FineNet: Few-shot Mobile Encrypted Traffic Classification via a Deep Triplet Learning Network based on Transformer

Shengbao Li[1,2,3], Qian Qiang[4], Tianning Zang[1,2,*], Lanqi Yang[1,2], Tianye Gao[1,2]
[1]Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100045, China
[2]School of Cyber Security, University of Chinese Academy of Sciences, Beijing 101408, China
[3] National Computer Network Emergency Response Technical Team/Coordination Center of China (Shandong Branch), Jinan 250004, China
[4]National Computer Network Emergency Response Technical Team/Coordination Center of China, Beijing 100000, China

2024 IEEE Wireless Communications and Networking Conference (WCNC) | 979-8-3503-0358-2/24/$31.00 ©2024 IEEE | DOI: 10.1109/WCNC57260.2024.10570907

***Abstract*—The existing encrypted mobile traffic classification methods often require a large number of training sets to ensure the effect of the model. When it is difficult to collect enough samples, these methods may not yield satisfactory results. Therefore, it is crucial to study an effective few-shot learning method to achieve accurate traffic classification with very few samples. In this paper, we propose FineNet, a few-shot fine-grained classification technology based on a triplet deep learning network with Transformer. The triplet deep learning network is able to discover subtle differences between traffic flows and effectively transfer existing classification knowledge to few-shot scenarios. Moreover, we introduce Transformer as the base model for the triplet network, fully leveraging Transformer's powerful representation capabilities for sequential data to enhance the express ability of FineNet. We conduct multiple comparative experiments, and the result proves that the accuracy rate is at least 2.4% higher than state-of-the-art approaches in few-shot environment.**

Keywords—App encrypted traffic, few-shot learning, Triplet Network, Transformer

I. Introduction

The development of mobile phone applications has been growing rapidly, and mobile Internet and its applications have been a gathering space for massive amounts of personal sensitive data. In order to protect user privacy, mobile Internet traffic encryption has emerged as a prevailing trend, and most mobile Internet applications adopt data encryption transmission technologies such as https[1]. However, while mobile Internet encryption technology protects user privacy, it brings great challenges to management of network security, data security and content security in mobile cyberspace. Research on encrypted traffic classification for mobile Internet applications has drawn widespread attention in the cybersecurity academic community.

However, most studies usually focus on solving the problem of coarse-grained traffic classification with sufficient training samples, and cannot effectively deal with challenging scenarios where App training samples are insufficient. Therefore, these methods may not fully meet the fine-grained needs of mobile cybersecurity management. In many cases, it is difficult and costly to acquire large-scale labeled datasets of encrypted App traffic. Therefore, research on few-shot learning methods for App encrypted traffic classification is highly necessary. Few-shot encrypted traffic classification enables the classification of new traffic types with a limited number of samples. Few-shot learning methods typically exhibit better feature representation capabilities, allowing them to discover subtle differences and similarities even with limited App traffic samples. Few-shot learning methods also possess stronger generalization abilities, enabling the transfer of existing general classification knowledge to classify traffic of new App classes.

To address the challenges of few-shot App traffic classification, we introduce FineNet, a fine-grained encrypted traffic classification method designed specifically for App traffic in a few-shot environment. We deeply study the characteristics of traffic classification in few-shot scenarios, and develop a few-shot learning framework based on meta-learning. FineNet is good at perceiving the subtle difference between samples, and realize high-precision classification of App traffic in challenging environment lacking enough training samples.

Our main contributions are as follows:

1) We propose a new few-shot learning traffic classification framework based on triplet deep network, and realize the fine-grained classification of App encrypted traffic with very few training samples.

2) To enhance the representation ability of App traffic sequence features, we adopt the Transformer as the base model of triplet network. Transformer's powerful sequence modeling capabilities, particularly its self-attention mechanism, enable FineNet to effectively capture deep relationship among flow bytes. Therefore, FineNet is able to discover subtle differences between traffic flows with limited samples.

3) Based on our self-built App traffic dataset, we conducted evaluation experiments. Experiments show that FineNet achieves better performance than existing methods for few-shot App traffic classification.

The rest of this paper is organized as follows. In Section II, we introduce the current progress of related work. In Section III, we introduce basic knowledge about our research. In Section IV, we introduce the main architecture of FineNet. In Section V, we conduct comparation experiments and make performance evaluation. Finally, we make a conclusion about our study.

