return precision
    def f1_m(y_true, y_pred):
        precision = precision_m(y_true, y_pred)
        recall = recall_m(y_true, y_pred)
        return 2 * ((precision * recall) / (precision + recall + K.epsilon()))
sdn_train_ds=pd.read_csv("D:/SDN-ntc-DL/Unicauca-dataset-April-June-2019-Network-flows.csv")
    # sdn_train = pd.read_csv("C:/Users/betemic/Desktop/SDN-ntc-DL/Darknet.csv")
    sdn_train_ds.head()
    # Check for missing values
    print(sdn_train_ds.isnull().sum())
    # Check for inconsistent values
    print(sdn_train_ds['label'].value_counts())
    # Check for duplicate rows
    print(sdn_train_ds.duplicated().sum())
    # Remove duplicate rows
    sdn_train = sdn_train_ds.drop_duplicates()
    sdn_train.columns
    #Encode categorical features
    ohe = OneHotEncoder(sparse=False)
    protocol_ohe = ohe.fit_transform(sdn_train['protocol'].values.reshape(-1, 1))
```

```
protocol_cols = ohe.get_feature_names_out(['protocol'])
protocol_df = pd.DataFrame(protocol_ohe, columns=protocol_cols)
# Concatenate the one-hot encoded protocol columns to the original dataframe
sdn_train = pd.concat([sdn_train, protocol_df], axis=1)
# Drop the original protocol column
sdn_train = sdn_train.drop('protocol', axis=1)
sdn_train = sdn_train.replace([np.inf, -np.inf], np.finfo('float64').max)
sdn_train.replace([np.inf, -np.inf], np.nan, inplace=True)
# Drop rows with NaN
sdn_train.dropna(inplace=True)
print(sdn_train)
with pd.option_context('mode.use_inf_as_na', True):
  sdn_train.dropna(inplace=True)
print(sdn_train)
train_features = sdn_train.copy()
print(train_features.head)
train_labels = train_features.pop('label')
print(train_labels)
print(sdn_train['label'].value_counts())
print(protocol_df)
print(sdn_train['label'].unique())
#Visulize the dataset
sdn_train.shape
sdn_train['label'].unique()
sdn_train['label'].value_counts().plot.bar()
traffic_class=pd.Categorical(sdn_train['label'])
```

```
traffic_class
sdn_train['label'].value_counts()
sdn_train.iloc[:,1:].describe().transpose()
sns.heatmap(train_features.corr(),cmap='coolwarm')
print(sdn_train.shape)
X = train_features
y = train_labels
#Feature selection and Engineering
# 'X' is the feature matrix and 'y' is the target vector
# Train the model
model = RandomForestClassifier()
model.fit(x, y)
# Get feature importances
importances = model.feature_importances_
# Sort the feature importances in ascending order and get the index of the least important
indices_to_remove = importances.argsort()[:1]
# Drop the least important features from the dataset
X = np.delete(x, indices_to_remove, axis=1)
from sklearn.preprocessing import LabelEncoder
label_enc = LabelEncoder()
label_enc.fit(y)
y = label_enc.fit_transform(y)
# Set the number of classes in the network traffic dataset
classes=label_enc.classes_
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.15,random_state=42)
```

```
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.15, stratify=y_train, random_state=42)
#Regularization
    from imblearn.over_sampling import SMOTE
    smote = SMOTE()
    X_train, y_train = smote.fit_resample(X_train, y_train)
    X_test, y_test = smote.fit_resample(X_test, y_test)
    X_val, y_val = smote.fit_resample(X_val, y_val)
    # Convert the labels to one-hot vectors
    num_classes = len(np.unique(y_train))
    y_train = tf.keras.utils.to_categorical(y_train, num_classes)
    y_test = tf.keras.utils.to_categorical(y_test, num_classes)
    y_val = tf.keras.utils.to_categorical(y_val, num_classes)
    #Scaling
    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test)
    X_val = scaler.transform(X_val)
    #Normalization of features
    X_train = (X_train - np.mean(X_train)) / np.std(X_train)
    X_test = (X_test - np.mean(X_test)) / np.std(X_test)
    X_val = (X_val - np.mean(X_val)) / np.std(X_val)
    # Scale data to range of 0-1
    scaler = MinMaxScaler()
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test)
```

```
X_val = scaler.transform(X_val)
# Reshape the features to match the input shape
input_shape = (X_train.shape[1],)
X_train = X_train.reshape(-1, *input_shape).astype("float32") / 255.0
X_test = X_test.reshape(-1, *input_shape).astype("float32") / 255.0
X_val = X_val.reshape(-1, *input_shape).astype("float32") / 255.0
# Create a batch normalization layer using TensorFlow
batch_norm = BatchNormalization()
X_train = batch_norm(X_train)
X_test = batch_norm(X_test)
X_val = batch_norm(X_val)
early_stop = EarlyStopping(monitor='val_loss', min_delta=0.01, patience=10, mode='min')
model = tf.keras.models.Sequential([
    tf.keras.layers.Dense(512, activation='relu', input_dim=208),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dropout(0.1),
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dropout(0.1),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dropout(0.1),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dropout(0.1),
```

