FSTC: Dynamic Category Adaptation for Encrypted Network Traffic Classification

Navid Malekghaini, Hauton Tsang, Mohammad A. Salahuddin, Noura Limam, Raouf Boutaba
David R. Cheriton School of Computer Science, University of Waterloo, Ontario, Canada
{nmalekgh, hauton.tsang, mohammad.salahuddin, noura.limam, rboutaba}@uwaterloo.ca

***Abstract*—With the advancement in security and privacy on the Internet, network traffic has become increasingly difficult to classify. Current deep learning (DL)-based encrypted network traffic classification approaches rely on protocol-specific features (e.g., TLS headers) and/or assume that the classification categories (i.e., applications) remain constant over time. However, both the encryption protocols and applications continue to evolve. Therefore, DL models must be retrained from scratch for newer encryption protocols or applications, which makes existing approaches intractable in practice. In this paper, we propose novel Transfer Learning (TL) approaches for introducing new traffic classes to DL models without retraining them from scratch. We also propose a framework named FSTC, which leverages Active Learning (AL) to achieve human-assisted TL for new traffic classes and minimizes the labeled data needed for encrypted network traffic classification. We evaluate our TL and AL approaches using protocol-agnostic features from the publicly available ISCXVPN2016 and QUIC datasets. To the best of our knowledge, neither proposal has been explored before in the existing literature.**

***Index Terms*—Encrypted Traffic Classification, Deep Learning, Active Learning, Transfer Learning, Web Traffic**

I. Introduction

Traffic classification (TC) is essential for a broad range of network operation and management activities. With advancements in cryptography, encrypted communication has become a defacto to preserve the privacy of Internet users. A majority of Web-based services now employ the Hypertext Transfer Protocol Secure (HTTPS) protocol, which encrypts traffic payload using Transport Layer Security (TLS), making TC a challenging endeavour. Numerous works have leveraged deep learning (DL) methods for encrypted network traffic classification (ENTC) (*e.g.*, [1–4]). DL offers benefits, such as automatic feature learning, which helps achieve superior performance in ENTC compared to traditional machine learning (ML). However, DL has its drawbacks, which greatly influences its real-world application in ENTC.

Many ENTC approaches (*e.g.*, [5]) use protocol-specific input features for DL models. Furthermore, though there are approaches that leverage protocol-agnostic methods (*e.g.*, [2, 6]), they still rely heavily on information derived from protocol-specific sources, such as the TLS handshake. Though these approaches may show improved ENTC performance in the context of the TLS protocol, their performance is likely to deteriorate even with minor changes to the protocol. Furthermore, many deep ENTC models (*e.g.*, [1, 2, 5, 6]) have a fixed output dimension (*i.e.*, network traffic classes). However, network traffic categories may change rapidly due to emerging technologies or new use cases, making an already trained DL model obsolete over time due to its inability to categorize new traffic classes.

ISBN 978-3-903176-57-7© 2023 IFIP

Rather than discarding the model and starting over from scratch, Transfer Learning (TL) can be used to allow a model to learn about new traffic classes while retaining the model's knowledge of existing classes. We call this class extension concept Dynamic Category Adaptation (DCA). Indeed, DCA can reduce training time immensely when classifying new traffic. TL can be further extended by integration into an Active Learning (AL) framework. AL incorporates human feedback in the training process of the classifier in cases where large labeled datasets are not available, such as classifying emerging network protocols and technologies. Furthermore the framework can automatically detect possible new classes, and query a human for labels on a small number of traffic samples that the DL model is least confident in. This is the core idea of the "Four Seasons Traffic Classifier (FSTC)", our AL framework for human-assisted TL in support of ENTC.