II. Related Work

A. Classification for App Encrypted Traffic

In recent years, research on App encrypted traffic classification has made significant progress. In [2][3], Taylor et al. proposed AppScanner, which is an effective encrypted App traffic classification method, leveraging machine learning methods to extract statistical features for classification. In [4], Thijs van Ede et al. propose a semi-

*Corresponding author. E-mail address: zangtianning@iie.ac.cn

979-8-3503-0358-2/24/$31.00 ©2024 IEEE

supervised app fingerprinting method for encrypted network traffic, which can automatically find temporal correlations of traffic flows and generate App fingerprints. In [5], based on message types and length block sequences, a Markov model is trained and concatenated with all application probabilities as a fingerprint for classification. In [6], C. Liu et al. propose an end-to-end classification model that learns representative features from raw streams based on deep learning. In [7], a deep learning approach is proposed for mobile app recognition task, which can automate the feature engineering process in an end-to-end manner. In order to classify App encrypted traffic at finer granularity, some scholars have carried out research on traffic classification of in-App activities. In [8] [9] [10] [11], several methods have been proposed to specifically identify user behavior in App encrypted traffic using different approaches.

B. *Classification for Encrypted Traffic based on Few-shot Learning*

Few-shot learning is a challenging scenario in encrypted traffic classification. Some studies have begun to focus on the few-shot learning of encrypted traffic. In [12], Payap Sirinam et al. proposed a fingerprint attack method based on triplet network, which only needs quite few samples to realize the accurate classification of Tor network encrypted traffic. In [13], Mingshu He et al. proposed a few-shot learning method based on CNN and AutoEncoder, which can realize abnormal traffic detection with few training samples. In [14], Tianyu Cui et al. proposed a few-shot learning method based on twin heterogeneous graph attention networks for the IPV6 address association attack problem. In [15], Xinjie Lin et al. achieved fine-tuning training under the condition of a small number of labeled samples by pre-training large-scale unlabeled data.

III. PRELIMINARIES

In this section, we elaborate on some necessary preparations and basic knowledge about our research, which will help to better understand this work.

A. *Problem definition*

The purpose of the paper is to classify App encrypted traffic with very few samples. In order to fully simulate the few-shot learning scenario, we divide the dataset into base dataset and few-shot dataset. The base dataset have a large-scale flows representing easy-to-obtain App traffic samples in real life, which is defined as $S_{base}=\{(x_i,y_i)\}$, where label $y_i \in \{C_1,...,C_i...C_n\}$. The few-shot dataset have limited flows representing App traffic samples that are difficult to obtain in real life, which is defined as $S_{few}=\{(x_i,y_i)\}$, where label $y_i \in \{C_{n+1},...,C_i...C_{n+m}\}$. The App classes included in the two datasets are completely different, which means $\{C_1,...,C_i...C_n\} \cap \{C_{n+1},...,C_i...C_{n+m}\}=\varnothing$. Our goal is to transfer the existing classification knowledge from App traffic in the large-scale dataset S_{base} to the new and few-shot App traffic in the few-shot dataset S_{few}, by learning a generalizable and transferrable few-shot model with S_{base}. Thus, we are aimed to achieve accurate classification of new different App traffic with the insufficient training samples from S_{few}, which is the main problem we are going to study.

B. *Triplet Network*

Few-shot learning is an important research direction in the field of machine learning, and it focuses on how to effectively train models with only a small number of labeled samples to achieve better generalization performance [16]. Meta-learning is one of the most commonly used and effective few-shot learning method. As a typical kind of few-shot learning method, meta-learning [17] is aimed to "learn how to learn", that is, using previous knowledge and experience to guide the learning of new tasks, and acquire the ability to learn.

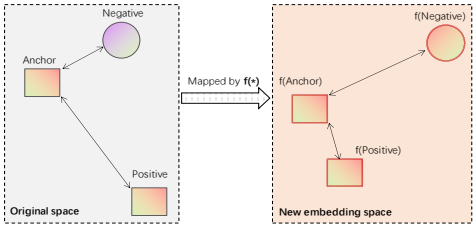

Fig. 1. Training the model as embedding feature generator f(*): in the new representation space, minimize the embedding distance of examples from the same App and maximize the embedding distance for examples from different Apps

Under the meta-learning paradigm, the triplet network [18] is proposed. It is a specialized neural network architecture, which consists of three identical base neural networks that share same parameters. The main principle of the triplet network is to learn the relative distances of samples in the representation space. By utilizing paired or grouped data, as is shown in Fig.1, the network acts as a feature embedding generator aiming at acquiring a new feature representation space, where similar samples are clustered close together and dissimilar samples are positioned relatively far apart. A triplet network is to use triplets as input of three base network consisting of identical structures and shared parameters. Each triplet contains three samples: anchor sample (A), positive sample (P) and negative sample (N).