```
        tf.keras.layers.Dense(32, activation='relu'),
        tf.keras.layers.BatchNormalization(),
        tf.keras.layers.Dropout(0.1),
        tf.keras.layers.Dense(16, activation='relu'),
        tf.keras.layers.BatchNormalization(),
        tf.keras.layers.Dropout(0.1),
        tf.keras.layers.Dense(24, activation='sigmoid')
    ])
    model.compile(optimizer=tf.compat.v1.train.AdamOptimizer(learning_rate=0.001),
                  loss='binary_crossentropy',
                  metrics=['acc', f1_m, precision_m, recall_m])
    start_time = time.time()
    start_cpu = psutil.cpu_percent()
    start_memory = psutil.virtual_memory().percent
history=model.fit(X_train,y_train,batch_size=128,epochs=100,
validation_data=(X_test,                y_test),callbacks=([early_stop]))
    # Record the end time and the final CPU and memory usage of the training process
    end_time = time.time()
    end_cpu = psutil.cpu_percent()
    end_memory = psutil.virtual_memory().percent
    # Calculate and print the training time and the CPU and memory utilization
    training_time = end_time - start_time
    cpu_utilization = end_cpu - start_cpu
    memory_utilization = end_memory - start_memory
    print(f"Training time: {training_time} seconds")
    print(f"CPU utilization: {cpu_utilization} %")
```

```
print(f"Memory utilization: {memory_utilization} %")
# Plot training Accuracy graph
plt.plot(history.history['acc'])
plt.title('model-accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train'], loc='upper left')
plt.show()
# Plot training Loss graph
plt.plot(history.history['loss'])
plt.title('model-loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['loss'], loc='upper right')
plt.show()
# Plot training Validation-accuracy graph
plt.plot(history.history['val_acc'])
plt.title('model-val_accuracy')
plt.ylabel('val_accuracy')
plt.xlabel('epoch')
plt.legend(['val_accuracy'], loc='upper left')
plt.show()
# Plot training Validation Loss graph
plt.plot(history.history['val_loss'])
plt.title('model-val_loss')
plt.ylabel('val_loss')
```

```
plt.xlabel('epoch')
plt.legend(['val_loss'], loc='upper right')
plt.show()
loss, accuracy,f1_score, precision, recall = model.evaluate(X_test, np.array(y_test), verbose=0)
print(loss, accuracy,f1_score, precision, recall)
# Prediction
#y_test contains the true labels and y_score contains the predicted probabilities
# y_score = # predicted probabilities from the deep neural network
# y_test = # true class labels
y_score = model.predict(X_test)
for i in y_score:
    print(np.argmax(i))
print(f'loss: {loss}, acc: {accuracy}, f1_score: {f1_score}, precision: {precision}, recall: {recall}')
print(model.summary())
# Binarize the output classes
n_classes = y_test.shape[1]
y_test_binarized = label_binarize(y_test, classes=[*range(n_classes)])
# Compute ROC curve and AUC for each class
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_binarized[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])
# Plot ROC curves for each class
```

```
plt.figure(figsize=(12, 8))
# Subplot 1: ROC Curves
# Add enough colours for 24 classes
plt.subplot(1, 2, 1)
# colors = cycle(['blue', 'green', 'red', 'cyan', 'magenta', 'yellow', 'black'])
# Define 24 unique colors for the classes
colors = cycle([
    'navy', 'turquoise', 'darkorange', 'cornflowerblue', 'teal',
    'red', 'yellow', 'purple', 'green', 'blue', 'black', 'pink',
    'brown', 'grey', 'violet', 'magenta', 'gold', 'lime', 'cyan',
    'olive', 'maroon', 'orchid', 'tomato', 'sienna'
])
for i, color in zip(range(n_classes), colors):
    plt.plot(fpr[i], tpr[i], color=color, lw=2,
             label='Class {0} (AUC = {1:0.2f})'.format(i, roc_auc[i]))
            # label=category_class[i]).format(i, roc_auc[i])
plt.plot([0, 1], [0, 1], 'k--', lw=2)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")
# plt.show()
# Subplot 2: Probability Distributions
category_class = ['Chat', 'Cloud', 'Collaborative', 'DataTransfer', 'Database',
```

