PAYLOAD BASED MULTI-PHASE TRAFFIC CLASSIFICATION WITH MAJORITY VOTING

by

Ilhan Selcuk Mert

B.S., Electrical and Electronics Engineering, Boğaziçi University, 2018

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Electrical and Electronics Engineering
Boğaziçi University
2022

ACKNOWLEDGEMENTS

I would first like to thank my thesis advisor Prof. Emin Anarım for his guidance and encouragement.

Next, I would like to thank my family, especially my wife, for their love and endless support. I wish to thank my friends who have helped and inspired me during my thesis study.

Lastly, I would like to thank my employer and colleagues for their support and allowing me to proceed with my education.

This work is supported by the Boğaziçi University Scientific Research Projects under the Spectral and Entropy Approaches in Packet Classification, 18281.

ABSTRACT

PAYLOAD BASED MULTI-PHASE TRAFFIC CLASSIFICATION WITH MAJORITY VOTING

Internet is becoming an essential part of our lives with even simple daily tasks depending on it. This led to an increase in network traffic accompanied with increase in number of applications hosted on internet. In this heavy traffic environment, classifying network flows in a fast and accurate manner, has great importance for network management. Internet Service Providers try to address this issue by using different approaches from port-based methods to machine learning models but due to widespread usage of dynamic ports and encrypted packets by modern applications, accuracy of these approaches declined. To overcome this challenge, recent studies focus on solutions using deep learning architectures.

In this thesis, a multi-phase classification model based on voting and deep learning is proposed for encrypted traffic classification. The proposed model relies on the payload of the transmitted packets to classify flows. In this approach, deep learning based classifiers are trained using different numbers of packets from flows as input and the prediction of multi-phase model is an ensemble of these classifiers calculated by different voting strategies. This approach enables classification of flows starting from the first transmitted packet with payload, and updates the predicted class as the number of transmitted packets in flow increases. This approach has been tested on datasets containing real network flows from various applications. The performance of proposed approach is evaluated by comparing different classification models and different voting strategies.

ÖZET

ÇOĞUNLUK OYLAMASI YARDIMI İLE ÇOK AŞAMALI TRAFİK SINIFLANDIRMA

Günümüzde internet giderek hayatımızın temel parçalarından biri olmakta, ve en basit günlük işlerimiz bile internete bağımlı durumda bulunmaktadır. Bu durum, hem internet üzerindeki uygulama sayısının hem de ağ trafiğinin artmasına neden olmaktadır. Bu yoğun trafik ortamında, ağ akışlarının hızlı ve doğru bir şekilde sınıflandırıması ağ yönetimi açısından büyük bir önem taşımaktadır. İnternet Servis Sağlayıcıları, bağlantı noktası tabanlı yöntemlerden makine öğrenme modellerine kadar farklı yaklaşımlar kullanarak bu sorunu çözmeye çalışmaktadır. Modern uygulamaların dinamik bağlantı noktalarını ve şifreli paketleri yaygın olarak kullanması nedeniyle bu yaklaşımların doğruluğu azalmaktadır. Bu zorluğun üstesinden gelmek için, son çalışmalar derin öğrenme mimarilerini kullanan çözümler üzerine odaklanmaktadır.

Bu tezde, şifreli trafik sınıflandırması için oylamaya ve derin öğrenmeye dayalı çok aşamalı bir sınıflandırma yordamı önerilmiştir. Önerilen yordam, akışları sınıflandırmak için iletilen paketlerin sahip olduğu yükü kullanmaktadır. Bu yaklaşım ile, derin öğrenme tabanlı sınıflandırıcılar, akışlardan farklı sayıda paketler kullanılarak eğitilir ve çok aşamalı modelin tahmini, bu sınıflandırıcıların çıktılarının farklı oylama stratejileri ile birleştirilmesi ile elde edilir. Bu yordam, yük içeren ilk paketten başlayarak akışları sınıflandırabilir ve akışta iletilen paket sayısı arttıkça tahmin edilen sınıfı günceller. Bu yordam, çeşitli uygulamalar için gerçek ağ akışlarını içeren veri kümeleri üzerinde test edilmiştir. Önerilen yordamın performansı, farklı sınıflandırma modelleri ve farklı oylama stratejileri ile karşılaştırılarak değerlendirilmiştir.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

c_t	Cell State
$\check{\mathbf{C}}_\mathbf{t}$	Cell Activation Vector
f_t	Output of Forget Gate
$\mathbf{h_t}$	Hidden State Vector
i_t	Output of Input Gate
n	Payload Length
o_t	Output of Output Gate
$\mathbf{P_i}$	Payload Vector of Packet i
$tf - idf_{t,p}$	Term Frequency-Inverse Document Frequency
$\mathbf{w}$	Weights of Neuron
$\mathbf{x}$	Input signal
y	Output Signal
$\phi\,(.)$	Activation Function

LIST OF ACRONYMS/ABBREVIATIONS

CBOW	Continuous Bag of Words
CNN	Convolutional Neural Network
DPI	Deep Packet Inspection
FNN	Feedforward Neural Network
GRU	Gated Recurrent Unit
ISP	Internet Service Provider
K-NN	K-Nearest Neighbor
LSTM	Long Short-term Memory
MLP	Multilayer Perceptron
PIAT	Packet Inter-Arrival Times
ReLU	Rectified Linear Unit
ResNet	Residual Network
RNN	Recurrent Neural Network
SVM	Support Vector Machine
SAE	Stacked Autoencoder
TF-IDF	Term Frequency-Inverse Document Frequency

1. INTRODUCTION

Network Traffic Classification is a task in network management dealing with the classification of network flows according to the different properties. A network flow consists of transmitted packets between two hosts in a given time interval. Typically, traffic classification is employed by internet service providers (ISPs) for traffic routing, Quality of Service management or policy enforcement (e.g., blocking packets coming from certain applications or logging). Different classes are processed with different policies by ISPs ensuring that users get the best service.

To classify network flows, different methods are employed. We can divide these methods into three as follows: port-based methods, Deep Packet Inspection (DPI) methods and statistical methods.

Port-based methods, as the name implies relies on the port information to classify flows. This is the simplest and the fastest classification method which was widely employed by ISPs in the past. It usually relies on a large look-up table containing known addresses of applications. Port-based methods are easy to implement and consumes very low resources but nowadays many applications use dynamic ports which decreases the accuracy of port-based methods making them obsolete.

Statistical methods use the statistical features extracted from flows. These features can be number of transmitted packets, mean packet size, packet inter arrival times, byte frequencies etc. Usually, these features are used to train a machine learning model (such as Decision Tree, K-NN, K-means etc.) to classify flows. These methods are slower compared to port-based methods, but they have greater accuracy.

DPI is an umbrella term including all methods which uses the payload of network packets. In DPI methods, network flows are classified by examining the payload of their packets. DPI methods are more accurate with the cost of being slower compared the previous two methods mentioned above and they require more processing power.

Signatures are extracted from payload to represent applications. These extracted signatures are used to create a signature database and packets are classified by comparing their payload signatures with the database. As mentioned above this method is more advanced compared to previous methods enabling more accurate classifications, but the drawback of this method is, encrypted traffic hinders signature comparison. To overcome this drawback recent studies focus on deep learning based approaches.

1.1. Motivation

Over the years, the number of applications and the transmission speed in internet increased steadily and most of the applications are using encrypted packets. These developments pose a challenge in terms of network management. Network managers need to classify network flows in a fast and accurate manner to ensure best service for their users. The legacy port-based and statistical methods are insufficient to handle today's network traffic. Although port-based methods work very fast, many new applications use dynamic ports which lowers port-based classifiers accuracy significantly. On the other hand, statistical methods are accurate but slow, they require sufficient amount of packets to be transmitted so that the calculated features are meaningful. In this thesis, a payload based multi-phase model is proposed for traffic classification. The proposed model starts to classify network flows starting with the first packet containing payload and updates its prediction via majority voting as the number of transmitted packets increase.

1.2. Contributions

The main contributions of this thesis can be summarized as follows:

- In contrast to existing classifiers, a multi-phase dynamic classifier is introduced which can classify flows starting with first packet containing payload, and updates its prediction as the number of transmitted packets increase via majority voting.
- Different voting procedures are explored for multi-phase model, to investigate which packets in transmission sequence contain more information for model.

- The multi-phase structure of proposed model allows asynchronous computation, which qualifies proposed model for real time scenarios.

1.3. Thesis Organization

The rest of this thesis is organized as follows. Chapter 2 presents the related works on network traffic classification and their advantages and disadvantages. Chapter 3 describes the proposed multi-phase model and its underlying architecture. Chapter 4 shows the experimental results and the evaluation of proposed model. Chapter 5 presents the conclusion and future work.

2. RELATED WORK

Network traffic classification is a topic, old as internet itself. Using machine learning algorithms for traffic classification is also not a new subject, it is studied for more than 20 years. Examples of such works can be found in literature as early as 1990 [6]. As stated in introduction, traffic classification methods can be divided into three categories as port-based methods, statistical methods and DPI based methods according to information they rely on.

2.1. Port-Based Methods

Port-based classifiers are the simplest and the fastest classifiers. As the name implies, they rely on the port information to classify flows. This approach assumes that every application uses distinct port numbers which can help to distinguish flows. These ports are recorded into a look-up table and compared with the used ports of incoming flows to classify flows. This assumption was reliable for the early stages of internet but as the internet grow and the number of applications increased, this approach became unreliable due to applications using dynamic ports. Nowadays port-based approaches are obsolete, research focus is shifted entirely on statistical classifiers and DPI based classifiers.

2.2. Statistical Methods

Statistical methods are slower compared to port-based methods but they far more accurate in today's world. As the name implies, statistical methods relies on different statistics (features) extracted from flows such as packet inter-arrival times (PIAT), packet sizes etc. These extracted features are used in conjunction with machine learning algorithms or threshold and probability based approaches to classify flows. In the past threshold and probability based approaches were more popular, but the rise of machine learning (especially deep learning) led researchers to focus on methods using different machine learning algorithms.

In [7] authors use, packet size, PIAT and arrival order to create histograms (fingerprints as the authors call them). To classify new flows, they calculate the above listed statistics from flows and compare them with their created fingerprints. Using thresholds, they assign the new flows into one of the existing classes or label it as unknown.

In [8], authors suggest a similar method to [7], but instead of using time based statistics, they use byte counts and probability vectors. They generate attribute meters, such as byte frequency vectors, various request response combinations (e.g. a 'GET' request followed by 'HTTP' response). As in [7], they create a fingerprint database from these attribute meters and compare the new flows with fingerprints and calculate Kullback-Leibler divergence. They use different thresholds to classify flows.

As mentioned, recent studies focus on using machine learning algorithms to classify flows instead of extracting fingerprints and using similarity metrics with thresholds. Machine learning based statistical methods can be divided into two as supervised approaches and unsupervised approaches. Supervised approaches are simpler compared to its unsupervised counterpart. In supervised approaches, known flows with their features are used to train a classifier. These classifiers learn to use extracted features to classify flows.

In [9], authors use 249 different features such as, flow duration, port information, payload size, PIAT to train a Naive Bayes classifier. In [10], authors use different machine learning algorithms such as Decision Trees, Multilayer Perceptrons (MLP), Bayesian Networks with a small set of statistical features to find a scalable, fast classifier. [11] presents a similar study to the [10] with addition of Support Vector Machine (SVM). Recent studies explore deep learning architectures to train classifiers. [12] uses recurrent and convolutional neural networks (CNNs) as classifier using the flow statistics.

In [13, 14] unsupervised approaches are examined. [13] uses Expectation Maximization algorithm which is an unsupervised Bayesian classifier, to learn classes from the flows and use the learned classifier to classify new flows. [14], statistical features are used with payload information to learn clusters and flows are classified calculating the distance between flow and cluster means.

There are also hybrid approaches which combine unsupervised learning with supervised learning. An example of such approach is given in [15]. In [15], first K-means algorithm is used to find clusters. The results of K-means is used as feature with additional statistical features to train a Decision Tree.

The main advantage of statistical methods is that they can work well with encrypted traffic which is a challenging issue for DPI based methods but the drawback is that they need to wait for certain number of packets to be transmitted in each flow to calculate meaningful statistics. The threshold based and unsupervised learning approaches can handle unknown traffic but the supervised learning approaches can not. Another important issue to consider for statistical methods is feature selection process which is vital for the success of classifiers [16]. Using too many features or leaked features can lead to overfitting whereas using too few features can result in underfitting. Both results is undesired.

2.3. Deep Packet Inspection Based Methods

DPI based methods use the payloads of the transmitted packets. They are slower compared to port-based and statistical methods, and usually require more processing power. To classify flows, payloads can be compared with a signature database (similar to fingerprints approach in [7]) where signatures are extracted from payloads or as in recent studies payload can be fed into a machine learning algorithm as feature.

The earliest DPI based methods relies on string matching as in [17]. In [17], authors put forward nine distinct identification methods. Six of these methods relies on DPI by comparing different signatures extracted from payload to a known signature database.