To achieve the objective of learning better feature representations for classification tasks, a new loss function called Triplet Loss [18] is designed. By optimizing the Triplet Loss during the training process, the model learns to map similar samples close together and dissimilar samples far apart in the feature space. In this way, the model can effectively create more discriminative feature representations, enabling better performance in classification tasks with limited training samples. The basic form of the Triplet Loss function is:

$$L(A,P,N) = max(\|f(A) - f(P)\|^2 - \|f(A) - f(N)\|^2 + margin, 0) \quad (1)$$

where, f represents the base model as embedding generator function mapping the input to a new embedding space, such as a neural network. $\|\cdot\|^2$ denotes the Euclidean distance. *Margin* is a hyperparameter, indicating the minimum gap between positive and negative samples, which is used to prevent model overfitting and make the learning more robust.

IV. METHODOLOGY

This section primarily introduces FineNet, a framework for classifying App encrypted traffic under few-shot condition, which is based on a deep meta-learning network. FineNet adopts a triplet network based on Transformer for few-shot meta-learning. As is shown in Fig.2, the triplet network is composed of three Transformer as base model in parallel with completely identical shared parameters. FineNet mainly consists of modules for traffic preprocessing, triplet task extraction, embedding generator pretraining, and few-shot finetune classification. We will introduce each module in detail below.

A. *Traffic Pre-processing Module*

The traffic preprocessing module is used to denoise, group

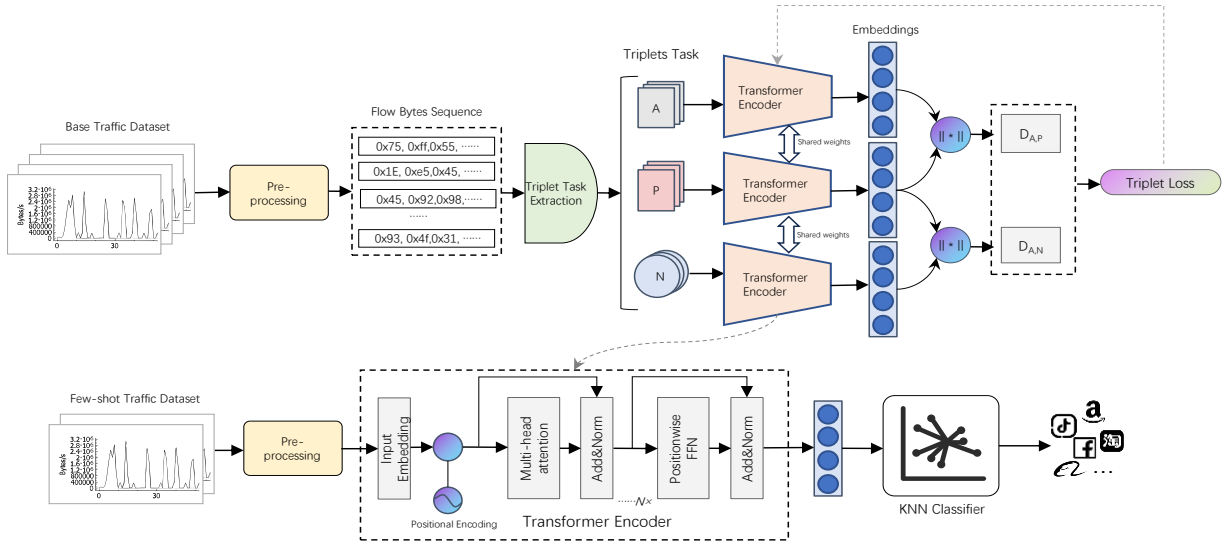

Fig. 2. General illustration of FineNet framework

and extract feature vectors from the original traffic, which match the input format of the classifier.

1)Traffic flow grouping. A traffic flow is usually the traffic generated by App's certain operations to access a specific service. As depicted in *Section III(A)*, it can be expressed as $FLOW=\{P_1, P_2, P_3, \cdots P_i \cdots\}$, where P_i is a packet in the flow. We use traffic flow as the basic unit to realize App feature labeling and traffic classification.

2)Traffic flow segmentation. A group of packets in a traffic flow are further segmented to form more fundamental flow units. The segmentation method is as follows: given a traffic flow denoted as *FLOW*, we set a time threshold symbolized as t_τ. Subsequently, we identify the gap where the time interval between two consecutive packets surpasses t_τ. This gap, characterized by its lack of traffic, is then utilized to segment the packet group within the traffic flow, forming more fundamental units, termed as $FLOW_s$.

