CAPC: Packet-Based Network Service Classifier
With Convolutional Autoencoder

KAI-CHENG CHIU, CHIEN-CHANG LIU, AND LI-DER CHOU, (Member, IEEE)
Department of Computer Science and Information Engineering, National Central University, Taoyuan 320, Taiwan
Corresponding author: Li-Der Chou (cld@csie.ncu.edu.tw)

This work was supported in part by the Ministry of Science and Technology (MOST), Taiwan, under Grant MOST 105-2221-E-008-071-MY3 and Grant MOST 108-2221-E-008-033-MY3.

ABSTRACT  The Internet has been evolving from a traditional mechanism to a modern service-oriented architecture, such as quality-of-service (QoS) policies, to meet users’ various requirements for high service quality. An instant and effective network traffic classification method is indispensable to identify network services to enforce QoS policies on the corresponding service. Network managers can easily flexibly deploy traffic classification modules and configure the network policies with the help of the emerging software-defined networking. However, most existing traffic classification solutions, such as port-based methods or deep packet inspection, cannot handle real-time and encrypted traffic classification. In this research, a Convolutional Autoencoder Packet Classifier (CAPC) has been proposed to immediately classify incoming packets in fine-grained and coarse-grained manners, that is, classifying a service to a single application and a rough genre, respectively. The CAPC is a packet-based deep learning model consisting of a 1D convolutional neural network and an autoencoder, which can handle dynamic-port and encrypted traffic and even cluster similar applications. This classifier is verified on not only the private self-captured traffic but also a public VPN dataset to demonstrate its performance. Moreover, the CAPC classifies different types of service traffic with an accuracy of over 99.9% on the private dataset of 16 services and over 97% on the public dataset of 24 services, thereby outperforming other deep learning classifiers. Experimental results also show other performance metrics, including stability, average precision, and recall and the highest F1-score values of 15 and 18 services on the private and public datasets, respectively.

INDEX TERMS  Autoencoder, deep learning, one-dimensional convolutional neural network, packet-based traffic classification.

I. INTRODUCTION
With the advancement of the diversity of network services, network functions demanded by users are not limited to data transmission, but obtainment of a low-latency response within a few milliseconds (e.g., voice-over-IP [VoIP]) or network services with a low packet loss rate (e.g. text message) [1]. A real-time traffic classifier is necessary to immediately process or forward different service traffic. However, the boundaries between services are fairly obscured; consequently, the identification of services merely by using simple rules or fixed patterns is difficult.

In recent years, machine learning (ML) methods have been widely used in the field of networks [2]. ML [3] has become a promising technique for traffic classification [4], [5].

The associate editor coordinating the review of this manuscript and approving it for publication was Wai-Keung Fung.

The ML-based methods can find hidden features within packets or traffic flow of applications instead of identifying network traffic by predefined patterns as deep packet inspection (DPI). The convergence time of a neural network [6] model in ML has been notably decreased due to the rapid evolution of graphic processing unit (GPU), thereby significantly accelerating parallel computing. Specifically, the deep learning paradigm in a neural network is widely deployed in various fields. Numerous deep learning models have been proposed, such as convolutional neural network (CNN) utilized in image processing [7], [8] and recurrent neural network [9], [10] applied in semantic analysis, have been proposed.

As an essential component of deep learning, CNN is especially widely used in image recognition, even surpassing human beings. The success of CNN in image recognition inspired us to exploit the characteristics of CNN to classify network traffic. The applicability of any deep learning
models depends on the scenario and input data. Accordingly, the data type of network packets is quite similar to images. Packets and images are composed of byte values ranging from 0 to 255, and the primary differences between them are merely the data sizes and dimensions. In terms of data size, the payload of an ethernet frame is at most 1500 B, according to the ethernet maximum transmission unit (MTU). The size of an image relies on its resolution, compression format, and coloring. With regard to data dimensionality, a packet is a 1D byte sequence, and an image is comprised of 2D pixels.

Deep autoencoders (DAEs) are often used for unsupervised learning or dimensionality reduction. We employ autoencoding in our task because the input scale is relatively large for packets. A general deep learning model arbitrarily initializes the parameters. Meanwhile, different randomized parameters in every initialization may lead to an unstable performance. Accordingly, autoencoding in this work is used for encoding the input packet to extract the important information among all packet bytes. The autoencoder is a more powerful encoder than before when combined with a 1D convolutional layer. This autoencoder uses the properties of CNN to effectively extract information. To our knowledge, this network structure is currently not applied in other traffic classification studies. The encoder is also a good network structure containing well-initialized parameters. After the encoder is constructed and trained, it would be concatenated with a deep neural network (DNN) classifier to classify packets. On the basis of the above-mentioned ideas, we propose a Convolutional Autoencoder Packet Classifier (CAPC) model to effectively differentiate packets of one service from another. CAPC uses CNN’s local perception, and DAE dimensionality reduction and parameter initialization to classify the packet into a certain type of service, which improves the classification accuracy. The classification target datasets contain the self-captured traffic [11] and the open virtual private network (VPN) dataset “ISCX VPN-nonVPN” [12]. VPN is a communication method commonly used to connect to private networks. It uses a tunnel protocol to achieve message communication method commonly used to connect to private networks. The size of an image relies on its resolution, compression format, and coloring. With regard to data dimensionality, a packet is a 1D byte sequence, and an image is comprised of 2D pixels. The private and public datasets are used for model training and performance evaluation.

The overall process of packet data preprocessing is demonstrated, including deidentification of addresses in a packet, thereby stabilizing the performance of models. The private and public datasets are used for model training and performance evaluation.

The overall process of packet data preprocessing is demonstrated, including deidentification of addresses in a packet, thereby stabilizing the performance of models. Experiments are presented to show several performance metrics, including accuracy, F1-score, precision, recall, and stability. The classification time is also evaluated to show the feasibility to deploy the model in a real case.

The remainder of the paper is organized as follows. Related work is discussed in Section II. The proposed data preprocessing scheme and traffic classification model are illustrated in Section III. Experimental results are evaluated in Section IV. Finally, conclusions are provided in Section V.

II. RELATED WORK

The CAPC, a supervised learning model, demands labeled service packets as training data to train the deep learning model. Fig. 1 shows that two approaches can be categorized depending on the training data type: flow-based and packet-based [13].

Flow-based approaches reply on statistical or time series features. A set of features constitutes a flow, including flow duration, packet inter arrival time, maximum packet length, etc. Iliofotou et al. [14] proposed a graph-based approach which uses Traffic Dispersion Graph (TDG) and selects crucial graph metrics to identify the P2P applications. Before the classification, the proposed method clusters similar flows together as a group by their flow-based features (duration, packet sizes, etc.) and constructs a TDG for a group. Besides, several ML algorithms, such as support vector machine [15], [16], decision tree [17], [18], and neural network [19]–[21], are also applied with flow features. Flow-based methods do not check packet payload, but they analyze statistical features, wherefore even encrypted traffic can be recognized. In summary, the advantages of flow-based methods are as follows: can identify encrypted packets and learn the relationship between packets in a flow. However, several limitations still persist: (1) Flow features are difficult to gather. (2) The selection of features heavily depends on expert experience. (3) At least several milliseconds are required to form a flow from service traffic.