In [18], a multi phase approach is presented. First, flows are classified as application or non-application. The second step is to classify application flows into the different applications. For the second classification, authors use the raw payload data as feature vector. Only first n bytes of payload is considered. They use Naive Bayes, AdaBoost and Regularized Maximum Entropy as classification algorithms.

More recent studies focus on using deep learning which can successfully handle raw input as in image classification and text processing.

In [19], authors propose a classifier employing two neural architectures to classify network flows by using payloads. First one uses stacked autoencoder (SAE), the second one uses CNN. They used “ISCX VPN-nonVPN” dataset [20] to evaluate the proposed architectures. In the proposed SAE architecture, they used five fully connected layers with dropout technique to prevent overfitting. In proposed CNN architecture, they used two one-dimensional convolutional layers followed by a pooling layer arguing that the one-dimensional convolutional layers can capture the spatial dependencies present in the payload which can distinguish different application flows. Authors also stated that CNN based architecture outperforms the SAE based architecture for application classification.

In [21], authors approach the network classification problem as an image classification problem. They create network images from network packets by grouping bits into pixels. Two different deep learning models are trained using these created network images as input. First of these models uses a CNN architecture. In [21], two dimensional convolutional layers are used unlike [19]. The second proposed architecture uses residual network (ResNet, a network with residual connection). It is shown in [21] that for large payload sizes ResNet architecture outperforms CNN architecture, but for small payload size (e.g. 36) CNN architecture performs considerably better compared to ResNet architecture.

In [22], a hybrid approach combining DPI and statistical methods are used to classify encrypted traffic. Payload data is processed by a hybrid network containing

CNN and recurrent neural network (RNN) layers to extract a payload vector which represents the flow. This vector is combined with a statistical characteristics vector, generated by an RNN using different statistical features extracted from flows such as packets per second. The combined vector is fed into a fully connected dense network for classification. The proposed model is tested using traffic classification dataset presented in [20]. As a convolutional layer, one dimensional convolutional layer is used and as a RNN, gated recurrent unit (GRU) is used. Contrary to above mentioned architectures, packet payloads belonging to same traffic processed in parallel CNN layers then concatenated into one where in above models payloads were first concatenated then processed.

In [23], authors use as similar payload based approach to [22]. Packet payloads are processed in parallel one dimensional convolutional layers and concatenated into one vector. This vector is processed again by using another one dimensional convolutional layer and fed into a softmax layer to generate predictions. The proposed model is evaluated by using same dataset presented in [20]. The results are compared with different proposed models including [19] and an architecture containing both CNNs and long short-term memory (LSTM) layers similar to [22]. The proposed model outperforms other models in terms of precision, recall and F1-score for almost all classes present in dataset regardless of the input packet size. Also the authors remark that their proposed method requires least number of epochs compared to other suggested deep learning models.

3. PAYLOAD BASED MULTI-PHASE TRAFFIC CLASSIFICATION

This chapter focuses on introducing general concepts regarding the network packets, deep learning and proposed model. Before delving into the details of proposed model, it will be appropriate to introduce preliminary information. First network packet structure and payload processing will be described. Next, we will introduce some general concepts regarding deep learning and techniques used in proposed model. Lastly, the proposed model will be presented.

3.1. Network Packet Structure

A network packet refers to the single unit of data block transmitted on a computer network [24]. A network packet has three main components header, trailer and payload. The header component contains the instructions to deliver the packet to its destination such as source IP, destination IP etc. The trailer contains control information to signal the receiver this is the end of transmitted packet. The payload component (called also user data) contains the actual message itself which wanted to be transmitted on network. An example of IP packet structure is given in Figure 3.1.

As shown in Figure 3.1 an IP packet contains only header and payload. The header parts carries information such as destination IP address, protocol and meta information about the transmitted packet. The data corresponds to payload. It has no trailer because IP packets are usually transmitted via Ethernet frame which possesses its own trailer and header.

3.2. Payload Processing

As stated before, the proposed model relies only on the payloads to classify network flows. Payload itself is consists of the bytes. It might have variable length (e.g. IP packets can have up to 65536 bytes). For classification task, the payload part of packets must be vectorized, and the length of each vector must be equal in order to train deep learning models. For vectorization, different approaches are used in literature with different complexities ranging from simple vector space models to representation vectors extracted through autoencoders.

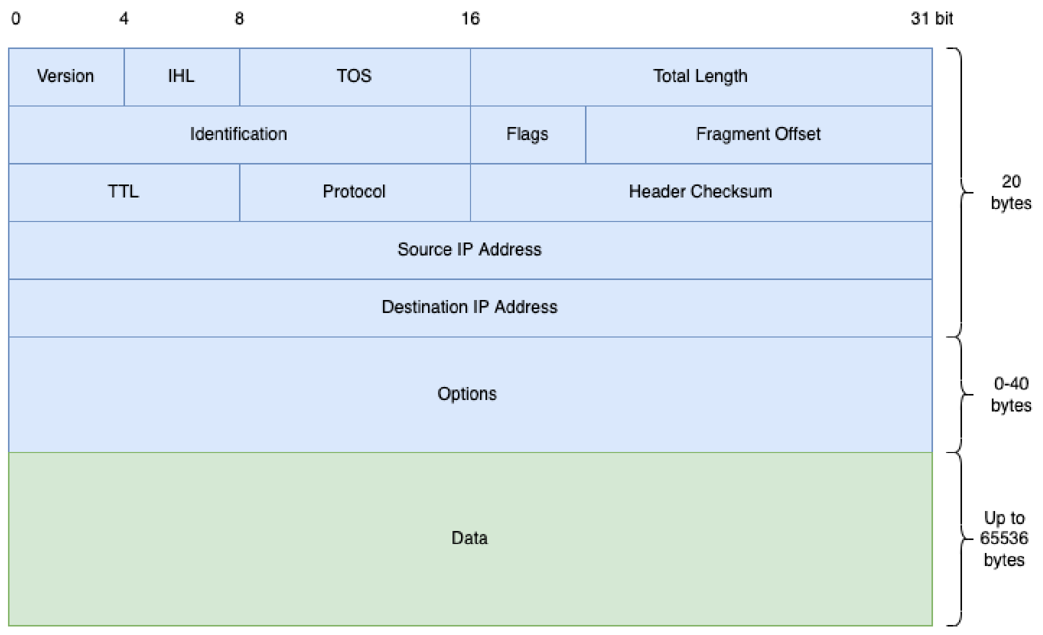

Figure 3.1. IP Packet Structure Reproduced from [1].

3.2.1. Vector Space Model

Vector Space approach is the simple and fast approach for turning a payload into a numerical vector. This approach converts each byte into a predefined numerical value. Usually each byte is represented by a number between 0 and 255 and the corresponding payload is converted into a numerical vector by this mapping. Another and more complex approach is that, instead of representing each byte as one number, we can group sequential bytes into arrays and represent these arrays as one number, an approach known as n-gram. The n-gram represents a continuous sequence extracted from an input with length of n. In other words, we extend the mapping dictionary to include not just bytes but the byte groups. The first approach results in a numerical vector having the same length as the payload itself and this approach can be considered as n-gram vectorization with $n = 1$ also known as unigram.

Another approach is to use count vectors. In contrast to above approach, in this approach each payload is represented by a fixed length vector. This fixed vector keeps track of the each possible n-gram and represents the payload as the count of each possible n-gram found in payload. This count vector can be also transformed to distinguish certain important n-grams from the less important ones by using different methods such as term frequency-inverse document frequency vectorization simply known as TF-IDF [25]. In simplest terms TF-IDF weights importantance of a n-gram by comparing the frequency of n-gram in a collection of payload. It looks how frequent given n-gram in a given payload (term frequency part of TF-IDF) and how frequently this n-gram found in other payloads in given collection (hence the inverse document frequency parf of TF-IDF). A simple calculation for TF-IDF is

$$tf - idf_{t,p} = count_p(t) \cdot log(\frac{N}{n_p}) \tag{3.1}$$

where $count_p(t)$ represent the count of n-gram t in payload p and N is the total number of payloads and n_t is the number of payloads where n-gram t is present. The main advantage of this vectorization approach is, it preserves the all bytes in a more compact representation in exchange of losing spatial information found in payload.

3.2.2. Feature Extraction via Autoencoders

Another widely used method to turn payload into a numerical vector is using autoencoders to learn a latent features representing the payload [26]. Autoencoders are special type of neural network where the input and output of network is the same. Network is trained in a way that it should output the input vector. To make the learned network robust, a small noise is added to input before feeding it into network. Usually, autoencoders are designed as symmetric networks with a bottleneck layer in the middle part. The output of bottleneck layer is considered as the learned representation of input in a lower dimension. The part covering from input layer to bottleneck layer called as encoder part which compresses the input and the part covering from bottleneck layer to output layer called as decoder part which decompresses the compressed input into its original state. An example of such network is given in Figure 3.2.

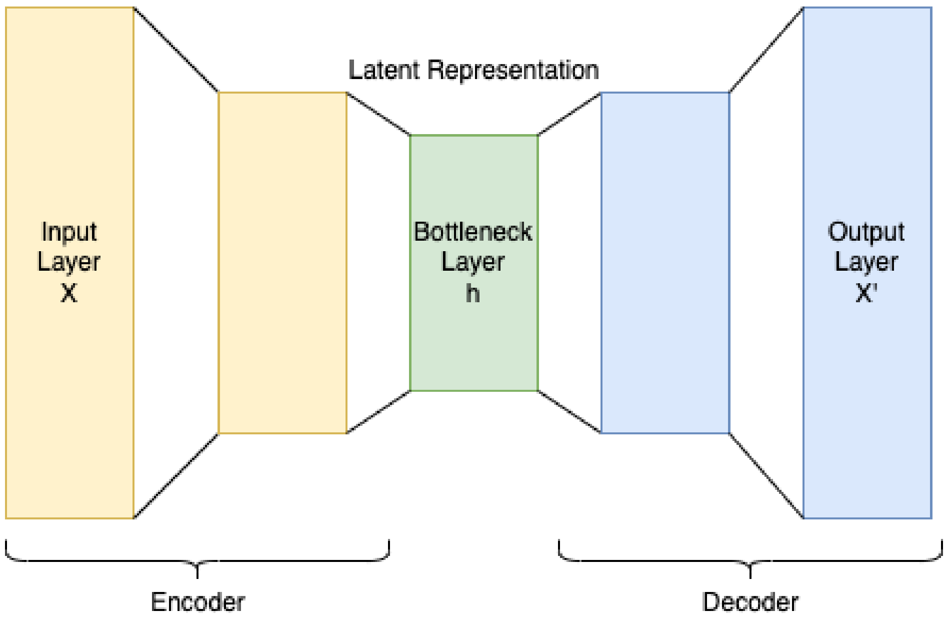

Figure 3.2. Architecture of a Simple Autoencoder Reproduced from [2].

A similar approach to autoencoders is called word2vec proposed in [27]. Word2vec uses neural networks as autoencoder but here instead of reconstructing same input at the output layer, goal is to predict the given input by using its neighbors called as continuous bag of words (CBOW) method or predict its neighbor by using the input called as skip gram method. [27] suggests to use CBOW for applications where speed is essential. By using word2vec method, latent representations for bytes can be extracted and payload can be represented as two dimensional vector consists of representations of bytes or aggregations can be used to lower the dimension of representation vector into one, for example the average of all bytes can be taken.

Both of word2vec and autoencoder approaches promise more compared to the vector space approach in exchange of computational complexity. Compared to the vector space approach, they need a separate neural network to be trained and require more time to process payload into vectors.

3.3. Deep Learning

This subsection focuses on introducing general concepts regarding deep learning which will be necessary for explaining the proposed model. Deep learning refers to a subcategory of broader family of machine learning algorithms which employs artificial neural networks [28]. Originally, the name deep learning is used to describe the neural network architectures which contain more than one layer between input and output layers. These intermediate layers are called as hidden layers. Deep learning architectures can be named by the neural networks they use such as MLP, RNNs, CNNs. Today, deep learning models are widely used in different fields including, computer vision, natural language processing, time-series forecasting, recommendation systems, bioinformatic etc.

3.3.1. Multilayer Perceptron

MLP is the special name of feedforward neural networks (FNNs) with full connections. In other words, each Layer contains at least one artificial neuron and that neuron is connected to the all neurons in preceding and succeeding layers. The name perceptron refers to a specific kind of artificial neuron with unit activation function [29]. Although in modern sense, MLP is used as an umbrella term containing all FNNs. The FNNs are the simplest neural network architecture and contains only forward connections without feedback connections. The one of the earliest examples of such network can be found in [30].