3)Traffic Feature Extraction. We use the flow's raw payload byte sequence as the feature vector. Compared to commonly used statistical features or packet length sequence features, the packet raw payload byte sequence provides much more detailed information about the traffic flow. Additionally, the packet raw payload byte sequence is more suitable as the input for the Transformer model used in our method. We intercept the first 1000 bytes of each flow as a fixed-length value sequence, and fill up with zero vectors if there are not enough bytes. Thus, each flow sample is represented as a two-dimensional vector, where the first dimension represents 1000-bytes sequence and the second dimension represents the raw bit content of each byte.

B. Training Task Extraction Module

The task extraction module is to extract task samples suitable for the triplet network from the original training dataset. Each sample is a triplet *(A, P, N)* consisting of three original samples: the anchor sample*(A)*, the positive sample*(P)*, the negative sample*(N)*. The permutations and combinations of triplets can greatly impact the speed of training convergence and performance of the model. We need to mine effective triplets from the original dataset.

An easy triplet is one where the distance between *A* and *P* is less than the distance between *A* and *N*. In other words, the model already has a good understanding of these instances and can correctly classify them and it is of little significance to train the triplet network. A hard triplet is one where the distance between *A* and *N* is less than the distance between *A* and *P*. In this case, the model is currently misclassifying the samples, and so learning from these triplets can help improve its performance. Therefore, we hope to mining hard triplets from original datasets. Our mining strategy is as follows: 1) Firstly, we build anchor-positive pairs from raw samples belonging to the same class. 2) Then, for each anchor-positive pair, we randomly find the corresponding negative sample to form a hard triplet, which satisfies that the distance between *A* and *N* is less than the distance between *A* and *P*.

This strategy strikes a balance between convergence speed and effectiveness. The algorithm is presented below:

Algorithm 1 Triplets Mining Algorithm

Input: Raw Training Samples
Output: Triplets Task Samples

$S_{Anchor}, S_{Positive}, S_{negative} \leftarrow null$
Step1: Build All Positive Pairs
for $m \leftarrow 1$ **to** *classNum* **do**
  $Pairs \leftarrow$ getAllPostivePairsForClass(m)
  $S_{Anchor} \leftarrow S_{Anchor} \cup Paris[0]$
  $S_{Positive} \leftarrow S_{Positive} \cup Paris[1]$
end for
Step2: Build Hard Negative Pairs
for $m \leftarrow 1$ **to** $S_{Anchor}.length$ **do**
  $sim \leftarrow$ caculateSimilarity($S_{Anchor}[m]$, $S_{Positive}[m]$)
  $allNegatives \leftarrow$ getNotSameClass($S_{Anchor}[m]$)
  $possibleNegatives \leftarrow$ null
  for $n \leftarrow 1$ **to** $allNegatives.length$
    If caculateSimilarity($allNegatives(n)$, $S_{Anchor}[m]$)$+alpha>sim$
      $possibleNegative \leftarrow possibleNegatives \cup$ Possibles(n)
  end for
  $S_{negative} \leftarrow S_{negative} \cup$random$(possibleNegatives)$
end for
return $(S_{Anchor}, S_{Positive}, S_{negative})$

C. Pre-training module

The primary function of the pre-training module is to train the triplet network with triplet loss function on a large-scale base dataset. The pre-training model, which acts as a feature embedding generator, receives the payload feature vector of a traffic sample and subsequently transforms it into an embedding vector within a new feature embedding space. In

TABLE I. STATISTICS OF THE DATASETS

No.	Apps	Flows	Packets	Type
1	Ivwen	2119	440197	Base Class
2	Zhihu	3723	1431512	Base Class
3	Sohu News	8057	1579329	Base Class
4	Kmxs Reader	2607	619277	Base Class
5	Taobao Trip	2340	652515	Base Class
6	Meituan	2407	118080	Base Class
7	58	6056	624936	Base Class
8	BaiduMap	8951	449163	Base Class
9	Autohome	3928	1475960	Base Class
10	Vipshop	2918	289565	Base Class
11	Baidu Search	10263	1448974	Few-shot Class
12	Suning Ebuy	3911	751891	Few-shot Class
13	CloudMusic	4721	1369701	Few-shot Class
14	EastMoney	4414	305163	Few-shot Class

the new embedding space, it reduces the distance between embedding vectors of samples belonging to the same category, while increasing the distance between vectors from different categories.