Packet-based approaches classify every single packet within service traffic and take every byte in a packet as a feature. The input dimensionality is relatively larger than a flow-based approach because of the data format. Deep learning is adopted in such cases [22]–[24] because it automatically extracts features from raw packets without human intervention, thereby making itself a promising approach for traffic classification. More discussions about the packet-based classification studies are presented at the penultimate paragraph of this section. Specifically, CNN is a prospective model to tackle raw packet data [25]–[27] owing to its properties, such as local perception, shared parameters,
and pooling [28]. Local perception and shared parameters are achieved by convolutional layers to filter several local patterns from all input features. Pooling is implemented by the pooling layer to reduce the dimensionality of results from convolution and retain vital information. Packet-based approaches do not require a period to collect packets to form a flow and are more efficient than flow-based methods. Despite the aforementioned benefits, packet-based approaches are unable to consider the relationship between packets in service traffic, which merely analyzes the payload of a packet. Under such circumstance, more packet data are required to extract characteristics from a certain service. Aside from distinguishing by the data format, references [29] and [30] also reported that traffic classification methods can be roughly categorized into three types, namely, port-based, payload-based, and ML-based methods (summarized in Table 1).

### TABLE 1. Overview of traffic classification approaches.

| Method               | Description                                                                 | Advantages                                      | Disadvantages                                      |
|----------------------|------------------------------------------------------------------------------|-------------------------------------------------|----------------------------------------------------|
| Port-based methods   | Classify packets by port numbers, such as TCP port 80 for the HTTP protocol. | Instant classification, High accuracy           | Different services using the same port, Dynamic port number utilization |
| Payload-based methods| Examine payload in the application layer with known signatures of protocols [35] [36] | Handle services with dynamic ports               | High computational cost, Encrypted data, Novel applications |
| ML-based methods     | Extract features from packet payload or statistical characteristics with an ML model [18] [47] | Handle dynamic ports and encrypted data, Cluster similar applications | Longer classification time compared with port-based methods, Training data |

Port-based methods were commonly used in the past; specifically, when most communication protocols utilized fixed well-known TCP/UDP port numbers [31] given by the Internet Assigned Numbers Authority [32]. However, several limitations have emerged. Port-based methods are no longer as effective as before [33], [34] because an increasing number of applications adopt dynamic or shared port numbers (e.g., Skype can be configured to communicate through custom port numbers, and YouTube utilizes port 443 as other HTTPS applications). Payload-based methods inspect packet headers and payload with predefined patterns to identify a service, also known as DPI [35]. Such methods provide high accuracy, can deal with dynamic port numbers, and widely used in researches [36]–[38]. However, such methods consume high computational resource. The packet signature library must also be updated and cannot tackle encrypted traffic.

To surpass the limits of traditional classification methods, more and more researches have been exploring the potential of ML to deal with encrypted or more complex network traffic in the future. Several studies have proposed their own feature selection scheme for numerous flow features. Reference [15] used principal component analysis and genetic algorithm to select critical features. Meanwhile, FWFS is specifically designed to conduct feature selection among 248 flow features for C4.5 algorithm to classify flows [18]. References [39]–[41] utilized random forest algorithm to randomly select features to obtain an optimal result. Information gain ratio can be a criterion in choosing features [42] for k-nearest neighbor algorithm. A neural network-based classifier can automatically update internal network parameters to increase the weights corresponding to significant flow features [43], [44]. The neural network model is also applied in anomaly detection and intrusion detection system (IDS) [45], [46] apart from the ordinary service traffic classification.

The deep learning method is a prospective solution when raw packets are taken as input data; even the hidden relationship between bytes of encrypted traffic can be discovered through the high-complexity computation [22], [47]–[49]. References [22] and [47] classified the “ISCX VPN-nonVPN” open VPN dataset [12] in a fine-grained manner, and the others [48], [49] did so in a coarse-grained manner. Lotfollahi et al. [22] used stacked autoencoder (SAE) and CNN models to classify encrypted traffic and train the models without needless packets, such as TCP connection establishment packets. Wang et al. [47] also DNN, SAE, and CNN models for the dataset and compared their performance. Wang et al. [48] proposed a 1D CNN model and tested different network structures. The Xu et al. [49] considered the time series factor; thus, the proposed methods, including CNN and long short-term memory (LSTM), are ensemble learning structures. The VPN encrypted dataset [12] was widely used for experiments in the studies discussed above, but it actually contains 24 services, whereas existing works merely took a part of the dataset for experiments. Mobile traffic has accounted for a large portion of the overall network traffic due to the advancement of mobile service diversity; accordingly, the classification of mobile services has become an attractive topic and is deeply discussed in several literatures [13], [41], [50]. Besides the separate application of flow-based and packet-based methods, another perspective is to combine the two to obtain an improved performance [51]. This mechanism uses SAE, CNN, and LSTM to verify the idea on the VPN-nonVPN dataset.

In this work, we design and examine the CAPC model composed of a 1D convolutional autoencoder and a DNN classifier. Subsequently, we compare the final performance with pure DNN, DAE, and CNN network structures. Self-captured traffic of 16 selected services and the open VPN-nonVPN traffic of 24 services are used for training. The fine-grained and coarse-grained classification schemes are proposed to verify the effectiveness and robustness of CAPC. The trained model can be deployed in a practical SDN environment to place CAPC in the control plane. When a new service packet is encapsulated and sent to the controller, CAPC can immediately classify which service the packet
belongs to. Thereafter, a service pool can be created to manage all the service traffic transmitted within the network, such as dynamic path decision, quality of service enforcement, and prevention of anomaly traffic.

### III. PROPOSED METHOD

Few procedures must be carried out in advance to construct a deep learning packet classifier: data collection, data preprocessing, and model training. The description of each procedure is shown in Fig. 2. First, we need to obtain self-collected private traffic of various services, such as YouTube, Skype, and LINE. Second, the collected traffic in raw format must be converted into computable data and normalized. Every record should share the same input dimensionality. Finally, we design and construct the model, set the hyperparameters, and train the model with the normalized dataset.

![FIGURE 2. Traffic classification categories according to the data format.](image)

#### A. DATA COLLECTION

Before training, we have to collect service traffic and attach a label to each packet according to the service type. The first step is to determine which services should be gathered. Accordingly, the traffic of the following four main categories is targeted: file transfer, video streaming, VoIP, and remote control. Each main category contains four applications, totally 16 services, and the correspondence of categories and services is listed in Table 2.