An artificial neuron receives the input signal and maps it into an output via weighted sum and activation function (also referred as transfer function). The output of a neuron is given with following formula

$$y = \phi\left(\sum_{i=0}^{N} w_i * x_i\right) \tag{3.2}$$

where y refers to the output of neuron, $\phi\left(.\right)$ is the activation function, and the vector $\mathbf{w} = \begin{bmatrix} w_1 & w_2 & \cdots & w_n \end{bmatrix}$ is the weights of the neuron and $\mathbf{x} = \begin{bmatrix} x_1 & x_2 & \cdots & x_n \end{bmatrix}$ is the input signal. A simple illustration of neuron can be found in Figure 3.3.

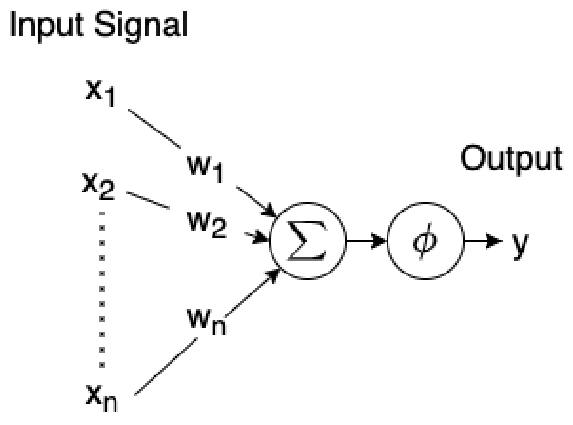

Figure 3.3. Illustration of Neuron Reproduced from [3].

By using unit step function as activation function (also called as heaviside step function) perceptron can be used as binary classifier or by using softmax as activation function it can be used in multiclass classification. Following functions given in Table 3.1 are the most popular choices for activation functions.

Usually, for a model to be considered as a deep model, it should have minimum of three layers, including the input and output layers. The deep learning models are trained using backpropagation and a selected loss function. Loss function gives a quantitative result measuring how well the model is fitted regarding the learning problem which can be a classification task or a regression task. The goal is to minimize the loss function. Backpropagation calculates the gradient of loss function with respect to the weights in corresponding layers. The calculated gradients are used in updating the weights using an optimization algorithm such as gradient descent. The magnitude of update is controlled by a special parameter called learning rate. This updates takes place in certain intervals where a batch of training observations is fed into model and the resulting predictions is compared with the actual outputs. The size of the batch can vary from one observation to entire training set. One complete tour of training data is called as epoch and models are trained in epochs. As the number of epoch increases, weights converge to a stable state which represents the optimization of loss function. The main advantage of MLP is the resulting deep structure can learn the non-linear functions. This is the direct result of cascading multiple layers.

Table 3.1. List of Activation Functions.

Name	**Formula**	**Range**
Identity	x	(-∞,∞)
Unit Step	$\begin{cases} 0 & x < 0 \\ 1 & x \geq 0 \end{cases}$	{0,1}
Sigmoid	$\dfrac{1}{1+e^{-x}}$	(0,1)
Tanh	$\dfrac{e^x - e^{-x}}{e^x + e^{-x}}$	(-1,1)
Rectified Linear Unit (ReLU)	$\begin{cases} 0 & x < 0 \\ x & x \geq 0 \end{cases}$	[0,∞)
Softmax	$\dfrac{e^{x_j}}{\sum_{i=1}^{N} e^{x_i}}$ for $j = 1, 2, .., N$	(0,1)

3.3.2. Convolutional Neural Networks

CNNs are special FNNs, mostly used in problems containing unstructured data like images, videos, speeches, texts [31, 32]. Main reason for this preference is that CNNs are great to exploit spatial relations due to their unique structure. They are more regularized compared to the MLPs meaning that they have considerably less learned parameters which makes them prone to overfitting. This regularization achieved by weight sharing property of convolutional layers. Hidden layers of a typical CNN architecture consists of 3 different kinds of layer. Usually these layers are found in following order, convolutional layer, pooling layer, fully connected layer. A simple example of CNN architecture is given in Figure 3.4

The convolutional layers, as its name implies, implement convolution operations to their inputs. Convolution operation helps to distinguish local patterns and they are implemented via filters. These filters traverse the input with respect to its dimensions. This is illustrated in below Figure 3.5. As it can be seen from Figure 3.5, filters scan the input and maps it into output by multiplying each cell with corresponding weight in the filter and sums the multiplication results. This reduces the size of input. This size reduction can be avoided by zero padding to input hence increasing its size so that the resulting output will have the same size with original input. The output of convolutional layers are called as feature map. The number of dimensions in filter which is used for convolution should be equal to number of dimensions in input. The hyperparameters in convolutional layers which needs to be decided are listed as follows, filter size, stride (decides how many step the filter is shifted in succeeding operations) and number of filters. All of these hyperparameters effects the shape of output. As mentioned above CNN has to learn fewer parameters compared to its fully connected correspondence. The reduced number of learned parameters can be explained as follows, consider an image input of size 32x32 which is a very small image compared to today's standards. If one decides to use a fully connected layer, each neuron must have 32x32=1024 parameters and usually a layer consists of more than one neuron which increases the number of learned parameters. Compared to fully connected layer, a convolutional layer with filter of size 3x3 has only 9 parameters which reduces the number of learned parameters significantly.

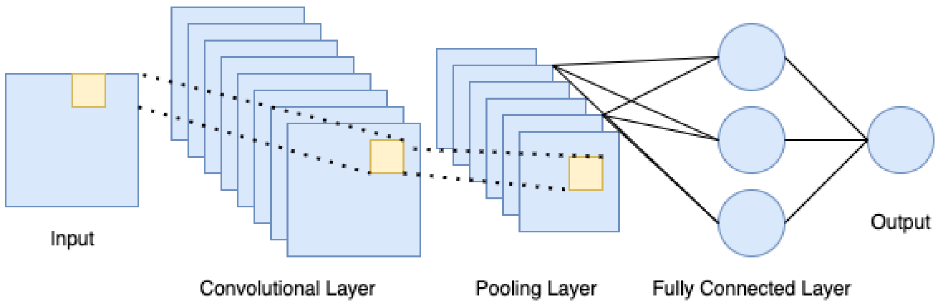

Figure 3.4. An Example of CNN.

Usually a convolutional layer is followed by a pooling layer. The purpose of pooling layer is to downsample the output of convolutional layer and provide robustness to local variations. There are two kinds of pooling, local pooling and global pooling. Local pooling generates a new feature map by downsampling the output of convolutional layer. Global pooling reduces each feature map into a single output, in other words it flattens the the output of convolutional layer. Global pooling is equal to local pooling with pooling filter size equal to size of feature map resulting from convolutional layer. Pooling layer works in a similar manner to convolutional layer. It consists of a filter and implements a convolution operation but it has no learnable parameters. In other words pooling layer can be considered as convolutional layer with a filter of constant values. Most popular pooling operations are max pooling and average pooling. Max pooling takes the maximum value of the current window of feature map whereas the average pooling takes the average of current window.

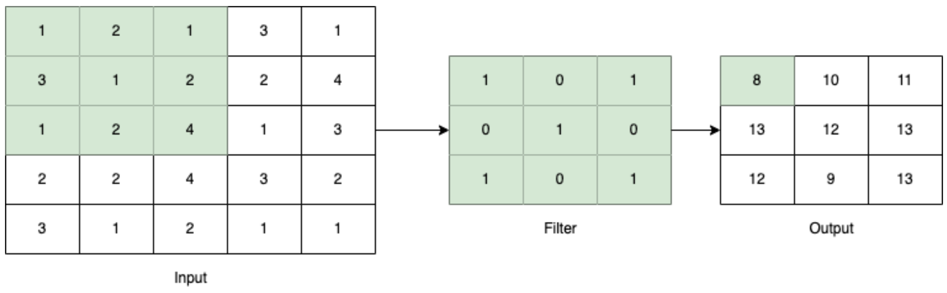

Figure 3.5. An Example of Convolution with Filter Size of 3x3 and No Padding.

The fully connected layer takes the output of pooling layer. If needed, the output of pooling layer is flattened before it is fed into fully connected layer. This layer is same as any hidden layer in MLP. Usually more than one fully connected layer is used to decrease the size of output of pooling layer gradually.

The CNN can be summarized as follows, first the convolutional layer extracts features from input. Then pooling layer provides feature selection. In the last step fully connected layers learns the given task (usually a classification task) using the extracted features. As stated in above CNNs are usually used in image processing due to its power to take spatial arrangements into account and extracts local patterns as features. Every filter can be considered as a special pattern detector scanning the entire input for a specific pattern. This is the result of weight sharing. In training phase, convolutional layer learns the which local patterns it should look for in a given input.

3.3.3. Recurrent Neural Networks

In contrast to CNN and MLP networks, the output of RNNs doesn't just depend on only the current input but it also depends on the current state of network. This state is called as memory. This special property of RNNs allows the processing of inputs with any length. RNNs can be considered as filters with infinite impulse response. The most popular RNN networks are implemented by using GRUs or LSTMs.

Classical architecture of RNN is given in below Figure 3.6. Left side of Figure 3.6 shows the the original structure and the right side shows the unfolded version of it. As one can see from the Figure 3.6, the current output y_t depends both on current input x_t and memory a_{t-1}. Due to this feedback connection provided by memory, RNNs can handle inputs with infinite length without increasing the model size. RNNs are very popular for processing sequential data like speech and text. Main reason is that the current output of model depends on the past inputs (represented by memory) as well as current input which can be very useful for tasks such as speech recognition, machine translation, time-series forecasting. A similar weight sharing process as in CNNs is also present in RNNs where weights are shared temporally. Meaning that the successive inputs in time share the same weights.

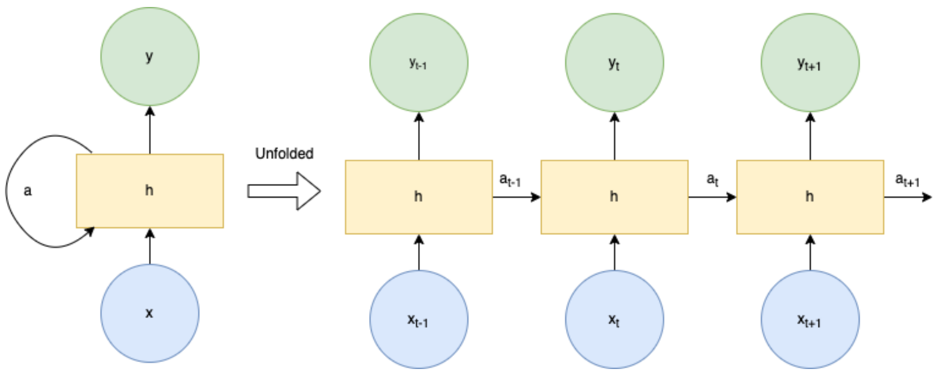

Figure 3.6. General Architecture of RNN Reproduced from [4].

The early vanilla RNN networks suffered gradient problems (vanishing, exploding) in training via backpropagation. In practise, gradients tends to vanish or converge to infinity rendering trained network useless. To overcome these gradient problems new variations of backpropagation are proposed such as backpropagation through time. As stated above two most popular RNN variations are GRU and LSTM.

The LSTM is first introduced in 1997 by [33]. It can successfully prevent vanishing gradient problem but exploding gradient problem still can be encountered. The vanishing gradient problem is solved by using gates in LSTM. A LSTM unit has three gates in total namely: forget gate, input gate and output gate. Forget gate decides if it need to retain past information coming from previous inputs or the past information can be forgotten. Input gate relegates the impact of current input to output of LSTM cell. Output gate calculates the output of cell. These cells work together to control the information flow in LSTM cell. Internal structure of a LSTM cell is given below Figure 3.7.

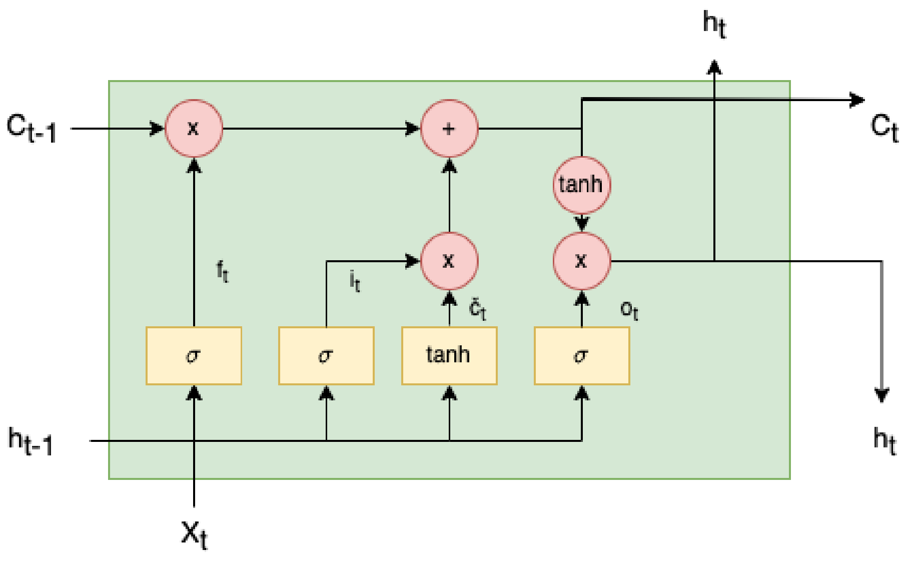

Figure 3.7. General Architechture of LSTM Cell Reproduced from [5].