In order to acquire better feature representation for raw payload byte sequence, we choose Transformer [19] as the base model of triplet network. Transformer is a deep learning model specifically designed for processing sequential data. Unlike traditional recurrent neural networks (RNNs) that process sequential data sequentially, Transformer can parallelize computations across the entire sequence, making it more efficient and scalable. With strong learning ability for traffic sequence, Transformer model is able to mining the detailed features of traffic flows and yield representative embedding vectors. Its key structural characteristics are mainly described as follows:

1) Multi-head Self-attention Mechanism. The multi-head self-attention model is a neural network structure for sequence modeling, which can capture the relationship between different positions in the sequence and model the relationship between different features at the same time. In the multi-head self-attention model, the input sequence is mapped into the *Q*, *K* and *V* matrices respectively, and then the representation of each position is obtained by calculating the attention. Finally, these representations are weighted and summed to obtain the final output. Different heads can capture different relations, thus improving the performance of the model. Specifically, the calculation of multi-head self-attention model is as follows:

$$Attention(Q,K,V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V \quad (2)$$

$$head_i = Attention\left(QW_i^Q, KW_i^K, VW_i^V\right) \quad (3)$$

$$MultiHead(Q,K,V) = Concat\left(head_1,\dots,head_h\right)W_o \quad (4)$$

Where W_i^Q, W_i^K, W_i^V represent three linear transformation matrices.

2) Position Encoding. In the self-attention model, since the representation of each position only depends on its relationship with other positions, it has nothing to do with its position in the sequence. Position encoding [19] is a method to express the position information of each position in the sequence by adding a fixed vector to the input sequence. The formula is as follows:

$$\begin{cases} PE_{2i}(p) = \sin\left(p/10000^{2i/d_{pos}}\right) \\ PE_{2i+1}(p) = \cos\left(p/10000^{2i/d_{pos}}\right) \end{cases} \quad (5)$$

Where *p* is the position in the sequence, and d_{pos} denotes the dimensionality of the Positional Encoding (PE). *2i* and *2i+1* refer to even and odd dimensions, respectively. By adding position encoding, the self-attention model can better capture the position information in the sequence and improve the performance of the model.

D. Few-shot Fine-tune module

The fine-tune module is mainly used for few-shot classification of App encrypted traffic with limited training samples. It consists of a Transformer which has been pre-trained previously and a K-Nearest Neighbor algorithm (KNN) classifier. The two models are connected in series.

Transformer model is used to generate embedding feature vectors and enhance feature representation in a new space, and KNN is used for final classification tasks. The KNN classifier is a simple yet effective classification algorithm. It determines the category of new samples just based on the distance between samples and a voting mechanism. Since the triplet Transformer has learned the similarity between samples, KNN classifier can directly use these learned features for classification with very few samples. Therefore, KNN is very suitable for few-shot classification problems based on metric learning.

V. Experiment and Evaluation

A. App Traffic Datasets

We synthesize and establish traffic sample datasets from mainstream Apps. Referring to our previous study [22], We simulate App operations and collect traffic through MonkeyRunner, an automated testing framework that comes with the Android SDK. We compile MonkeyRunner test scripts to automatically install and run APK files, and simulate operations of Apps randomly. Then we tested 14 mainstream Apps from App online market in China. The traffic of Apps can be collected by a tool named TcpDump, and each test result is saved as a pcap file separately. A total of 10GB of network traffic packets have been collected. We construct a mobile traffic dataset containing 14 Apps. According to *Section III(A)*, in order to fully simulate the few-shot learning scenario, we divide the dataset into base dataset and few-shot dataset. As Table I shows, we choose 10 Apps as base classes to create base dataset, which are used to learn the App traffic feature representation in pretraining process. The rest 4 Apps are randomly chosen as few-shot classes to create few-shot dataset, and we sample 100 samples from each few-shot class. The ultimate goal of our experiment is to achieve accurate classification of App traffic for few-shot classes.

B. Evaluation metrics

To evaluate the performance of our model, we employ commonly used metrics such as *Precision*, *Recall rate*, *F1-score*, etc. *Precision* reflects the proportion of the data actually labeled true in the data predicted to be true. *Recall* reflects the proportion of the data predicted to be true in the data actually labeled true. *F1-score* is the balance between comprehensive precision and recall. For multi-classification, we use the *Macro* method to calculate the above metric values. That is, sum and average the values of each category.

C. Model implementation

In order to improve the classification effect of the model, we make the following optimization configurations for the model and its related parameters.