```
        'Download-FileTransfer-FileSharing', 'Email', 'Game', 'Media',
        'Mining', 'Music', 'Network', 'RPC', 'RemoteAccess', 'Shopping',
        'SocialNetwork', 'SoftwareUpdate', 'Streaming', 'System',
        'Unspecified', 'VPN', 'Video', 'VoIP', 'Web']
# Number of classes
n_classes = len(category_class)
plt.subplot(1, 2, 2)
for i, color in zip(range(n_classes), colors):
    plt.hist(y_score[:, i], bins=10, color=color, alpha=0.3,
             # label='Class {0}'.format(i))
             label=category_class[i])
plt.title('Predicted Probability Distribution')
plt.xlabel('Probability')
plt.ylabel('Frequency')
plt.legend(loc="upper right")
plt.tight_layout()
plt.show()
#Make over all predictions
y_pred = model.predict(X_train)
y_pred_classes = np.argmax(y_pred, axis=1)
# Calculate over all accuracy
accuracy = accuracy_score(np.argmax(y_train, axis=1), y_pred_classes)
print(f"Accuracy: {accuracy:.2f}")
# confusion matrix
conf_matrix = confusion_matrix(np.argmax(y_train, axis=1), y_pred_classes)
# Plot confusion matrix
```

```
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.title("Confusion Matrix")
plt.show()
```

Appendix B:

```
# Building SAE model
input_layer = tf.keras.layers.Input(shape=(X_train.shape[1],))
hidden_layer_1 = tf.keras.layers.Dense(128, activation='relu')(input_layer)
hidden_layer_2 = tf.keras.layers.Dense(64, activation='relu')(hidden_layer_1)
hidden_layer_3 = tf.keras.layers.Dense(32, activation='relu')(hidden_layer_2)
hidden_layer_4 = tf.keras.layers.Dense(16, activation='relu')(hidden_layer_3)
encoded_layer = tf.keras.layers.Dense(8, activation='relu')(hidden_layer_4)
hidden_layer_6 = tf.keras.layers.Dense(16, activation='relu')(encoded_layer)
hidden_layer_7 = tf.keras.layers.Dense(32, activation='relu')(hidden_layer_6)
hidden_layer_8 = tf.keras.layers.Dense(64, activation='relu')(hidden_layer_7)
hidden_layer_9 = tf.keras.layers.Dense(128, activation='relu')(hidden_layer_8)
output_layer=tf.keras.layers.Dense(y_train.shape[1],activation='sigmoid')(hidden_lay
er_9)
model = tf.keras.Model(inputs=input_layer, outputs=output_layer)
model.compile(optimizer=tf.compat.v1.train.AdamOptimizer(learning_rate=0.001),
        loss='binary_crossentropy',
        metrics=['acc', f1_m, precision_m, recall_m])
tensorboard = tf.keras.callbacks.TensorBoard(log_dir="logs/".format(file_Name))
start_time = time.time()
start_cpu = psutil.cpu_percent()
history=model.fit(X_train,y_train,batch_size=128,epochs=100,
validation_data=(X_al, y_val))
# Record the end time and the final CPU and memory usage of the training process
end_time = time.time()
end_cpu = psutil.cpu_percent()
```

```
end_memory = psutil.virtual_memory().percent
# Calculate and print the training time and the CPU and memory utilization
training_time = end_time - start_time
cpu_utilization = end_cpu - start_cpu
memory_utilization = end_memory - start_memory
print(f"Training time: {training_time} seconds")
print(f"CPU utilization: {cpu_utilization} %")
print(f"Memory utilization: {memory_utilization} %")
```

Appendix C:

```
# Building RF model
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score
# Assuming X_train, y_train, X_test, y_test is already defined in the previous models
# model = RandomForestClassifier(n_estimators=24, random_state=42)
# model.fit(X_train, y_train)
Build Random Forest model
rfc = RandomForestClassifier()
rfc.fit(X_train, y_train)
# Feature Importances
feature_importances = rfc.feature_importances_
indices = np.argsort(feature_importances)[::-1]
plt.figure(figsize=(12, 6))
plt.title('Feature Importances')
plt.bar(range(X_train.shape[1]), feature_importances[indices], align='center')
plt.xticks(range(X_train.shape[1]), sdn_train.columns[indices], rotation=90)
plt.tight_layout()
plt.show()
# Make predictions
predictions = rfc.predict(X_test)
# Evaluate the model
accuracy = accuracy_score(y_test, predictions)
f1 = f1_score(y_test, predictions, average='macro')
precision = precision_score(y_test, predictions, average='macro')
recall = recall_score(y_test, predictions, average='macro')
```

```
print(f'Accuracy: {accuracy}')
print(f'F1 Score: {f1}')
print(f'Precision: {precision}')
print(f'Recall: {recall}')
```