The governing equations of LSTM can be written as

$$i_t = \sigma\left(x_t U^i + h_{t-1} W^i\right) \quad (3.3)$$

$$f_t = \sigma\left(x_t U^f + h_{t-1} W^f\right) \quad (3.4)$$

$$o_t = \sigma\left(x_t U^o + h_{t-1} W^o\right) \quad (3.5)$$

$$\check{C}_t = tanh\left(x_t U^g + h_{t-1} W^g\right) \quad (3.6)$$

$$C_t = \sigma\left(f_t * C_{t-1} + i_t * \check{C}_t\right) \quad (3.7)$$

$$h_t = tanh\left(C_t\right) * o_t \quad (3.8)$$

where x_t denotes input vector, i_t is output of input gate, f_t is output of forget gate, $\check{C}_t$ is the cell activation vector, o_t is the output of output gate, c_t is cell state, h_t is the hidden state vector, also called as output vector. The U and W matrices denote the learnable weights of LSTM cell. As one can see from the governing equations the cell state is controlled by previous state (effected by output of forget gate), cell activation vector and input vector. The output of LSTM unit is calculated by cell state and output of output gate.

A simplified version of LSTM is proposed in [34] which is known as GRU. The internal structure of GRU cell is very similar to LSTM cell but it doesn't have the output gate, hence it has fewer learnable parameters compared to LSTM cell. This simplification reduces the required computational power and the training time in exchange for a simpler model which can perform worse compared to its LSTM counterpart. But in certain tasks, it is observed that networks employing GRU cells perform close to its LSTM counterpart hence justifying their use.

To summarize, RNNs are very useful for temporal data where the current output is effected by past outputs and inputs. Due to this special property they are widely used in time-series problems as well as text and speech processing problems. By reversing the flow of data and training a new neural network, a bidirectional RNN can be implemented. In this mode, the output of forward and backward network are concatenated and the combined output is the output of bidirectional network.

3.4. Proposed Model

In this thesis our main purpose is to present a deep learning based ensemble model which can classify network flows by using their packet payloads. This section presents the proposed model. The model consists of three main modules. These modules are payload preprocessor, deep learning models and majority voter. The overall architecture is given in Figure 3.8. The proposed model predicts an application class for a given flow starting from the first packet with payload and updates its predictions as the new packets with payload are acquired which enables the proposed model to be flexible and responsive to the contents of new packets.

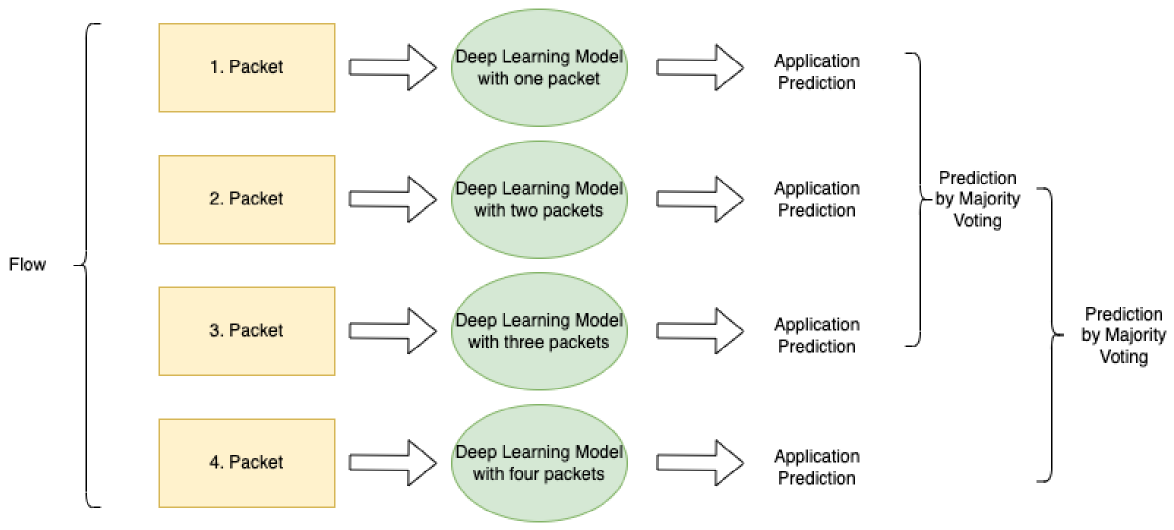
Flow
1. Packet
2. Packet
3. Packet
4. Packet
Deep Learning Model with one packet
Deep Learning Model with two packets
Deep Learning Model with three packets
Deep Learning Model with four packets
Application Prediction
Application Prediction
Application Prediction
Application Prediction
Prediction by Majority Voting
Prediction by Majority Voting

Figure 3.8. Overall Architecture of Proposed Model.

As shown in Figure 3.8, the outlying working principles of proposed model can be explained as follows. The model starts to generate predictions with the first packet containing payload from the flow. The payload is turned into a numerical vector with a fixed length and fed into the corresponding deep learning model which uses one packet vector to classify flows. The returned application prediction is published as the application type of the flow and stored. When the second packet with payload is acquired from flow, it is also processed as the first packet and the payload vectors of first two packets fed into a different deep learning model. This model is similar to the first model but it uses two payload vectors to classify flows. The returned application prediction is published as the application type of the flow and stored. Same process is repeated with the third packet with payload but this time instead of returning the application prediction of the deep learning model with three packets as is, majority voting is applied using the current prediction with previous two application predictions obtained from previous models. The resulting prediction of voter is returned as the application type of the flow. This process is repeated as the new packets with payloads are acquired from the flow and the application type of flow is updated according to the result of majority voting module. This approach enables asynchronous computation, meaning we can start to generate predictions as the new packets arrive and store them to use later in voting procedure. In other words by the time, fifth packet arrives, we will be already have predictions from the previous models to use in voting.

In the remainder of this section the modules of proposed model will be explained in details, starting with the payload preprocessing followed by deep learning models and majority voting methodology.

3.4.1. Payload Preprocessing

As described in above section 3.2 payloads are needed to be vectorized and made model ready to train subsequent deep learning models. Usually deep learning models require a fixed size of input to be trained. This is also true for our proposed deep learning architecture. To preprocess payloads into a training ready vectors, two preprocessing methods are considered. One employing a vector space approach, other is using feature extraction via autoencoders.

The implemented vector space approach can be described as follows, in the first step headers are removed from the packets. Second step is to turn the bytes in payload into corresponding numerical values between 0 and 255. Third step is the length modification by truncation and padding. This modification is done to equalize the length of vectors. Let $\mathbf{P_i}$ denote the payload vector of the i^{th} packet

$$\mathbf{P_i} = \begin{bmatrix} x_1^i & x_2^i & \cdots & x_n^i \end{bmatrix} \tag{3.9}$$

where $x_t \in \{0, 1, \cdots, 255\}$ corresponds the t^{th} payload byte of i^{th} packet and n is equal to length of payload vector which is a hyperparameter to be decided. If the payload size is larger than n, payload is truncated in such a way that it contains first n bytes.

On the other hand, if the payload size is smaller than n, payload vector is padded with zeros until its length is equal to n. As a result, each packet payload is represented by a vector $\mathbf{P_i}$ with equal size of n. This process is illustrated in Figure 3.9. As shown in Figure 3.9, first payloads are extracted, then they are truncated or padded to make their size equal.

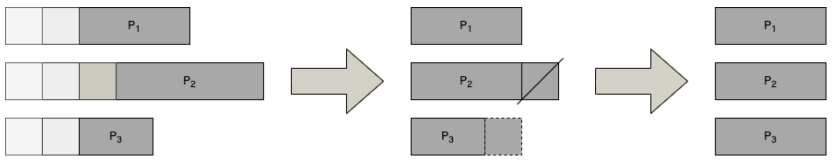

Figure 3.9. Packet Vectorization.

The above described vector space approach retains the spatial relations in payload but to retain the all bytes in payloads, the hyperparameter n which is the length of payload vector must be set large enough which will be impractical in implementation. To overcome this, TF-IDF approach can be used but we opted to not use it due to the nature of TF-IDF which removes the spatial relations from payload. In this problem spatial relations are considered more important and by setting an appropriate number for n the information loss can be minimized.

The second approach uses autoencoders to extract latent features from payloads. For autoencoders two architecture is considered one uses simple fully connected layers and the other one uses convolutional layers. The autoencoder takes the processed payload vectors from the first approach as input. Also one needs to condsider an additional hyperparameter in autoencoder approach, h which is the embedding size. An example of usage of autoencoders to process payloads given in [35].

3.4.2. Proposed Deep Learning Architecture

Our proposed deep learning architecture carries properties of all introduced networks namely CNN,RNN and MLP, in other words it uses fully connected layers, convolutional layers and LSTM. We propose one general architecture similar to introduced in [22]. This architecture is adapted with small adjustments according to number of packets in input side. The overall architecture for classifier which uses 4 packets is given Figure 3.10. The proposed architecture can be divided into two parts. First part consists of parallel flows which extracts payload features from payload vectors and uses LSTM to extract flow based features. The first part covers from input layer to bidirectional LSTM and can be named as feature extraction part. Second part can be named as classifier part. This part classifies the flows based on the extracted features coming from the first part.

This architecture integrates the second payload preprocessing approach (feature extraction via autoencoders) in it via convolutional layers. Instead of training a separate network to extract features and train a separate classifier, this is done in one integrated network. So the first part of architecture can be considered as the extension of payload preprocessing to extract latent representation vector from payload. The proposed architecture complies with intrinsic nature of network packets indicating that the relation between sequential packets are retained. The parallel processing of packets from the same flow, speeds up the processing and extracts individual properties of each payload. At the end of the first part, the extracted payload features are combined and fed into bidirectional LSTM to extract flow features because sequential packets can't be considered independent from each other. For example, let us consider a video which will be streamed. It is impossible to transmit all of the video in one packet due to established protocol limits. So usually this video is divided into separate packets and transmitted as multiple packets. Then the receiver combines the information from these packets and reconstruct the video. As one can see from this example, an object is divided into separate sequential packets, revealing the relationship between sequential packets. In this case, the sequential packets are the smaller parts of the same object. As stated, processing the extracted features from payloads with bidirectional LSTM, can be considered as extracting the flow based features for classification. In the classifier part, these flow based features are fed into fully connected layers to generate application prediction.

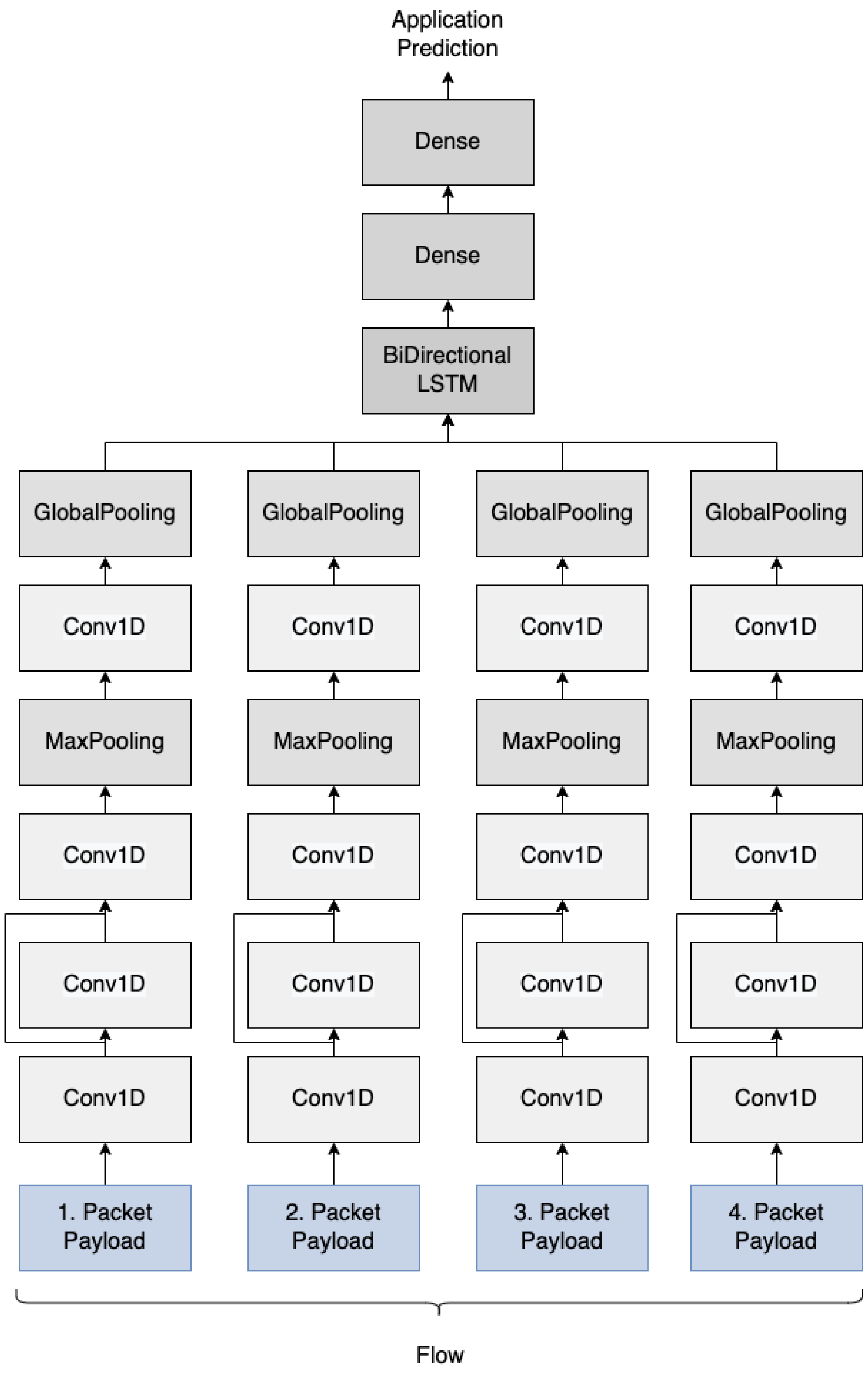

Figure 3.10. Proposed Deep Learning Architecture for 4 Packets.