1) Pre-training model. In the pre-training process, we

TABLE II. COMPARATION WITH THE BASELINE APPROACHES FOR CLASSIFICATION ON FEW-SHOT CLASSES

Approaches	Metrics											
	Precison			Recall			F1-scores			Accuracy		
	N=10	N=50	N=100	N=10	N=50	N=100	N=10	N=50	N=100	N=10	N=50	N=100
Deep FP	0.250	0.250	0.255	0.058	0.272	0.317	0.095	0.152	0.201	0.234	0.422	0.401
LSTM FP	0.197	0.329	0.341	0.172	0.401	0.340	0.129	0.212	0.197	0.185	0.221	0.232
AppScanner	0.557	0.728	0.779	0.543	0.699	0.753	0.519	0.704	0.762	0.524	0.723	0.784
ET-BERT	0.505	0.830	0.908	0.425	0.817	0.902	0.415	0.817	0.903	0.425	0.817	0.902
Triplet FP	0.625	0.751	0.837	0.688	0.764	0.840	0.631	0.739	0.838	0.750	0.775	0.838
FineNet	0.860	0.890	0.911	0.863	0.889	0.908	0.860	0.889	0.910	0.860	0.890	0.926

choose the first 1000 raw bytes of every traffic flow as feature sequence. We choose Transformer as the pre-training model in the triplet network. The model consists of a 6-layer Transformer encoder with a 3-head self-attention mechanism. In the triplet network, we choose SGD as the optimizer to minimize the loss function. The learning rate is set to 0.001, the decay value is set to 0.99, the batch value is set to 128, and the total number of training epochs is 15.

2) Few-shot model. In the fine-tuning process, we construct the model consisting of a pre-trained Transformer and a KNN classifier. The parameters of the Transformer are fixed, and here we mainly configure the KNN model. The main parameters are set as follows:

n_neighbors=10. This parameter specifies the number of nearest neighbors to consider when making predictions.

metric='minkowski'. The metric parameter specifies the distance metric used to measure the similarity between data points.

weights='distance'. The weights parameter determines how the neighbors' contributions are weighted when making predictions. 'distance' means the influence of each neighbor is weighted by its inverse distance from the query point.

We implement all of our test in Python environment and use Tensorflow and Keras as the deep learning library. The training uses a 10-fold cross-validation method. We run our experiments on a workstation machine with Windows 10 OS, i9-10850K CPU@3.6-5.2 GHz, 128 GB DDR4 3200 Memory, and Tesla V100 Graphics.

D. Comparation experiment with Baseline

In order to demonstrate the few-shot classification effectiveness of FineNet, we compare it with several state-of-the-art approaches from recent literature. These approaches, including Deep Fingerprinting [20], LSTM Fingerprinting [21], AppScanner [2], ET-BERT [15], Triplet Fingerprinting [12], have made important process on several aspects of encrypted traffic classification. We will test their ability to classify App encrypted traffic under few-shot training sample conditions.

Based on the few-shot dataset in our paper, we compare FineNet with these state-of-the-art methods as baselines, respectively when N (samples of every few-shot class) is 10,50,100. As is shown in Table II, conventional deep learning methods (Deep FP and LSTM FP) perform very poorly in few-shot environments, whose accuracy even does not exceed 25% when N=10. The reason lies that deep learning models usually rely on a large amount of labeled data for training, otherwise the model cannot fully learn the characteristics and patterns of the data. As a model based on machine learning, AppScanner performs better. Its accuracy exceeds 50%, 70%, 75% when N=10, 50, 100 respectively. AppScanner utilizes statistic feature extraction and reinforcement learning techniques, which is less dependent on the number of training samples than deep learning methods. Among the baselines, ET-BERT achieves has the best overall performance when N=50,100，which is closest to FineNet. Its prediction accuracy exceeds 80%(N=50) and 90%(N=100), which may benefit from the good contextual representation ability and strong learning ability of its pre-trained model. However, in the case of N=10, its performance has declined sharply, and the prediction accuracy rate is less than 50%. As a few-shot learning method specifically for Tor traffic classification, Triplet FP also adopts the triplet network architecture. In the case of very few samples (N=10), its accuracy rate (more than 75%) has achieved an overwhelming advantage among the baselines. However, its relatively weak feature representation ability leads to worse performance than ET-BERT in scenarios with slightly more samples (N=50, N=100). In total, the test shows that FineNet significantly surpasses these baseline approaches on all metrics, and its accuracy outperforms the best baseline by 11%(N=10), 7.3%(N=50), 2.4%(N=100). The above results show that the fewer samples, the more obvious the advantages of FineNet. It is largely due to its strong ability of discover the nuances of different traffic flows of new different Apps with insufficient samples.