![TABLE 2. Description of self-captured service traffic.](image)

| File Transfer | Video Streaming | VoIP | Remote Control |
|---------------|-----------------|------|----------------|
| FTP           | RTP / MPEG Transport Stream video | LINE VoIP | RDP (Windows) |
| SFTP          | RTSP video      | Skype VoIP | VMware |
| OneDrive      | UDP (legacy) video | Zoom VoIP | XenServer |
| SCP           | YouTube video (GQUIC) | Join.me VoIP | NCU cloud |

Among the above-mentioned services, FTP, SFTP, and SCP traffic were generated by self-established file servers. One client host was in charge of requesting for the files on the file servers, and the transmitted packets were captured. RTP/MPEG Transport Stream, RTSP, and UDP (legacy) video traffic were produced by the VLC streaming toolkits, from which there were still several other protocols to choose (e.g. HTTP, MS-WMSP). The captured YouTube traffic was transmitted by the GQUIC protocol over UDP. The traffic of other services was obtained by capturing the corresponding commercial software traffic. All of the VoIP services transmitted their voice data over UDP, and the remote-control services communicated over TCP. To generate such traffic, we setup up two clients for every VoIP service, and launched a phone call on one client to another. The traffic of such services was captured during the phone call. As for remote control traffic, we captured the traffic of VMware and XenServer in our lab, and created the sessions of Windows RDP and NCU cloud service (provided by our university). We need to follow a certain process to save the traffic of every application, such as a PCAP or PCAPNG format file [52]. A PCAP file contains the arriving time, headers, and payload of each packet and can be read by open-source libraries. To show the authenticity of the traffic, the captured traffic data were uploaded to IEEE DataPort for open access [11].

The entire process of capturing the traffic of a service is shown in Fig. 3. The notations used in the figure are narrated in Table 3. Wireshark [53] is used in this research to capture service traffic. Fig. 3 demonstrates that the objective in $r_{mfc}(i)$ is to capture $f_{tfc}(i)$ of svc, The IP address in every captured packet $p_{k^tfc}(i)$ would be checked to determine if that packet belongs to svc, or not. If the packet belongs to svc, then it would be collected to $f_{tfc}$ (i); otherwise, it would be discarded.

![FIGURE 3. Flow chart of network service traffic capturing.](image)
TABLE 3. Notation table of network service traffic capturing.

| Notation | Description |
|----------|-------------|
| svc\(_i\) | \(i^{th}\) service |
| tf\(_{svc}\)(i) | Collection of byte in every packet of svc\(_i\) |
| rnd\(_{svc}\)(i) | \(i^{th}\) round to record tf\(_{svc}\)(i) |
| pcap\(_{svc}\)(i) | Recorded PCAP file of tf\(_{svc}\)(i) |
| txt\(_{svc}\)(i) | Hex dump file of pcap\(_{svc}\)(i) in text format |
| pkt\(_{svc}\)(i) | \(i^{th}\) captured packet in rnd\(_{svc}\)(i) |
| IP\(_{svc}\)(i) | IP address of svc\(_i\) |
| MAC\(_{pkt}\)(j,i) | Source MAC address of pkt\(_{j}^{\text{svc}}\)(i) |
| MAC\(_{pkt}\)(j,i) | Destination MAC address of pkt\(_{j}^{\text{svc}}\)(i) |
| IP\(_{pkt}\)(j,i) | Source IP address of pkt\(_{j}^{\text{svc}}\)(i) |
| IP\(_{pkt}\)(j,i) | Destination IP address of pkt\(_{j}^{\text{svc}}\)(i) |

The addresses in every packet would be modified to fixed values to ensure that the classifier actually classifies the traffic by packet content. By doing so, our classifier will not directly classify traffic by IP or MAC addresses. When the above-mentioned procedures are completed, the tf\(_{svc}\)(i) would be saved as pcap\(_{svc}\)(i), and the hex of pcap\(_{svc}\)(i) in text format would be converted to txt\(_{svc}\)(i).

B. DATA PREPROCESSING

Preprocessing is necessary to allow the deep learning model to read and compute the input packet data. Two main problems must be solved. First, the ML models require that the input data be kept in the fixed dimensionality, while every packet may appear in different lengths. Second, neural networks are sensitive to input data scale; thus, every byte in a packet should be scaled from 0–255 to 0–1 as (1) to ensure that every byte (feature) have the same influence on the model. The notations in the equation are listed in Table 4, and the overall process of data preprocessing is shown in Fig. 4.

\[
byte_{k}^{\text{norm}}(j, i) = \frac{byte_{k}^{\text{rec}}(j, i)}{255} \tag{1}
\]

Fig. 4 demonstrates that hex dump text files of every service would be read and checked if the length len\(_{rec}\)(j, i) of a packet rec\(_{j}^{\text{svc}}\)(i) is less than 1500 (ethernet MTU) or not. If the requirement is met, then ‘0 × 00’s will be padded after rec\(_{j}^{\text{svc}}\)(i), until the length is equal to 1500. Otherwise, extra bytes of rec\(_{j}^{\text{svc}}\)(i) will be truncated, until the length is equal to 1500. Fig. 5 illustrates the padding and truncation of a packet to reshape every packet in a fixed length.

After the packet length modification, every byte in a packet should be scaled from 0–255 to 0–1 in proportion, because a raw byte is ranged from 0 to 255. Accordingly, deep learning models can obtain stable results. Fig. 6 is the result of Fig. 5 after value scaling. For example, the first byte byte\(_{1}^{\text{rec}}\)(j, i) in the first packet rec\(_{j}^{\text{svc}}\)(i) is originally 0 × 69 (105 in decimal), while it becomes 0.41 in Fig. 6 after proportional scaling.

Finally, the normalized data would be saved as txt\(_{tfc}\)(i) for models to read and train.

C. MODEL TRAINING

We can construct the packet classifier with the obtained dataset after preprocessing on the basis of an autoencoder, a 1D convolution, and a DNN. As a packet-based classifier, CAPC does not need time to gather flow features, and can be directly applied with the obtained packet bytes as shown in Fig. 1. The autoencoder encodes large-scale input data into...
few essential features. This network is widely used in unsupervised learning tasks. A typical autoencoder is composed of an encoder and a decoder. The former is in charge of condensing the large-scale input information. The latter is responsible for recovering the original information from the compact data. DAE, which is a combination of an autoencoder and a DNN, has been widely applied in packet processing [22], [47], [41] and obtain optimal results. The autoencoder can be also combined with convolutional layers. Accordingly, the autoencoder is robust in extracting the essence from input packets. Lastly, the DNN classifier can be applied to take the outcome of the encoder as input to classify data. In this research, the process to create the proposed CAPC can be separated into two parts: encoder and classifier training.

1) ENCODER TRAINING
The autoencoder in CAPC is composed of convolutional and pooling layers. The structure of CAPC is shown in Fig. 7. The input layer includes 1500 neurons because every packet has been normalized to the fixed length. Three layers are concatenated after the input layer, constituting a so-called encoder. Each layer is comprised of a 1D convolutional layer, followed by a ReLU activation function (2). A max-pooling layer (3) is applied behind the activation function. The encoding layers 1, 2 and 3 all include a one-dimensional CNN layers (1D-CNN) with 128, 64, and 32 filters respectively. In contrast, the decoding layers 1, 2 and 3 all contain a 1D-CNN with 32, 64, and 128 filters, respectively.