The parallel part of the feature extraction part is implemented to extract packet based features from payloads to represent each packet. This feature extraction is realized by using convolutional layers. For this purpose, we opted to use one dimensional convolutional layers. As one can see from the Figure 3.10 convolutional layers are stacked and residual connection is used to further increase the non-linearity of extracted features enabling model to explore more complex relations in payloads. By using pooling layers, downsampling, in other words feature selection is achieved which provides dimension reduction and robustness to local variations. There are two main reasons behind using convolutional layers, first one is their proven ability to extract local features by exploring spatial relations and the second one is encryption. Nowadays most of the transmitted packets are encrypted and this makes it impossible to extract semantic information from the payloads directly. By using convolutional layers we hope to extract patterns which can be more useful in this case as in [19]. This parallel structure is named as ResPacket in [22].

Another popular choice to extract flow based features is to use two dimensional convolutional layers as in [36]. In this approach network flows are represented as two dimensional images constructed by sequential packet payloads. The width of the constructed images is equal to length of payload vectors and the height of the constructed image is equal to the number of sequential packets which will be used for application prediction. In this approach both payload based and flow based relations explored concurrently and the resulting vector is can be considered as flow based features which are ready to classification omitting the need of using LSTM. In our work this approach is also considered but it is deemed inferior to above approach. The main reason is this approach can fall to a pitfall and extract non-existing relations between packets which could lead model to overfit training data. By non-existing relations, we refer the bytes found in same location in sequential packets. For example, in this approach a two dimensional convolutional layer will try to explore relations between fifth bytes of successive packets but this relation does not carry meaning in terms of network traffic especially if the payloads are encrypted which is the case here. So instead of using two dimensional convolutional layer approach as in [36], we preferred to one dimensional convolutional layer to extract payload based features as in [22], [23] and use a recurrent layer to explore flow based relations which considered more logical compared to the two dimensional convolutional layer approach.

As mentioned above the second part of the feature extraction part is implemented by bidirectional LSTM. This part starts with merging of of payload based feature vectors which were extracted in first part via convolutional and pooling layers in parallel. This merged vector is fed into a bidirectional RNN layer to explore both forward and backward relations between packets. For RNN layer, both GRU and LSTM are considered and the performance of both GRU and LSTM based architectures are compared. The performance of both architecture were similar with LSTM having the upper hand with a small margin which led us to choose LSTM over GRU. We chose to use LSTM to extract flow based features. LSTMs are widely used in time series problems. Network flow classification also can be considered as a time series problem. Both GRU and LSTM has memory to retain long term relations found in input. For example let us consider a flow with two sent packets with a received packet in between. Here two transmitted packets can be related and the received packet can interrupt this but LSTM can preserve this information due to its nature. The journey of payload vectors are shown in Figure 3.11.

In the classifier part of proposed architecture, the extracted flow features are fed into fully connected layers followed by an output layer with softmax function to generate predictions. This part is a typical multi-class MLP classifier.

As mentioned in previous sections for different number of available packets different models are trained. In total five model is trained for inputs containing 1,2,3,4 and 5 packets. These models share the same architecture described in this section. The only significant difference is that the models for inputs containing one and two packets do not include LSTM. The reason for this exclusion is that these models contain very few packets that the exploring correlations between packets degrades the performance of trained models. In these models an additional fully connected layer is included in place of LSTM.

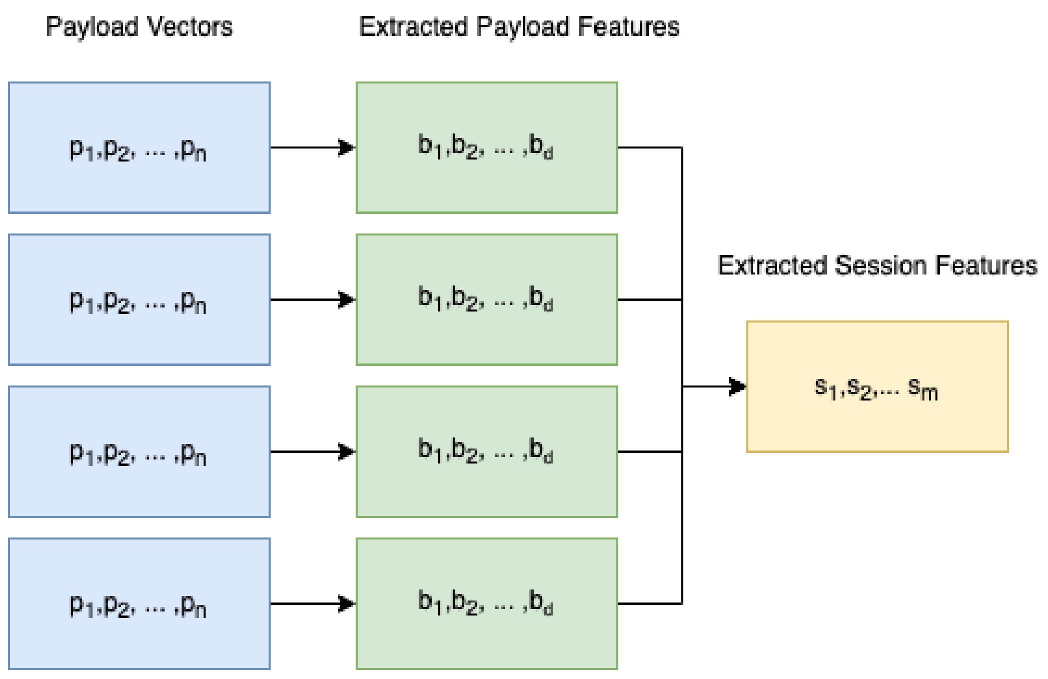

Figure 3.11. Illustration of Feature Extraction for 4 Packets as Input.

3.4.3. Majority Voting

The last module in proposed traffic classification model is the majority voting based on the past predictions made by previous classifiers which used less packets. The main purpose of including a voting mechanism in proposed model is to increase the accuracy of flow classification without waiting for additional packets. This enables model to make better predictions with available information. Our proposed model starts to make application predictions for flows starting with the first packet with payload, in other words it starts to classification as soon as possible and as the number of transmitted packets increases new predictions are generated by using corresponding classifiers and the overall prediction for flow is updated. Instead of directly using the prediction of current classifier, the proposed model looks for a consensus between recent predictions. For example when the fourth packet with payload is received, the corresponding classifier (the one which uses four packets as input) is used to generate application prediction, then the previous predictions, made by classifiers with inputs containing two and three packets, use in conjunction with latest prediction to update the application prediction via majority voting. By using majority voting, we are aiming to prevent latest received packets from misleading the classifier due to its content. For example, the flow can be belong to a chat application but the latest received packet can contain images which can lead model to predict the application as a social media application. Of course, as the number of the packets used in input increases, the effect of these misleading packets are reduced but our purpose in this study to make correct classification in a fast manner using minimum number of packets so that it can be used in real time systems. In other words we don't have the luxury to wait additional packets to reduce the effect of these misleading packets. Our solution is as mentioned above is to use majority voting.

Different strategies are tested for majority voting. These strategies can be summarized as follows.

- Majority voting by using all available predictions
- Majority voting by using all latest three predictions
- Majority voting by using the odd-numbered predictions

In case of an equality the latest available prediction is selected as the overall prediction.

4. EXPERIMENTS AND RESULTS

The proposed network classification model is tested on two real world traffic datasets. The proposed model is compared with other baseline models to evaluate the its performance and to show the advantages of proposed model. Before presenting the experimentation results, we will introduce the used datasets briefly and give details regarding the experiment setting.

4.1. Traffic Classification Datasets

To evaluate the performance of both proposed model and other benchmark models, two dataset is selected. First one is "Application Based Network Traffic Dataset" [37] which is public dataset hosted on Kaggle and the second one is a private dataset created by a private company which we can't reveal. For the remainder of this thesis the second dataset will be named as "Private Dataset".

"Application Based Network Traffic Dataset" dataset contains flows from 22 different applications varying from chat applications to video streaming applications. The complete list of applications given with their number of flows in below Table 4.1. More details regarding the dataset can be found in [38]. Although the dataset contains approximately 4800 flows, we discard the flows which don't contain any packet with payload. We also discard the packets whose payload consists of zero bytes only. The remaining number of flows is 39879 and its distribution to applications is given in Table 4.1. As the stated in previous sections, we only care for the first five packets with payloads. Each flow is represented by a flow matrix constructed from first 5 packets with payload and chosen fixed length of payload. For this thesis we represent each flow with 180 bytes of each payload. These payloads are extracted from each flows converted into numerical vectors with clipping and padding operations to make it length 180. If a flow contains less than five packets with payload, then padded payload vectors (vectors with zeros) are appended to flows to make it up for missing packets. In the end we have 39879 flows, each represented by 5x180 matrix.

Table 4.1. Application List of Application Based Network Traffic Dataset.

Application	**Number of Flows**
Amazon Prime Video	1978
CyberGhost	385
Deezer	750
Discord	440
Dropbox	491
Epic Games	1346
Facebook	436
Hotspot Shield	437
iTunes	742
Microsoft Teams	495
ProtonVPN	554
Skype	912
Slack	1352
Soulseekqt	110
Spotify	1776
Steam	520
Telegram	372
TunnelBear	12339
TuneIn	1790
Ultrasurf	11437
Whatsapp	432
Zoom	784

The "Private Dataset" contains 34 different applications. The total number of flows is 20495. The complete list of applications with their flow counts given in Table 4.2. Same preprocessing steps described in above are also applied to this dataset. This dataset is more challenging compared to the above described public dataset due to its limited flow size and increased number of application types.

Table 4.2. Application List from Private Dataset.

Application	**Number of Flows**
Bip	720
Discord	395
Facebook	1405
Fiesta	180
Hangouts	125
ICQ	110
League Of Legends	685
Line	125
Linkedin	400
Ms Stream	100
Minecraft	3090
PokemonGo	530
Pplive	255
Ppstream	965
Pubg	575
Quake	310
Riot Games	650
Signal	100
Skype	135
Snapchat	860
Steam	960
Tor	810
Tango	1255
Telegram	215
Tiktok	1680
Twitter	400
Ultrasurf	835
UTorrent	150

Table 4.2. Application List from Private Dataset. (cont.)

WebThunder	370
Wickr	355
Yammer	205
iTunes	1440
Tunnelbear	105

The resulting distributions of datasets are unbalanced. For example Tunnelbear contains more than times flows compared to Soulseekqt. To balance these datasets we could use undersampling or oversampling methods but considering that the deep learning models require large amount of data we opted to not use any undersampling methods. We also decided to not use oversampling methods to balance the dataset in fear of overfitting the training data. Instead we decided to use Recall, Precision and F1 score as our main metrics which are widely used in unbalanced problems.

4.2. Experiment Setting and Evaluation

To evaluate the performance of proposed classification model, we trained 5 deep learning models in total, each corresponding to a different input size and compared them with the three baseline models. These baseline models are Logistic Regression, SVM and RandomForest. The baseline models uses TF-IDF vectors extracted from payloads. We selected the model which have 5 packets as input, to decide the payload length parameter for classification models. The decided payload length is used in classification models. We also used a transfer learning method to shorten the training time of models. We used the learned weights in previous model to train next model. For example we used the learned weights from deep learning model which has 3 packets as input, to initialize the training of model which has 4 packets as input. Subsequently, for the training of model with 5 packets, we used the weights from the model with 4 packets. We used categorical crossentropy as loss function for all trained models and all models are trained using the ADAM optimizer [39]. The 80 percent of the data is used as training set and the remaining 20 percent used as test set. We didn't set a separate validation set instead we used 5-fold cross validation for hyperparameter tuning. After finding the optimal parameters with cross validation we used all of the training set to train our model with the found hyperparameters, and evaluated the resulting model using the test set. The deep learning models are implemented using $Keras$ [40] library in $Python$.