E. Sensitivity experiment on Different Configuration

In order to optimize performance of the model, we repeatedly test the prediction results of the model with different parameters.

Impact of sample numbers in few-shot dataset. Generally speaking, the number of training samples directly affects the effect of model training. We observe prediction metrics of the model by changing the sample number of each class in the few-shot dataset ($N \in \{1,5,10,20,50,100\}$). As is shown in Fig.3, it is found that basically the larger the number of samples, the higher the accuracy of the model. In the case of very few samples (N=1, 5), FineNet still achieves a good accuracy rate (70.3%, 84.8%). Although it is not an absolutely high value, by comparison, the traditional classification model can hardly operate normally in this extreme situation.

Impact of n_neighbors in KNN classifier. n_neighbors is an important parameter in the KNN classifier, which is used to specify the number of nearest neighbors to consider when predicting. We observe performing metrics of the model by changing the value of the *n_neighbors* parameter (*n_neighbors*=1, 3, 5, 10). As is shown in Fig.4, it is found that basically as *n_neighbors* become larger, the accuracy of the model is improved. When *n_neighbors*=10, the model performs best.

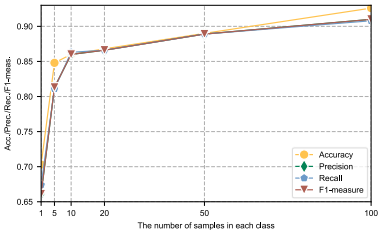

Fig. 3. Impact of sample numbers in few-shot dataset

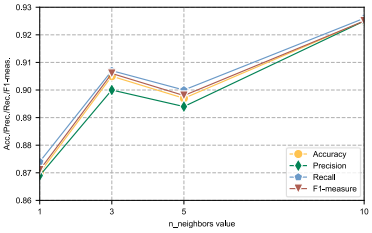

Fig. 4. Impact of n_neighbors in KNN classifier

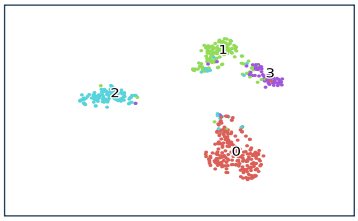

Fig. 5. Distribution of original test samples on the new embedding space

F. Analysis on Representation Ability

In order to intuitively demonstrate the feature representation ability of FineNet in few-shot condition, we levarage t-SNE to visualize the distribution of samples in the new feature space after pre-training processing. t-SNE is used to realize visual display in two-dimensional space by maping high-dimensional data to low-dimensional space, while preserving the similarity relationship between data samples as much as possible. We send test samples of 4 few-shot App classes to the feature extractor in FineNet, and perform t-SNE transformation on the output new features. As is shown in Fig.5, we found that in the new feature space, App samples of the same class are clustered together, and the distance between clusters is farther away. It shows that FineNet is able to discover subtle differences between different types of traffic in a small sample environment, and has played an obvious role in enhancing the representation of traffic characteristics.

VI. Conclusion

In this paper, we proposed a new few-shot learning traffic classification framework based on triplet deep network, and realize the fine-grained classification of App encrypted traffic with very few training samples. To enhance the representation ability of App traffic sequence features, we adopt the Transformer as the base model of triplet network. Transformer's powerful sequence modeling capabilities, particularly its self-attention mechanism, enable FineNet to effectively capture deep relationship among flow bytes. Based on our self-built App traffic dataset, we conducted experiments and it shows that FineNet achieves better performance than existing methods for few-shot App traffic classification. FineNet significantly surpasses baseline approaches on all metrics, and its accuracy outperforms the best baseline by 11%(N=10), 7.3%(N=50), 2.4%(N=100). The limitation of this research is that the model has not been deployed and verified in large-scale real network traffic, which is what we will study in the future.

References

[1] T. Micro, Mobile Security: 80% of Android Apps Now Encrypt Network Traffic by Default. 2019. [Online]. Available: https://www.trendmicro.com/vinfo/hk-en/security/news/mobile-safety/mobile-security-80-of-android-apps-now-encrypt-network-traffic-by-default.

[2] V. F. Taylor, R. Spolaor, M. Conti, and I. Martinovic, "Appscanner: Automatic fingerprinting of smartphone apps from encrypted network traffic," in 2016 IEEE European Symposium on Security and Privacy (EuroS&P), 2016, pp. 439–454.