The computation of 1D convolution is depicted in Fig. 8. In the figure, $x_1$ stands for the first input value, and $w_{f,1}^j$ represents the first weight of the filter $f$ in the $j$th layer. The number of a filter in this case is three. Accordingly, the filter will perform an inner product on every three adjacent values among all input values. After the inner product, a bias value $b_{f,1}^j$ of filter $f$ in the $j$th layer will be added to the outcome. Then, the outcome value would be applied to ReLU (2), thereby making the value nonlinear and turning any negative values to zero to keep the important information. Once the above-mentioned procedures have been completed, an obtained value $fm_{f,n}^j$ represents the $n$th value in the feature map of $f$. We can obtain an entire feature map consisting of $fm_{f,n}^j$ because all input values have been applied with $f$.

After the convolution, the obtained feature maps would be subsampled by a max-pooling layer (3) and is depicted in Fig. 9. A 1D max-pooling layer takes every two values in a feature map as input, keeps the greater value, and discards the smaller one. The purpose of the operation is to filter out nonessential data among $n$ values, which are the smaller ones, and keep the critical data as the greater one. The $n$ in this case is set to two. Fig. 9 shows that the feature map generated by a convolution consists of a number of values as $fm_{f,n}^j$, and is followed by a max-pooling process. The greater value is kept as $mp_{f,n}^j$ for further computation in the next layer.

The encoder in CAPC involves three encoding layers, and the last layer is a max-pooling layer. The neural network concatenated after the encoder is the decoder, the structure of which is opposite to the encoder. The output dimensionality of an autoencoder must be same as the input. The targeted values of the output layer are set to be same as the input values. The first layer of the decoder must be started with an up-sampling layer in contrast with the max-pooling layer as the end of the encoder, to create a reverse network of the encoder. The operation of up-sampling is given by (4) and
depicted in Fig. 10. Up-sampling will double the number of values by duplicating each value into two because max-pooling reduces the values in half. The $mp_{j,a}^l$ values obtained after max-pooling in Fig. 10 are applied to up-sampling, and we can get the output values as $w^{l+1}_{j,a}$.

$$UpSampling(x) = [x, x]$$ (4)

FIGURE 10. 1D up-sampling operation diagram.

After up-sampling, the 1D convolution computation will be operated and repeated for three layers in contrast with the encoder. In the last activation function of the decoder, sigmoid is used to produce the output, thereby making the value between zero and one as (5). To date, the autoencoder has been established, and it is ready for training. The training target is set to the original input. The goal of the encoder has been established, and it is ready for training. The training value between zero and one as (5). To date, the autoencoder is used to produce the output, thereby making the sigmoid the encoder. In the last activation function of the decoder, be operated and repeated for three layers in contrast with FIGURE 10.

Up-sampling will double the number of values by duplicating each value into two because max-pooling reduces the values in half. The $mp_{j,a}^l$ values obtained after max-pooling in Fig. 10 are applied to up-sampling, and we can get the output values as $w^{l+1}_{j,a}$.

$$UpSampling(x) = [x, x]$$ (4)

FIGURE 10. 1D up-sampling operation diagram.

After up-sampling, the 1D convolution computation will be operated and repeated for three layers in contrast with the encoder. In the last activation function of the decoder, sigmoid is used to produce the output, thereby making the value between zero and one as (5). To date, the autoencoder has been established, and it is ready for training. The training target is set to the original input. The goal of the encoder and decoder is to approximate the input values to the output values through updating their internal parameters by gradient descent (6). A criterion is needed to judge how good the autoencoder is to update the parameters. The criterion is known as the loss function and set as mean square error (7).

$$sigmoid(z) = \frac{1}{1+e^{-z}},$$ (5)

where $w$ is weight, and $b$ is bias in the model, and they are updated by subtracting the partial differentials timed by a learning rate $\eta$ of each value to lower the loss function value; $m$ is the number of training records; $\hat{y}_i$ is the targeted value of a record; $y_i$ is the output value of the record calculated by the model; and $MSE$ stands for the error between the input and the output. Once the loss function value converges and barely decreases, the training automatically ceases. Afterward, the first three encoding layers are extracted from the autoencoder as the encoder. The remaining encoding layers are discarded.

2) CLASSIFIER TRAINING

A DNN classifier will be concatenated after the encoder with the pretrained convolutional 1D encoder (Fig. 11). Hence, the encoder can be regarded as a well-initialized network architecture that provides a good starting point for a DNN classifier to categorize services.

The connected layer of the encoder and the classifier is a 1D convolutional layer and is flattened for the input of the classifier. The encoded values are taken as the input, and the classifier is built up of three fully connected layers containing 64, 32, and 16 neurons, respectively. A layer is composed of numerous neurons. The value of every neuron can be computed as (8), in which the output of $f(x)$ is the value of a targeted neuron, and $x_i$ is a neuron in the previous layer. Every $x$ is multiplied by a weight $\omega_i$ and added by a bias $b_i$. The outcome value would be applied to an activation function to achieve a nonlinear output. The activation function can be sigmoid (5) or ReLU (2) in this case.

$$f(x) = \sigma \left( \sum_i \omega_i \cdot x_i + b_i \right)$$ (8)

A DNN model is constituted of several layers of neurons, taking the value of a neuron computed by (8) as the input to produce another neuron. The values of layer $l$ are generated by the previous layer $l-1$ calculated by (9). $l$ is the $j^{th}$ layer and also a 1D vector, while $W^{(j-1)}$ is a 2D matrix standing for weights. $W^{(j-1)}$ is applied by $f^{(j-1)}$ and added to a 1D vector $b^{(j-1)}$. $l$ can be obtained after an activation function is applied.

$$l^j = \sigma \left( W^{(j-1)} \cdot f^{(j-1)} + b^{(j-1)} \right)$$ (9)

Before the output layer, the last activation function is set as softmax (10) to yield the classification output. In a classification problem, the number of the final layer would be set as the number of targeted categories and is set as 16 for the self-captured traffic. The softmax function takes the value of each neuron in the last layer as input and converts them between zero and one. Accordingly, the sum of all output values is one. At this point, each output value represents a corresponding category, and the greatest one is the predicted category.

$$softmax(z) = \frac{e^{z_i}}{\sum_i^n e^{z_i}}$$ (10)

When the construction of CAPC has been completed, we can commence the training process. The loss function of the classifier is set as cross entropy (11) to classify an input record. In the equation, $n$ stands for the number of training records, and $y_m$ represents the service type of that record. $f(x_i, \theta)$ is the whole deep learning model, $x_i$ is the $i^{th}$ record, and $\theta$ stands for the set of every parameter $W^j$ and $b^j$. With the criterion to judge the outcome of the model, the model can be updated by the gradient descent in (6).

$$cross\ entropy = - \sum_{i=1}^n y_m \cdot \ln f(x_i, \theta)$$ (11)
IV. EVALUATION AND EXPERIMENTAL RESULTS

In this section, the environment settings, namely, training and testing datasets, software and hardware specifications, and models for comparison, are first presented. Thereafter, the experimental results are demonstrated, including the accuracy, F1-score, and stability on the testing data of CAPC and other three models for comparison.