To compare the proposed models with baseline models and evaluate their performances following classification metrics are selected:

- Accuracy
- Precision
- Recall
- F1 score

The accuracy measures overall performance of classifier, by calculating the ratio of correct predictions to all predictions. The precision and recall are class specific metrics. Precision measures the correctness of all predictions for a given class. It is calculated for a given class by using the all predictions carrying the label of that class. It is the ratio of the number of observations belonging to specific class and predicted correctly to the number of observations predicted as belonging to that specific class. The recall metric can be considered as a complementary part for precision. The recall measures how much of the observations belonging to specific class predicted correctly. It is the ratio of number of observations belonging to specific class and predicted correctly to number of observations belonging to specific class. A good classifier must have high precision and recall. The last metric which we used for evaluation called as F1 score. It is the harmonic mean of precision and recall and provides a single measurement to compare classifiers. F1 score is used in situations where accuracy metric may not be reliable its own. This usually occurs when the dataset is unbalanced as in our case. In such cases predicting only the majority class as label can lead to misleading results in terms of accuracy. For example if the majority class consists of the 90 percent of dataset, then returning this class as prediction for all observations will result in accuracy of 90 percent which is can be considered as a good result in most of the cases.

So in cases where the underlying dataset is unbalanced, precision and recall are used as primary metrics. But it is not easy to compare two classifiers where one of them has higher recall and the other one has higher precision. This is where the F1 score steps in. In this work although all metrics are reported, we rely on weighted F1 score to compare different models. The weighted F1 score is calculated by taking the mean of F1 scores of all classes with respect to their support.

4.3. Results

4.3.1. Payload Length

In this section, the results of experiments are presented showing the advantages of using the proposed model but firstly, we present the results of payload length experiment in which we tried to decide the optimal number for payload length to use in our models as mentioned in above sections. To decide the payload length we used the model with 5 packets (in other words model whose input consists of first five packets with payload of flows) with "Application Based Network Traffic Dataset" as input dataset. Using this setting, we tried different numbers as payload length n ranging from 50 to 180. The results are presented in Figure 4.1 and Figure 4.2.

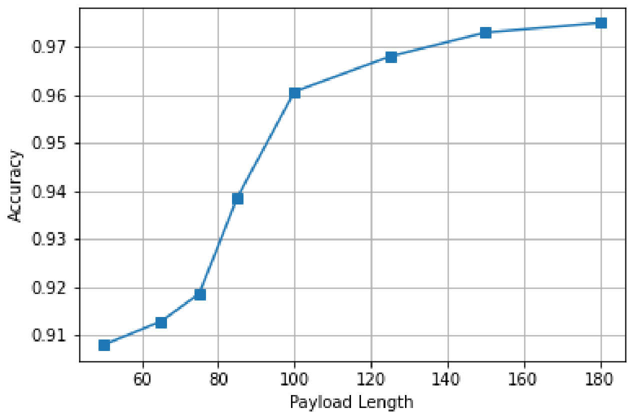

Figure 4.1. Accuracy versus Payload Length.

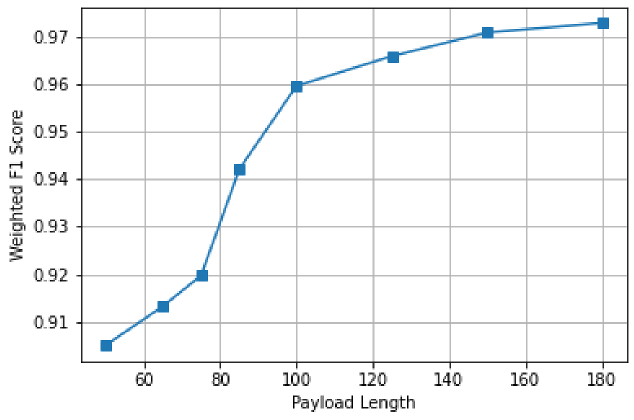

Figure 4.2. Weighted F1 Score versus Payload Length.

As expected, as the payload length increases, both accuracy and weighted F1 score is also increase. The shape of the Figure 4.1 and Figure 4.2 are similar to sigmoid function, showing a clear jump between payload length of 75 and 100, indicating that it is better to select payload length greater than 85. After the payload length of 85, we see a diminishing return. Increasing the length of payload after 150 seems to be very inefficient because we see that after payload length of 150 both plots reach to a plateau. In light of these observations, we decided to select payload length n as 100 which can be considered as the start of plateau, so it is a reasonable choice, hence all following models are trained using the payload length of 100.

4.3.2. Application Based Network Traffic Dataset

The overall flow classification results for "Application Based Network Traffic Dataset" are given in Table 4.3. The precision, recall and F1 score metrics are calculated by taking the weighted averages. When we inspect the Table 4.3, we can see easily that increasing the number of packets leads to more accurate classifications. This holds true for all of the models listed in Table 4.3. We can also see that the proposed deep learning based classifier outperforms all the baseline models for all number of packets in all metrics with a considerable margin and adding voting to proposed classifier results in a better performance.

Table 4.3. Classification Results for Application Based Network Traffic Dataset.

Model	Number of Packets	Metric			
		Accuracy	Precision	Recall	F1 Score
SVM	1	0.786	0.767	0.786	0.769
	2	0.804	0.852	0.804	0.797
	3	0.868	0.86	0.868	0.856
	4	0.884	0.909	0.884	0.891
	5	0.909	0.921	0.909	0.912
Random Forest	1	0.786	0.741	0.786	0.74
	2	0.82	0.791	0.82	0.784
	3	0.838	0.817	0.838	0.81
	4	0.843	0.818	0.843	0.812
	5	0.853	0.834	0.853	0.821
Logistic Regression	1	0.819	0.784	0.819	0.793
	2	0.876	0.863	0.876	0.859
	3	0.88	0.869	0.88	0.87
	4	0.9	0.898	0.9	0.89
	5	0.926	0.926	0.926	0.925
Proposed Classifier	1	0.85	0.844	0.85	0.835
	2	0.893	0.901	0.893	0.892
	3	0.902	0.9	0.902	0.897
	4	0.922	0.929	0.922	0.915
	5	0.961	0.963	0.961	0.96
Proposed Classifier with Voting	3	0.913	0.919	0.913	0.911
	4	0.94	0.946	0.94	0.932
	5	0.971	0.974	0.971	0.965

The confusion matrices for individual models are also given in from Figure 4.3 to Figure 4.7.When we inspect confusion matrices closely we can see that the classifier confuses some applications such as Slack and Spotify. Similar confusion is also present in applications which regulates network traffic such as ProtoVPN and Ultrasurf. Classifiers tend to misclassify flows belonging to these applications. Increasing the number of packets from flows corrects these classification errors.

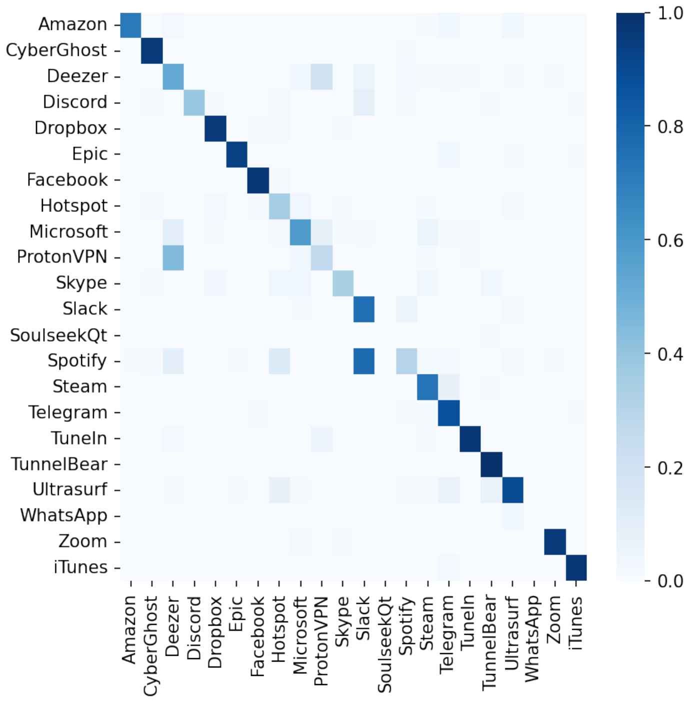

Figure 4.3. Confusion Matrix for Model with 1 Packet.

We also explored different majority voting scenarios which are presented in Table 4.4. The results suggest that the best approach is to apply majority voting to the the previous two models in conjunction with current model. Another point to consider is using models from early stages (models with one or two payload packets) are not that helpful. We can argue that for example when we reached to the fifth packet, the prediction of the first model can be outdated and using it in majority voting with fifth model is not beneficial as other scenarios. This can be seen from comparing the results of 1,2,3,4,5 with 2,3,4,5 or 1,2,3,4,5 with 1,3,5 which also shows the same effect. Removing models with two and four packets, decreases the performance considerably.

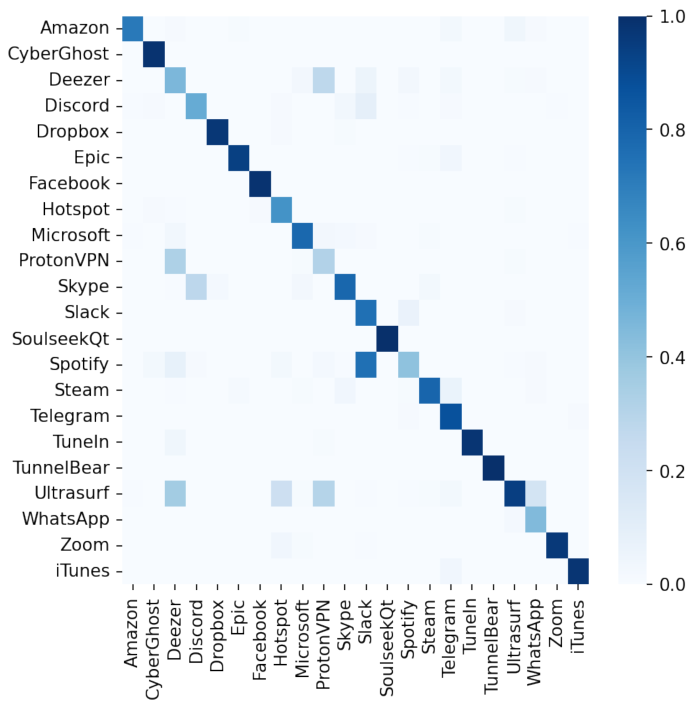

Figure 4.4. Confusion Matrix for Model with 2 Packets.

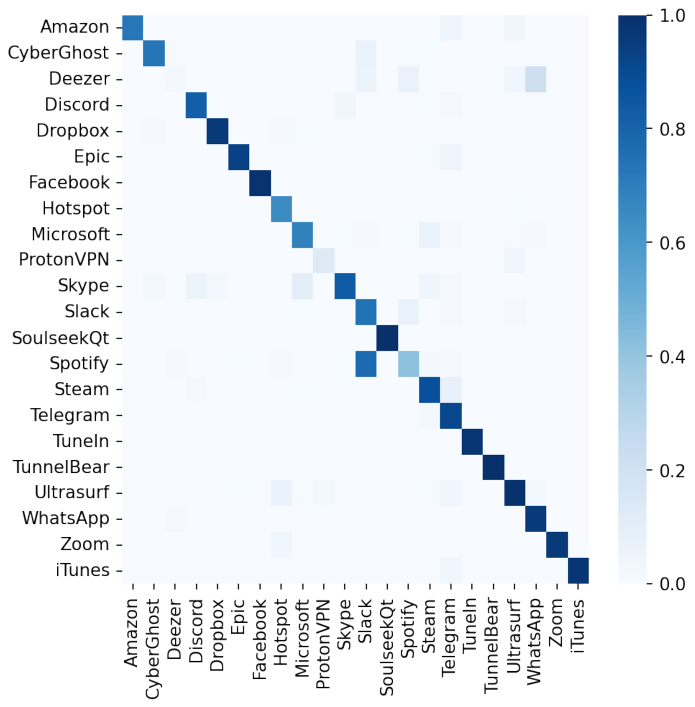

Figure 4.5. Confusion Matrix for Model with 3 Packets.