[3] V. F. Taylor, R. Spolaor, M. Conti, and I. Martinovic, "Robust smartphone app identification via encrypted network traffic analysis," IEEE Transactions on Information Forensics and Security, vol. 13, no. 1, pp. 63–78, 2017.

[4] T. van Ede et al., "Flowprint: Semi-supervised mobile-app fingerprinting on encrypted network traffic," in Network and Distributed System Security Symposium (NDSS), 2020, vol. 27.

[5] C. Liu, Z. Cao, G. Xiong, G. Gou, S.-M. Yiu, and L. He, "Mampf: Encrypted traffic classification based on multi-attribute markov probability fingerprints," in 2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS), 2018, pp. 1–10.

[6] C. Liu, L. He, G. Xiong, Z. Cao, and Z. Li, "Fs-net: A flow sequence network for encrypted traffic classification," in IEEE INFOCOM 2019-IEEE Conference On Computer Communications, 2019, pp. 1171–1179.

[7] X. Wang, S. Chen, and J. Su, "Real network traffic collection and deep learning for mobile app identification," Wireless Communications and Mobile Computing, vol. 2020, 2020.

[8] B. Saltaformaggio et al., "Eavesdropping on Fine-Grained User Activities Within Smartphone Apps Over Encrypted Network Traffic," WOOT, vol. 16, pp. 69–78, 2016.

[9] Y. Fu, H. Xiong, X. Lu, J. Yang, and C. Chen, "Service usage classification with encrypted internet traffic in mobile messaging apps," IEEE Transactions on Mobile Computing, vol. 15, no. 11, pp. 2851–2864, 2016.

[10] D. Li, W. Li, X. Wang, C.-T. Nguyen, and S. Lu, "Activetracker: Uncovering the trajectory of app activities over encrypted internet traffic streams," in 2019 16th Annual IEEE international conference on sensing, communication, and networking (SECON), 2019, pp. 1–9.

[11] I. Guarino, G. Aceto, D. Ciuonzo, A. Montieri, V. Persico, and A. Pescapè, "Contextual counters and multimodal Deep Learning for activity-level traffic classification of mobile communication apps during COVID-19 pandemic," Computer Networks, vol. 219, p. 109452, 2022.

[12] P. Sirinam, N. Mathews, M. S. Rahman, and M. Wright, "Triplet fingerprinting: More practical and portable website fingerprinting with N-shot learning," in Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, 2019.

[13] M. He, X. Wang, J. Zhou, Y. Xi, L. Jin, and X. Wang, "Deep-feature-based autoencoder network for few-shot malicious traffic detection," Secur. Commun. Netw., vol. 2021, pp. 1–13, 2021.

[14] T. Cui, G. Gou, G. Xiong, Z. Li, M. Cui, and C. Liu, "SiamHAN: IPv6 address correlation attacks on TLS encrypted traffic via Siamese Heterogeneous Graph Attention Network," arXiv [cs.CR], 2022.

[15] X. Lin, G. Xiong, G. Gou, Z. Li, J. Shi, and J. Yu, "ET-BERT: A contextualized datagram representation with pre-training transformers for encrypted traffic classification," in Proceedings of the ACM Web Conference 2022, 2022.

[16] Y. Wang, Q. Yao, J. T. Kwok, and L. M. Ni, "Generalizing from a few examples: A survey on few-shot learning," ACM Computing Surveys (CSUR), vol. 53, no. 3, pp. 1–34, 2020.

[17] M. Huisman, J. N. van Rijn, and A. Plaat, "A survey of deep meta-learning," Artificial Intelligence Review, pp. 1–59, 2021.

[18] F. Schroff, D. Kalenichenko, and J. Philbin, "FaceNet: A unified embedding for face recognition and clustering," in 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015.

[19] A. Vaswani et al., "Attention is all you need," Advances in neural information processing systems, vol. 30, 2017.

[20] P. Sirinam, M. Imani, M. Juarez, and M. Wright, "Deep fingerprinting: Undermining website fingerprinting defenses with deep learning," in Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, 2018, pp. 1928–1943.

[21] V. Rimmer, D. Preuveneers, M. Juarez, T. Van Goethem, and W. Joosen, "Automated website fingerprinting through deep learning," arXiv preprint arXiv:1708. 06376, 2017.

[22] S. Li et al., "FusionTC: Encrypted App Traffic Classification Using Decision-Level Multimodal Fusion Learning of Flow Sequence," Wireless Communications and Mobile Computing, vol. 2023, 2023.