A. EXPERIMENT SETTINGS

1) DATASET FOR TRAINING AND TESTING

The two datasets of the evaluated network traffic are self-captured services and “ISCX VPN-nonVPN” encrypted traffic [12]. The service types of the self-captured private traffic are previously shown in Table 2, and the public VPN traffic is described in Table 5.

| TABLE 5. Description of the open VPN dataset. |
|------------------------------------------------|
| Service        | Type            | Full | Selected |
| AIM chat       | real-time text  | 2,097| 2,097    |
| email client   | text            | 21,297| 10,000  |
| Facebook audio | VoIP            | 1,092| 10,000  |
| Facebook chat  | real-time text  | 12,605| 10,000  |
| Facebook video | real-time video | 98,663| 10,000  |
| FTP            | file transfer   | 237,455| 10,000  |
| Gmail chat     | real-time text  | 8,417 | 8,417   |
| Hangouts audio | VoIP            | 175,356| 10,000  |
| Hangouts chat  | real-time text  | 24,415| 10,000  |
| Hangouts video | real-time video | 143,783| 10,000  |
| ICQ            | real-time text  | 5,074 | 5,074   |
| Netflix        | video           | 27,442| 10,000  |
| SCP            | file transfer   | 45,385| 10,000  |
| SFTP           | file transfer   | 64,507| 10,000  |
| Skype audio    | VoIP            | 142,400| 10,000  |
| Skype chat     | real-time text  | 7,644 | 7,644   |
| Skype file     | file transfer   | 154,849| 10,000  |
| Skype video    | real-time video | 140,195| 10,000  |
| Spotify        | music           | 94,894| 10,000  |
| BitTorrent     | P2P             | 67,428| 10,000  |
| Tor Twitter    | web content     | 8,065 | 8,065   |
| Vimeo          | video           | 95,272| 10,000  |
| VoIPbaster     | VoIP            | 147,267| 10,000  |
| YouTube        | video           | 217,345| 10,000  |
| TOTAL          |                 | 2,142,717| 222,797 |

The private dataset contains 16 services in four main service types, and 10,000 packets for each application are taken for evaluation, totally 16,000 packets. This dataset is a pure balanced dataset because the quantity of each service is the same. By contrast, the public dataset consists of 24 services, including several different VPN or nonVPN encrypted types. Our target is to identify a specific application no matter it is encrypted or not. The VPN and non-VPN traffic of a single application is mixed together while model training and testing. The quantity of services is imbalanced, and some of them are more than 100,000, and others are less than 10,000. At most 10,000 instances are arbitrarily sampled for each service to balance the data much as possible.

The datasets are split into three parts for the training and testing purposes: training, validation, and testing data, which are 70%, 10%, and 20% in proportion to the number of instances for each service, respectively.

2) SOFTWARE AND HARDWARE SPECIFICATIONS

The specifications are summarized in Table 6. The GPU is used to accelerate the training process, and the software platform is built on top of TensorFlow [55] GPU version and Keras library [56].

3) MODELS FOR COMPARISON

Three other deep learning models are built for comparison to highlight the performance of CAPC; these models include DNN, DAE, and 1D CNN, which have been evaluated in several dissertations [22], [47], [48] and gained excellent performance. However, the classification task of the public VPN dataset in the works [22], [47], [48] merely involve 17, 15, and 12 applications respectively. To obtain better performance in the more complex scenario (24 applications), the proposed CAPC is compared with the three models utilized in those researches. The DAE structure is the same as the DNN, while it merely has an extra procedure of autoencoder training. The CAPC is also the same as the 1D CNN, whereas the convolutional layers are built from the pretrained encoder. The description of each network structure for comparison is listed in Table 7 and the graphical presentation is shown in Fig. 12. The input layer contains 1500 neurons of each model due to the ethernet MTU size, and the output layer depends on the targeted number of categories (16 for the private dataset and 24 for the public dataset).

B. EXPERIMENTAL RESULTS

Fine-grained confusion matrices on the private and public testing data of the four models are shown in Fig. 14. A column
in a confusion matrix stands for the real (true) instances in that service. Hence, the diagonal of a matrix contains the correctly classified instances, and the values in other regions stand for the misclassified instances. Among the results on the private dataset, the DNN model in Fig. 14(a) cannot correctly classify the packets of FTP, SFTP, VMware, and Xen Server. Fig. 14(b) demonstrates that even the DAE model structure is identical to the DNN model, it still performs much better than DNN due to the parameter initialization of the autoencoder mechanism. The 1D CNN and CAPC model in Figs. 14(c) and 14(d), respectively, outperform the other two models, and almost achieve 100% accuracy. The results indicate that the autoencoding and convolution help in the private traffic classification task. The models on the public VPN dataset cannot perfectly classify all services, especially between Skype video and audio. There are some possible reasons for the non-ideal result of the Skype traffic classification: (1) The traffic of each service in the public dataset contains some irrelevant packets (ARP and DNS). (2) The Skype video traffic includes both the video and voice packets, which might confuse the model. Nevertheless, CAPC in Fig. 14(h) still has the best performance among all models. However, 1D CNN (Fig. 14(g)) in this time performs even worse than the DAE model (Fig. 14(f)). Therefore, the autoencoding mechanism is more influential on the public dataset than the convolution. Accordingly, DAE has higher accuracy than 1D CNN. Nevertheless, the convolution still helps CAPC in classifying better than DAE.

Several performance metrics, namely, accuracy, precision, and recall, are evaluated for a concise presentation. In a binary confusion matrix, TP, FP, FN, and TN are presented (Fig. 13). The metrics are described below:

- **Accuracy**: the ratio of all correctly predicted instances (TP and TN) of a class over all instances and is calculated as follows:
  \[ \text{Accuracy} = \frac{TP + TN}{TP + FP + FN + TN} \] (12)

- **Precision**: the ratio of correctly predicted instances (TP) of a class over all the instances predicted as that class (TP and FP) and is calculated as follows:
  \[ \text{Precision} = \frac{TP}{TP + FP} \] (13)

- **Recall**: the ratio of correctly predicted instances (TP) of a class over all the actual instances of that class (TP and FN) and is calculated as follows:
  \[ \text{Recall} = \frac{TP}{TP + FN} \] (14)

The metrics of each model on the private and public datasets are shown in Table 8. The table indicates that the proposed CAPC model outperforms on the private and public dataset, regardless of the accuracy, precision, or recall.

### Table 8. Metrics of each model on the datasets (the metrics are presented as the average values of the services).

| Model   | Structure                                      | Optimizer |
|---------|------------------------------------------------|-----------|
| DNN     | three fully connected layers, that have 256, 128, 64 neurons, followed by ReLU | adam      |
| DAE     | same as DNN, formed by a pretrained encoder    | adam      |
| CNN     | three 1D convolutional layers, that have 126, 64, 32 filters, followed by ReLU and max-pooling | adam      |

Among the metrics, precision emphasizes if the predictions are correct or not. Meanwhile, recall emphasizes if the actual class instances are correctly predicted or not. If we attach the same importance to precision and recall, then we can judge by \( F_1 \)-score as follows:

\[ F_1 - \text{score} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \] (15)

The \( F_1 \)-score of each service of four models on the datasets is listed in Table 9. The CNN-based structures efficiently perform on the private data, and DAE is slightly worse than the two. CAPC obtains the highest \( F_1 \)-score among most
FIGURE 14. Confusion matrices on the private and public datasets of the four models. (a) DNN on the private dataset. (b) DAE on the private dataset. (c) 1D CNN on the private dataset. (d) CAPC on the private dataset. (e) DNN on the public dataset. (f) DAE on the public dataset. (g) 1D CNN on the public dataset. (h) CAPC on the public dataset.
TABLE 9. F1-score of each service on the datasets.