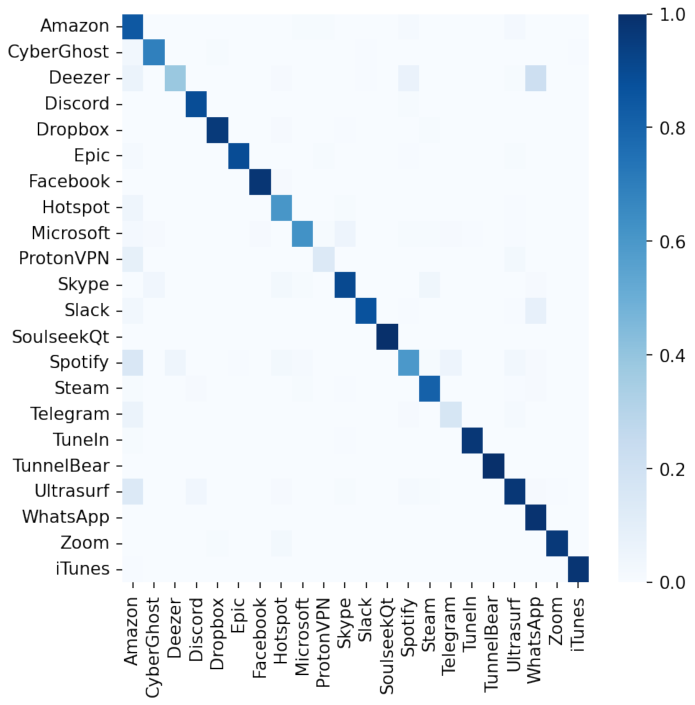

Figure 4.6. Confusion Matrix for Model with 4 Packets.

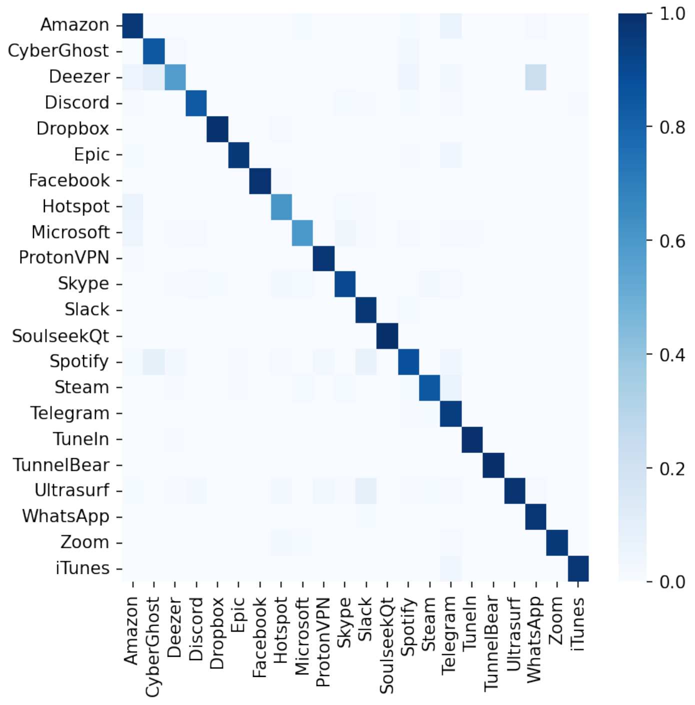

Figure 4.7. Confusion Matrix for Model with 5 Packets.

We applied majority voting starting with model whose input consist of three packets. Given majority voting mechanism in Table 4.3 uses the previous two models in addition to the current model. For example the results given as proposed classifier with voting for packet number 5, are acquired by using the majority voting between classifiers with packet numbers 3,4 and 5.

Table 4.4. Voting Scenarios.

Models used in Voting	Accuracy	Precision	Recall	F1 Score
1,2,3,4,5	0.964	0.966	0.964	0.961
1,2,3,4	0.925	0.93	0.925	0.918
2,3,4,5	0.965	0.969	0.965	0.963
1,3,5	0.956	0.962	0.956	0.954
2,4,5	0.968	0.970	0.968	0.963

Results listed in Table 4.3 and 4.4 implicate that using a voting mechanism beneficial for network classification task. It boosts the performance of given latest applicable models. In other words if we have first 5 packets from a network flow, it is more beneficial to use a voting strategy instead of simply relying the result of classifier with 5 packets as input. This performance gain comes from the decreasing of effects of packets with misleading payloads. This can be seen from the sankey diagram given in Figure 4.8. Figure 4.8 shows the overall accuracy of proposed classifier for different number of packets. If a flow is classified correctly then it belongs to "True" side if not then it is belong to "False" side. From Figure 4.8, we can see the number of correctly classified flows for different number of packets. It also shows the journey of a classification status of flows from one classifier to the next one. From the given sankey diagram, we can see that at the each level (each time an additional packet is used) status of some flows changes. Some flows join the false group and some flows join the true group. This exchange occurs at every stage, but as expected as the packet number increases, the size of false group and the size of true classified flows which join to the false group decreases. Indicating that using more packets will result in better performance wit respect to classifying flows as soon as possible without additional packets.

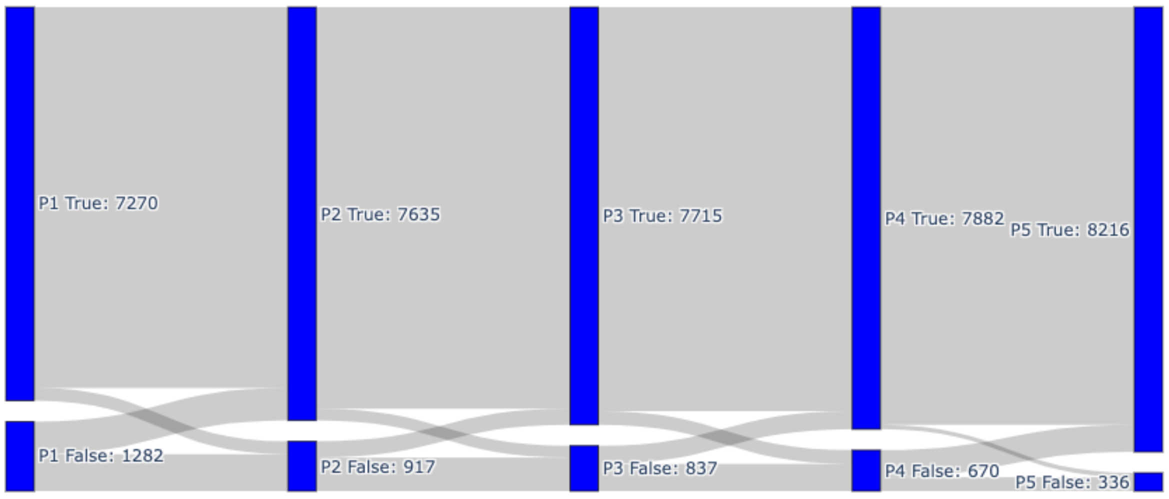

Figure 4.8. Overall Accuracy for different Packet Numbers for Application Based Network Traffic Dataset.

The same sankey diagram for classifier which employs voting starting from third stage is given in Figure 4.9. Comparing two diagrams reveal the advantages of using voting. The main difference between them are the size of the true predictions which turn into false at the next stage. We can argue that the latest packet from these flows misleads the classifier resulting in an error. By employing voting the size of this group can be reduced considerably which translates into better overall performance.

4.3.3. Private Dataset

We also tested the proposed approach on dataset we called as ”Private Dataset”. The results are given in Table 4.5. For this dataset, we applied the same majority voting strategy as in above using the previous two models in addition to the current model. The results verify the advantages of using proposed approach. The proposed classifier outperforms all baseline models and adding voting mechanism increases the performance showing the benefits of employing voting. As stated in section 4.1 this dataset is more challenging (both due to its smaller size and its increased class number) compared to the ”Application Based Network Traffic Dataset” and the overall results shows that. The sankey diagrams are given in Figure 4.10 and Figure 4.11.

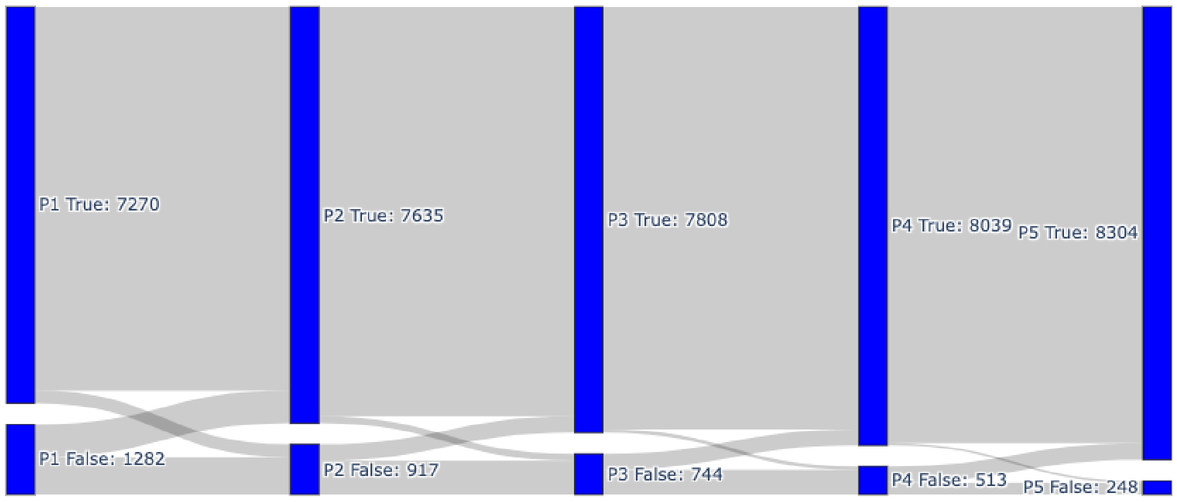

Figure 4.9. Overall Accuracy for different Packet Numbers using Voting for Application Based Network Traffic Dataset.

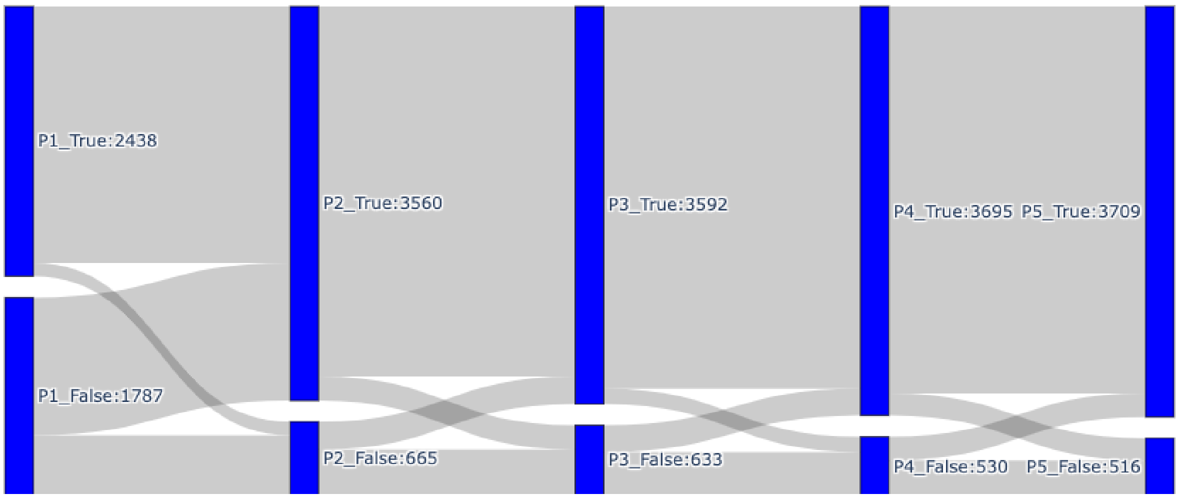

Figure 4.10. Overall Accuracy for different Packet Numbers for Private Dataset.

The effects of the proposed voting mechanism is much more visible in this case which can be attributed to more challenging characteristic of the ”Private Dataset”. The size of the true predictions which turn into false predictions at the next stage are reduced drastically. In light of these observations, we can argue that the employing voting becomes more crucial in cases with more classes, which are naturally more challenging to classify and make easier for classifier to be misled by new packets.

Using voting minimizes the errors attributed to misleading content of new packets. It is also more apparent that the even though this misclassification is avoided with voting significantly, the performance boost is not that significant. This is due to the stagnation resulting from majority voting. On the downside using voting anchors the predictions to their respective classes even though the content of the new packet contains necessary information for correct classification. But the benefits of using outweights this downside resulting in a overall better performance.

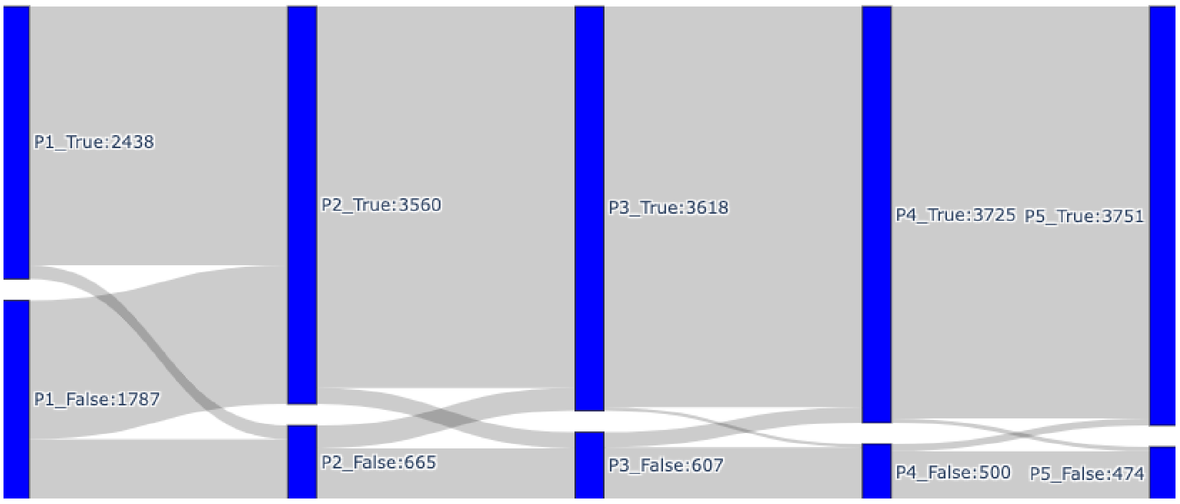

Figure 4.11. Classifier Accuracy for different Packet Number using Voting for Private Dataset.

Table 4.5. Classification Results for Private Dataset.

Model	Number of Packets	Metric			
		Accuracy	Precision	Recall	F1 Score
SVM	1	0.514	0.558	0.514	0.511
	2	0.744	0.724	0.744	0.732
	3	0.789	0.81	0.789	0.774
	4	0.796	0.824	0.796	0.782
	5	0.827	0.839	0.827	0.822
Random Forest	1	0.52	0.553	0.52	0.539
	2	0.672	0.659	0.672	0.662
	3	0.775	0.811	0.775	0.74
	4	0.783	0.823	0.783	0.775
	5	0.819	0.826	0.819	0.808
Logistic Regression	1	0.539	0.567	0.539	0.537
	2	0.762	0.769	0.762	0.753
	3	0.795	0.803	0.795	0.785
	4	0.832	0.841	0.832	0.829
	5	0.846	0.86	0.846	0.842
Proposed Classifier	1	0.578	0.581	0.578	0.563
	2	0.842	0.832	0.842	0.833
	3	0.85	0.862	0.85	0.852
	4	0.874	0.875	0.874	0.868
	5	0.877	0.879	0.877	0.873
Proposed Classifier with Voting	3	0.856	0.853	0.856	0.852
	4	0.882	0.88	0.882	0.878
	5	0.888	0.891	0.888	0.885

5. CONCLUSION

The classification of network flows is an important issue for the network management and it is crucial for ISPs that the classification should be done as soon as possible. The legacy port-based and DPI methods have shortcomings dealing with the dynamic ports and encrypted content. Although statistical methods can deal both of these problems, its slowness pose a problem. The new DPI based methods employs deep learning based approaches to deal with encrypted content. Using deep learning based approaches help to overcome the challenge posed by encrypted content.

This thesis proposes a payload based multi-phase classifier which employs voting mechanism to deliver fast and accurate predictions. In this context, by fast we refer to using minimum amount of packets without waiting more packets to be transmitted in network. To classify flows based on the payloads carried by the packets, a deep learning architecture employing CNN and RNN is proposed and to enhance the performance of proposed architecture majority voting is employed. The payload from the packets are vectorized and fed into the proposed architecture in parallel as input. The proposed architecture process these payloads vectors in parallel by using convolutional layers to extract payload features. These payload features are further fed into a LSTM to extract flow features. Fully connected layers use extracted flow features to classify flows.

The proposed classifier is tested against baseline models using two datasets, one is a public dataset available at kaggle [37] and the other one is a private dataset created by a private company. In both datasets, the proposed architecture outperforms the all baseline models (SVM, RandomForest and Logistic Regression) for different number of input packets. In addition, the proposed classifier with voting mechanism outperforms the proposed architecture. To find optimal payload length different lengths are tested. It is seen that the increasing payload length increases the model performance but after a certain number performance gain in minimal indicating a diminishing return. For all models increasing the number of packets, increases the model performance. The performance boost of voting mechanism can be attributed to the prevention of new mistakes due to latest packets. This is shown clearly in sankey diagrams given in Section 4. The benefits of using voting mechanism can be seen by comparing the size of the true predictions which turn into false prediction at the next stage.

As for the future works, we plan to test this method in larger datasets with more classes and another aspect we didn't consider but plan to is the unknown class. It is impractical to limit the number of applications in a network. To achieve a good performance in real life, the proposed classifier should be able to handle network flows belonging to the unknown or new applications. For this purpose a threshold can be introduced which can help to detect flows belonging to the unknown class.

REFERENCES

1. Wikipedia, “IPv4”, `https://en.wikipedia.org/wiki/IPv4`, accessed on July 12, 2022.

2. Wikipedia, “Autoencoder”, `https://en.wikipedia.org/wiki/Autoencoder`, accessed on July 15, 2022.

3. Wikipedia, “Artificial Neuron”, `https://en.wikipedia.org/wiki/Artificial_neuron`, accessed on July 15, 2022.

4. Wikipedia, “Recurrent Neural Network”, `https://en.wikipedia.org/wiki/Recurrent_neural_network`, accessed on July 15, 2022.

5. Wikipedia, “Long Short-term Memory”, `https://en.wikipedia.org/wiki/Long_short-term_memory`, accessed on July 15, 2022.

6. Silver, B., “Netman: A Learning Network Traffic Controller”, *Proceedings of the 3rd International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, Charleston, South Carolina, USA, 1990.

7. Crotti, M., M. Dusi, F. Gringoli and L. Salgarelli, “Traffic Classification through Simple Statistical Fingerprinting”, *Computer Communication Review*, Vol. 37, No. 1, pp. 5–16, 2007.

8. Hjelmvik, E. and W. John, “Statistical Protocol Identification with SPID: Preliminary Results”, *Swedish National Computer Networking Workshop*, Uppsala, Sweden, 2009.

9. Zuev, D. and A. W. Moore, “Traffic Classification Using a Statistical Approach”, *Passive and Active Network Measurement*, Berlin, Germany, 2005.

10. Soysal, M. and E. G. Schmidt, “Machine Learning Algorithms for Accurate Flow-

based Network Traffic Classification: Evaluation and Comparison", *Performance Evaluation*, Vol. 67, No. 6, pp. 451–467, 2010.

11. Shafiq, M., X. Yu, A. A. Laghari, L. Yao, N. K. Karn and F. Abdessamia, "Network Traffic Classification Techniques and Comparative Analysis using Machine Learning Algorithms", *International Conference on Computer and Communications*, Chengdu, China, 2016.

12. Lopez-Martin, M., B. Carro, A. Sanchez-Esguevillas and J. Lloret, "Network Traffic Classifier With Convolutional and Recurrent Neural Networks for Internet of Things", *IEEE Access*, Vol. 5, pp. 18042–18050, 2017.

13. Zander, S., T. Nguyen and G. Armitage, "Self-Learning IP Traffic Classification Based on Statistical Flow Characteristics", *Passive and Active Network Measurement*, Berlin, Germany, 2005.

14. Zhang, J., Y. Xiang, W. Zhou and Y. Wang, "Unsupervised Traffic Classification Using Flow Statistical Properties and IP Packet Payload", *Journal of Computer and System Sciences*, Vol. 79, No. 5, pp. 573–585, 2013.

15. Vlăduţu, A., D. Comăneci and C. Dobre, "Internet Traffic Classification based on Flows Statistical Properties with Machine Learning", *International Journal of Network Management*, Vol. 27, No. 3, p. 1929, 2017.

16. Dhote, Y., S. Agrawal and A. J. Deen, "Self-Learning IP Traffic Classification Based on Statistical Flow Characteristics", *International Conference on Computational Intelligence and Communication Networks*, Jabalpur, India, 2015.

17. Moore, A. W. and K. Papagiannaki, "Toward the Accurate Identification of Network Applications", *Passive and Active Network Measurement*, Berlin, Germany, 2005.

18. Haffner, P., S. Sen, O. Spatscheck and D. Wang, "ACAS: Automated Construction

of Application Signatures", *Proceedings of the Association for Computing Machinery Special Interest Group on Data Communications (SIGCOMM) Workshop on Mining Network Data*, Philadelphia, Pennsylvania, USA, 2005.

19. Lotfollahi, M., R. S. H. Zade, M. J. Siavoshani and M. Saberian, "Deep packet: a novel approach for encrypted traffic classification using deep learning", *Soft Computing*, Vol. 24, No. 3, pp. 1999–2012, 2020.

20. Habibi Lashkari, A., G. Draper Gil, M. Mamun and A. Ghorbani, "Characterization of Encrypted and VPN Traffic Using Time-Related Features", *The International Conference on Information Systems Security and Privacy (ICISSP)*, Rome, Italy, 2016.

21. Lim, H.-K., J.-B. Kim, J.-S. Heo, K. Kim, Y.-G. Hong and Y.-H. Han, "Packet-based Network Traffic Classification Using Deep Learning", *International Conference on Artificial Intelligence in Information and Communication (ICAIIC)*, Okinawa, Japan, 2019.

22. Dong, C., C. Zhang, Z. Lu, B. Liu and B. Jiang, "CETAnalytics: Comprehensive Effective Traffic Information Analytics for Encrypted Traffic Classification", *Computer Networks*, Vol. 176, p. 107258, 2020.

23. Zhang, Z., L. Liu, X. Lu, Z. Yan and H. Li, "Encrypted Network Traffic Classification: A data driven approach", *International Conference on Parallel Distributed Processing with Applications, Big Data Cloud Computing, Sustainable Computing Communications, Social Computing Networking*, Exeter, UK, 2020.

24. Stallings, W., *Business Data Communications*, Pearson, New York, 2013.

25. Sammut, C. and G. I. Webb, *Encyclopedia of Machine Learning*, Springer US, Boston, MA, 2010.

26. Bengio, Y., A. C. Courville and P. Vincent, "Unsupervised Feature Learning and

Deep Learning: A Review and New Perspectives", ArXiv:1206.5538 [cs], 2012.

27. Mikolov, T., K. Chen, G. Corrado and J. Dean, "Efficient Estimation of Word Representations in Vector Space", *International Conference on Learning Representations (ICLR)*, Arizona, USA, 2013.

28. LeCun, Y., Y. Bengio and G. Hinton, "Deep learning", *Nature*, Vol. 521, No. 7553, pp. 436–444, 2015.

29. Freund, Y. and R. E. Schapire, "Large Margin Classification Using the Perceptron Algorithm", *Machine Learning*, Vol. 37, No. 3, pp. 277–296, 199.

30. McCulloch, W. S. and W. Pitts, "A Logical Calculus of the Ideas Immanent in Nervous Activity", *The Bulletin of Mathematical Biophysics*, Vol. 5, pp. 115–133, 1943.

31. Krizhevsky, A., I. Sutskever and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks", *Association for Computing Machinery*, Vol. 60, No. 6, pp. 84–90, 2017.

32. Kim, Y., "Convolutional Neural Networks for Sentence Classification", ArXiv:1508.5882 [cs], 2014.

33. Hochreiter, S. and J. Schmidhuber, "Long Short-Term Memory", *Neural Computation*, Vol. 9, No. 8, pp. 1735–1780, 1997.

34. Cho, K., B. van Merrienboer, D. Bahdanau and Y. Bengio, "On the Properties of Neural Machine Translation: Encoder-Decoder Approaches", ArXiv:1409.1259 [cs], 2014.

35. Zhang, J., Y. Xiang, W. Zhou and Y. Wang, "APAD: Autoencoder-based Payload Anomaly Detection for industrial IoE", *Applied Soft Computing*, Vol. 88, p. 106017, 2020.

36. Wang, W., M. Zhu, X. Zeng, X. Ye and Y. Sheng, "Malware Traffic Classification Using Convolutional Neural Network for Representation Learning", *International Conference on Information Networking (ICOIN)*, Da Nang, Vietnam, 2017.

37. Kaggle, "Application Based Network Traffic Dataset", `https://www.kaggle.com/applicationdataset/applicationbasednetworktrafficdataset`, accessed on February 21, 2022.

38. Karayaka, M., A. Bayer, S. Balki, E. Anarim and M. Koca, "Application Based Network Traffic Dataset and SPID Analysis", *Signal Processing and Communications Applications Conference (SIU)*, Karabük, Turkey, 2022.

39. Kingma, D. P. and J. Ba, "Adam: A Method for Stochastic Optimization", ArXiv:1412.6980 [cs], 2014.

40. Chollet, F., "Keras", `hhttps://github.com/keras-team/keras`, accessed on July 15, 2022.