| Private dataset | DNN | DAE | 1D CNN | CAPC |
|-----------------|-----|-----|--------|------|
| FTP             | 0.8329 | 0.9912 | 0.9987 | 1.0000 |
| SFTP            | 0.921 | 0.9988 | 0.9995 | 1.0000 |
| OneDrive        | 0.9737 | 0.9998 | 0.9993 | 1.0000 |
| SCP             | 0.9833 | 0.9891 | 0.9999 | 0.9992 |
| RTP video       | 0.9995 | 0.9998 | 0.9999 | 0.9999 |
| RTSP video      | 0.9952 | 0.997 | 0.999 | 0.9993 |
| UDP video       | 0.998 | 0.999 | 0.999 | 0.9997 |
| YouTube         | 0.9967 | 0.995 | 0.9995 | 0.9998 |
| LINE VoIP       | 0.9992 | 1.000 | 1.000 | 1.000 |
| Skype VoIP      | 0.995 | 0.9993 | 1.000 | 1.000 |
| Zoom VoIP       | 1.000 | 1.000 | 1.000 | 1.000 |
| join.me VoIP    | 1.000 | 1.000 | 1.000 | 1.000 |
| Windows RDP     | 0.9814 | 0.9993 | 0.9997 | 0.9997 |
| VMware          | 0.9976 | 0.9965 | 0.9993 | 0.9998 |
| XEN Server      | 0.9435 | 0.9965 | 0.9988 | 1.000 |
| NCU Cloud       | 0.8227 | 1.000 | 0.9995 | 1.000 |

| Public dataset  | DNN | DAE | 1D CNN | CAPC |
|-----------------|-----|-----|--------|------|
| AIM chat        | 0.9684 | 0.993 | 0.9847 | 0.9928 |
| email client    | 0.9718 | 0.9816 | 0.9647 | 0.9772 |
| Facebook audio  | 0.992 | 0.992 | 0.9917 | 0.994 |
| Facebook chat   | 0.935 | 0.9641 | 0.9647 | 0.9722 |
| Facebook video  | 0.976 | 0.9935 | 0.992 | 0.9945 |
| FTP             | 0.9391 | 0.9965 | 0.9985 | 0.9978 |
| Gmail chat      | 0.9304 | 0.9709 | 0.9407 | 0.9675 |
| Hangouts audio  | 0.9837 | 0.9875 | 0.978 | 0.9915 |
| Hangouts chat   | 0.9306 | 0.9899 | 0.9907 | 0.9912 |
| Hangouts video  | 0.982 | 0.9902 | 0.977 | 0.9925 |
| ICQ chat        | 0.9791 | 0.9646 | 0.9565 | 0.969 |
| Netflex         | 0.9835 | 0.995 | 0.995 | 0.995 |
| SCP             | 0.9995 | 0.9995 | 1.000 | 1.000 |
| SFTP            | 0.9943 | 0.9995 | 1.000 | 0.9995 |
| Skype audio     | 0.8357 | 0.8043 | 0.8339 | 0.8308 |
| Skype chat      | 0.9415 | 0.9665 | 0.9625 | 0.9741 |
| Skype file      | 0.9841 | 0.9799 | 0.9882 | 0.9875 |
| Skype video     | 0.7323 | 0.7754 | 0.7583 | 0.7754 |
| Spotify         | 0.9591 | 0.9819 | 0.9889 | 0.9892 |
| BitTorrent      | 0.9948 | 0.9962 | 0.996 | 0.9978 |
| Tor Twitter     | 0.9795 | 0.9924 | 0.9994 | 0.9989 |
| Vimeo           | 0.8673 | 0.9835 | 0.9158 | 0.9523 |
| VoipBuster      | 0.9985 | 0.9962 | 0.9954 | 0.997 |
| YouTube         | 0.8125 | 0.9785 | 0.9046 | 0.9935 |

Aside from the fine-grained classification, the models are tested with combined services of the private dataset in a coarse-grained manner. After the combination, 16 applications are turned into four main categories (Table 2).

The confusion matrices of the four models on the coarse-grained classification data are depicted in Fig. 15.

Despite the above-mentioned consequence, we test the stability of each model by continuously conducting the fine-grained classification five times on both two datasets. This step is conducted to prevent the models from occasionally going wrong due to the randomness of neural network initialization or other factors. The metric in the stability test is the overall accuracy in percentage (%) (Fig. 16).

The outcome in Fig. 16(a) shows that CAPC and 1D CNN have stable accuracy (maintaining at over 99.90% rates) on the private dataset of 16 services. By contrast, DNN moderately performs (oscillating between 94% and 97%). The classification on the public dataset is difficult because more services (i.e., 24 applications) are involved. Certain service types, such as Facebook audio call and Skype VoIP, are rather similar. The difficulty can be reflected

![Confusion matrices of four models on the private data coarse-grained classification. (a) DNN; (b) DAE; (c) 1D CNN; (d) CAPC.](image)

![Fine-grained classification stability performance of the four models. (a) private and (b) public datasets.](image)
in Fig. 16(b), in which CAPC can merely reach 97.42% at most, and the others perform even worse.

A simple SDN environment built with Mininet [57] is used to ensure the feasibility of deploying packet-based classification models in a network environment (as Fig. 17). The environment is built in an Ubuntu virtual machine on a VMware server, and the specifications of the system are listed in Table 10. The classification models are deployed within the controller, attempting to classify the incoming packets encapsulated in packet-in messages in real time. The self-captured PCAP files are stored on the host 2, such as FTP, Skype VoIP, and Windows RDP. The traffic replaying toolkit, BitTwist [58], is used to replay the stored traffic files from host 2. The first incoming packet of a service would be forwarded to the controller according to the preinstalled rule on the switch (sw2). The packet transmitted to the controller would be further classified by the models, and the time used for classification of each model is recorded for comparison.

The consuming time of classifying the very first incoming packet of a service of each deep learning model is measured and depicted in Fig. 18. The consuming time is calculated from receiving the packet to the end of classification. As packet-based classification models, each model can accomplish the classification task within 0.1s, which are much faster than flow-based methods that almost take more than 1s to form a flow. As shown in the figure, the two CNN-based models (1D CNN and CAPC) often take 0.02s to classify a packet of a service, which may result from the more complex computation of CNN; the two non-CNN-based models (DNN and DAE) have lower classification time at around 0.015s, which are a bit faster than the two with CNN. The classification time is acceptable, because once the classification results are generated, the controller then can add a rule to the corresponding service, and the packets of that service will not be forwarded to the controller later.

Overall, the CAPC model gains the highest performance when it comes to the accuracy, F1-score, or stability, but requires a bit more time to complete a classification task than the models without CNN calculation. However, all of models are still more efficient than flow-based methods, and more suitable to be deployed in a real-time-demanding system with the lower classification time.

V. CONCLUSION

In this research, CAPC has been developed and evaluated with outstanding performance compared with other deep learning models on the basis of the idea of combining 1D convolution and autoencoding. On the private dataset with 16 services, CAPC achieves average precision and recall of 0.9998 and 0.9998, respectively, and high accuracy of 99.98% in the fine-grained classification. CAPC also obtains a good result of 99.99% accuracy in the coarse-grained classification. On the public dataset with 24 services, CAPC achieves average precision and recall of 0.9736 and 0.9733, respectively, and high accuracy of 97.42%. The numbers of services with highest F1-score values on the two datasets are more than the other models (i.e., 15 and 18). During the five consecutive stability tests, CAPC constantly obtains a satisfactory performance. Last but not least, the classification time of CAPC is also evaluated and obtains an acceptable result (around 0.02s for each classification), which can ensure the feasibility to deploy the model in a network.

| Hardware/Software | Specification |
|-------------------|--------------|
| CPU               | AMD Opteron™ Processor 4386 |
| Hypervisor        | VMware ESXi 6.7.0 1306603 |
| Operating System  | Ubuntu 16.04 LTS |
| Network Simulator | Mininet 2.3.0d6 |
| OpenFlow Switch Simulator | Open vSwitch 2.11.0 |
| SDN Controller    | Ryu 4.32 |
| Traffic Generator | Bit-Twist 2.0 |
| Machine Learning Engine | TensorFlow 1.14.0 |
| Machine Learning Toolkits | Keras 2.1.6, scikit-learn 0.22.1 |
| Data Preprocessing | pandas 1.0.0, numpy 1.16.0 |

FIGURE 17. Simulated network environment for real-time traffic classification.

FIGURE 18. Classification time of every model on each service.
environment. The studies related to traffic classification have been comprehensively surveyed, and different types of classification schemes, such as fine/coarse-grained and flow/packet-based, have been elaborated. In future work, the proposed model can be deployed in a practical SDN environment on the basis of the packet-based classification scheme to instantly classify the incoming packets. The corresponding service policies can be subsequently enforced.

REFERENCES

[1] Y. Chen, T. Farley, and N. Ye, “QoS requirements of network applications on the Internet,” Inf. Knowl. Syst. Manage., vol. 4, no. 1, pp. 55–76, Mar. 2004, doi: 10.1109/IKSM.2010.1514575.
[2] F.-H. Tseng, X. Chen, L.-D. Chou, H.-C. Chao, and S. Chen, “Support vector machine approach for virtual machine migration in cloud data center,” Multimedia Tools Appl., vol. 74, no. 10, pp. 3419–3440, May 2015.
[3] E. Alpaydin, Introduction to Machine Learning. Cambridge, MA, USA: MIT Press, 2020.
[4] J. Xie, F. R. Yu, T. Huang, R. Xie, J. Liu, C. Wang, and Y. Liu, “A survey of machine learning techniques applied to software defined networking (SDN): Research issues and challenges,” IEEE Commun. Surveys Tuts., vol. 21, no. 1, pp. 393–430, 1st Quart., 2019, doi: 10.1109/COMST.2018.2866942.
[5] Y. Zhao, Y. Li, X. Zhang, G. Geng, W. Zhang, and Y. Sun, “A survey of networking applications applying the software defined networking concept based on machine learning,” IEEE Access, vol. 7, pp. 95397–95417, 2019, doi: 10.1109/ACCESS.2019.2928564.
[6] M. T. Hagan, H. B. Demuth, M. H. Beale, and J. De Orlando, Neural Network Design. 2016.
[7] K. Zhang, W. Zuo, Y. Chen, D. Meng, and L. Zhang, “Beyond a Gaussian denoiser: Residual learning of deep CNN for image denoising,” IEEE Trans. Image Process., vol. 26, no. 7, pp. 3142–3155, Jul. 2017, doi: 10.1109/TIP.2017.2662206.
[8] F. Radenovic, G. Tolias, and O. Chum, “Fine-tuning CNN image retrieval with no human annotation,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 41, no. 7, pp. 1655–1668, Jul. 2019, doi: 10.1109/TPAMI.2018.2846566.
[9] H. Xie, M. A. B. Ahmadon, and S. Yamaguchi, “Evaluation of rough sets data preprocessing on context-driven semantic analysis with RRN,” in Proc. IEEE 7th Global Conf. Consum. Electron. (GCCE), Oct. 2018, pp. 410–413.
[10] J.-D. Zhang and C.-Y. Chow, “SEMA: Deeply learning semantic meanings and temporal dynamics for recommendations,” IEEE Access, vol. 6, pp. 54106–54116, 2018, doi: 10.1109/ACCESS.2018.2871970.
[11] K.-C. Chiu (2020). Network Service Traffic. IEEE Dataport, Accessed: Aug. 3, 2020. [Online]. Available: http://dx.doi.org/10.21227/c8w4-nj38.
[12] A. H. Lashkari, G. Draper-Gil, M. S. I. Mannun, and A. A. Ghorbani, “Characterization of encrypted and vpn traffic using time-related,” in Proc. Int. Conf. Inf. Syst. Secur. Privac. (ICISSP), Feb. 2016, pp. 407–414, doi: 10.1109/TIP.2017.2846566.
[13] P. Wang, X. Chen, F. Ye, and Z. Sun, “A survey of techniques for mobile network service traffic classification with a convolutional long short-term memory neural network,” in Proc. IEEE 16th Int. Conf. Smart City, IEEE 4th Int. Conf. Data Syst. (HPC/SmartCity/DSS), Zhangjiajie, China, Aug. 2019, pp. 429–436.
[14] K. Zhang, S. Zhao, J. Zhang, X. Ma, and F. Huang, “STNN: A novel TLS/SSL encrypted traffic classification using capsule neural network,” in Proc. IEEE 25th Int. Conf. Parallel Distrib. Syst. (ICPADS), Tianjin, China, Dec. 2019, pp. 907–910.
[15] H.-K. Lim, J.-B. Kim, J.-S. Heo, K. Kim, Y.-G. Hong, and Y.-H. Han, “Packet-based network traffic classification using deep learning,” in Proc. Int. Conf. Artif. Intell. Inf. Commun. (ICAIHC), Okinawa, Japan, Feb. 2019, pp. 046–051.
[16] Y. Ma, X.-Y. Fan, and L.-X. Chen, “A CNN-based packet classification of eMMB, mMTC and URLLC applications for 5G,” in Proc. Int. Conf. Intell. Comput. its Emerg. Appl. (ICEA), Tainan, Taiwan, Aug. 2019, pp. 140–145.
[17] Z. Zou, J. Ge, H. Zheng, Y. Wu, C. Han, and Z. Yao, “Encrypted traffic classification with a convolutional long short-term memory neural network,” in Proc. IEEE 20th Int. Conf. High Perform. Comput. Commun.; IEEE 14th Int. Conf. Smart City; IEEE 4th Int. Conf. Data Syst. (HPC/SmartCity/DSS), Exeter, U.K., Jun. 2018, pp. 329–334.
[18] I. Goodfellow, Y. Bengio, and A. Courville, Deep Learning. Cambridge, MA, USA: MIT Press, 2016, [Online]. Available: http://www.deeplearningbook.org.
[19] P. Amaral, J. Dinis, P. Pinto, L. Bernaldo, J. Tavares, and H. S. Mamede, “Machine learning in software defined networks: Data collection and traffic classification,” in Proc. IEEE 24th Int. Conf. Netw. Protocols (ICNP), Singapore, Nov. 2016, pp. 1–5.
[20] J. Yan and J. Yuan, “A survey of traffic classification in software defined networks,” in Proc. 1st IEEE Int. Conf. Hot Information-Centric Netw. (HotICN), Aug. 2018, pp. 200–206.
[21] J. Touch, E. Lear, A. Mankin, M. Kojo, K. Ono, and M. Stiemerling, “Service name and transport protocol port number registry,” Internet Assigned Numbers Authority (IANA), Marina del Rey, CA, USA. [Online]. Available: https://www.iana.org/assignments/service-names-port-numbers
[22] IANA, Internet Assigned Numbers Authority. Internet Assigned Numbers Authority. [Online]. Available: https://www.iana.org/
[23] A. Moore and K. Papagiannaki, “Toward the accurate identification of network applications,” in Proc. Int. Workshop Passive Active Netw. Meas., Berlin, Germany, 2005, pp. 41–54.
[24] A. Madhukar and C. Williamson, “A longitudinal study of P2P traffic classification,” in Proc. Int. Workshop Netw. Modeling, Anal., Simulation, Monterey, CA, USA, Sep. 2006, pp. 179–188.
[25] M. Finsterbusch, C. Richter, E. Rocha, J.-A. Muller, and K. Hansgen, “A survey of payload-based traffic classification approaches,” IEEE Commun. Surveys Tuts., vol. 16, no. 2, pp. 1135–1156, 2nd Quart., 2014, doi: 10.1109/SURV.2013.100613.00161.
[26] F. Gringoli, “GT: Picking up the truth from the ground for Internet traffic,” in Proc. 13th Int. Conf. Comput. Intell. Secur. Privacy (ICISSP), Fuzhou, China, Nov. 2018, pp. 1–5.
[40] F. F. Taylor, R. Spolaoar, M. Conti, and I. Martinovic, “AppScanner: Automatic fingerprinting of smartphone apps from encrypted network traffic,” in Proc. IEEE Eur. Symp. Secur. Privacy (EuroS&P), Mar. 2016, pp. 439–454.

[41] M. Conti, L. V. Mancini, R. Spolaoar, and N. V. Verde, “Analyzing Android encrypted network traffic to identify user actions,” IEEE Trans. Inf. Forensics Security, vol. 11, no. 1, pp. 114–125, Jan. 2016, doi: 10.1109/TIFS.2015.2478741.

[42] Y.-N. Dong, J.-J. Zhao, and J. Jin, “Novel feature selection and classification of Internet video traffic based on a hierarchical scheme,” Comput. Netw., vol. 119, pp. 102–111, Jun. 2017.

[43] O. Salman, I. H. Elhajj, A. Chehab, and A. Kayssi, “A multi-level Internet traffic classifier using deep learning,” in Proc. 9th Int. Conf. Netw. Future (NOF), Nov. 2018, pp. 68–75.

[44] M. Lopez-Martin, B. Carro, A. Sanchez-Esguevillas, and J. Lloret, “Network traffic classifier with convolutional and recurrent neural networks for Internet of Things,” IEEE Access, vol. 5, pp. 18042–18050, 2017, doi: 10.1109/ACCESS.2017.2747560.

[45] R. Vinayakumar, K. P. Soman, and P. Poonachandran, “Applying convolutional neural network for network intrusion detection,” in Proc. Int. Conf. Adv. Comput., Commun. Informat. (ICACCI), Sep. 2017, pp. 1222–1228.

[46] W. Wang, Y. Sheng, J. Wang, X. Zeng, X. Ye, Y. Huang, and M. Zhu, “HAST-IDS: Learning hierarchical spatial-temporal features using deep neural networks to improve intrusion detection,” IEEE Access, vol. 6, pp. 1792–1806, 2018, doi: 10.1109/ACCESS.2017.2782850.

[47] P. Wang, F. Ye, X. Chen, and Y. Qian, “Datnet: Deep learning based encrypted network traffic classification in SDN home gateway,” IEEE Access, vol. 6, pp. 55380–55391, 2018, doi: 10.1109/ACCESS.2018.2872430.

[48] W. Wang, M. Zhu, J. Wang, X. Zeng, and Z. Yang, “End-to-end encrypted traffic classification with one-dimensional convolution neural networks,” in Proc. IEEE Int. Conf. Intel. Secur. Informat. (IISI), Beijing, China, Jul. 2017, pp. 43–48.

[49] L. Xu, X. Zhou, Y. Ren, and Y. Qin, “A traffic classification method based on packet transport layer payload by ensemble learning,” in Proc. IEEE Symp. Comput. Commun. (ISCC), Barcelona, Spain, Jun. 2019, pp. 1–6.

[50] G. Aceto, D. Ciuonzo, A. Montieri, and A. Pescapà, “Mobile encrypted traffic classification using deep learning: Experimental evaluation, lessons learned, and challenges,” IEEE Trans. Netw. Service Manage., vol. 16, no. 2, pp. 445–458, Jun. 2019, doi: 10.1109/TNSM.2019.2899085.

[51] L. Vu, H. V. Thuy, Q. U. Nguyen, T. N. Ngoc, D. N. Nguyen, D. T. Hoang, and E. Dutkiewicz, “Time series analysis for encrypted traffic classification: A deep learning approach,” in Proc. 18th Int. Symp. Commun. Inf. Technol. (ISCIT), Bangkok, Thailand, Sep. 2018, pp. 121–126.

[52] Wikipedia. (2020). pcap. Accessed: Apr. 3, 2020. [Online]. Available: https://en.wikipedia.org/wiki/Pcap

[53] Wireshark. (2020). Wireshark. Go Deep. Accessed: Apr. 23, 2020. [Online]. Available: https://www.wireshark.org/

[54] Y. Yang, C. Kang, G. Gou, Z. Li, and G. Xiong, “TLS/SSL encrypted traffic classification with autoencoder and convolutional neural network,” in Proc. IEEE 20th Int. Conf. High Perform. Comput. Commun., IEEE 16th Int. Conf. Smart City, IEEE 4th Int. Conf. Data Sci. Syst. (HPCC/SmartCity/DSS), Exeter, U.K., Jun. 2018, pp. 362–369.

[55] M. Abadi, A. Agarwal, P. Barham, and E. Brevdo, “TensorFlow: Large-scale machine learning on heterogeneous distributed systems,” 2016, arXiv:1603.04467. [Online]. Available: http://arxiv.org/abs/1603.04467

[56] F. Chollet (2015). Keras. [Online]. Available: https://github.com/keras-team/keras

[57] Mininet Team. (2018). Mininet: An Instant Virtual Network on Your Laptop (or Other PC). Accessed: Jul. 6, 2020. [Online]. Available: http://mininet.org/

[58] A. Heng. (2006). Bit-Twist: Libpcap-Based Ethernet Packet Generator. Accessed: Jun. 26, 2020. [Online]. Available: http://bit-twist.sourceforge.net/