Our main contributions are:

- We propose novel TL approaches for introducing new traffic classes to DL models using output layer replacement and "Neural Adaptor" techniques
- We compare the performance of our novel TL approaches to state-of-the-art approaches in ENTC using two sets of protocol-agnostic features: (i) standard flow statistics, and (ii) flow time-series information. To the best of our knowledge, this is the first time that TL has been leveraged for DL-based ENTC.
- We propose "FSTC": an AL framework that enables human-assisted TL for ENTC. To the best of our knowledge, this is the first time a framework for DCA using AL has been proposed.

In Section II, we provide a brief survey of works that have inspired our approach. Section III delineates our methodology for FSTC along with its building blocks, such as ENTC models, TL approaches, and DCA methods. In Section IV, we evaluate the performance of our framework. We conclude in Section V and instigate future directions.

II. Related works

A. Encrypted Network Traffic Classification with DL

Recent advancement in DL and its successful application in areas such as computer vision and natural language processing (NLP), have motivated its exploration in other domains, including ENTC. Broadly, DL-based methods for ENTC can be categorized based on the following types of DL models: Multilayer Perceptron (MLP) [2, 7], Stacked Autoencoder (SAE) [1], Convolution Neural Networks (CNN) [2, 6, 8–10], and Long Short-term Memory (LSTM) [2, 6, 9]. We can classify the type of input features these models leverage into four main categories [11]:

1) **Statistical features**: These features include statistical data about a flow, including minimum, maximum, mean, and standard deviation of packet lengths, inter-arrival times, number of packets, etc.
2) **Time series**: These features include a time-series data of packets in a flow. The features of a packet can include the direction of packet, inter-arrival time, size of packet, etc. Such information can be treated as a more fine-grained version of statistical features.
3) **Payload**: These features include raw bytes of the packet payload, mostly above layer 4 of the network protocol stack.
4) **Header**: These features include only raw bytes of header fields in the packet. Typically, these headers are extracted from layer 3 or layer 4 of the network protocol stack.

Akbari *et al.* [2] used a comprehensive multi-part model to achieve the highest accuracy reported on pure encrypted data in ENTC. They used three input feature categories including statistical, time-series, and header features. They excluded the payload category. Their model uses a CNN-based architecture for the header features, similar to [6, 10], and a stacked LSTM for time-series information. They used the standard statistical information captured via CICFlowMeter [12] as the last input features and fed it to an MLP layer. Finally, the output of each part of their model is concatenated and fed into a dense layer followed by a Softmax output layer. The authors claim that their model is protocol agnostic, however, the header bytes are preprocessed to extract features present in specific protocols, *i.e.*, TLS handshake. This may cause model performance to deteriorate for other protocols or newer version of TLS. Furthermore, this type of information leakage in the header during handshake may not be present in newer protocols such as QUIC, which uses additional encryption in the handshake phase. However, the time-series and statistical parts of their model can be applied on all traffic flows regardless of the protocol.

B. Novel Class Approaches

There have been several notable works for TL in traditional ML (*e.g.*, [13, 14]). In our case, for DL-based ENTC, semi-supervised approaches generally apply TL when only a small labeled dataset is available. Iliyasu and Deng [15] leverage a small portion of real unlabeled samples, while employing a Deep Convolutional Generative Adversarial Network (DC-GAN) to generate a large amount of unlabeled samples. They used two baseline models, an MLP and a CNN model, to compare the performance of their approach. They used the public ISCX VPN-NonVPN dataset [12] and a synthetically generated QUIC dataset. They showed that their approach outperforms a similar approach in [16]. Since we are introducing a novel approach that has not been studied in the DL-based ENTC domain, we also use the same method to evaluate our models, *i.e.*, we also evaluate two separate models to obtain a more comprehensive analysis.

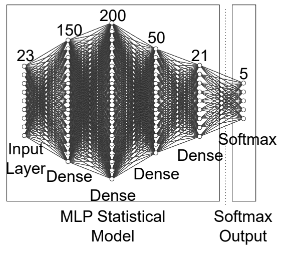
Many works apply techniques from other domains to ENTC. For example, Shi *et al.* [17] explain how existing solutions for DL in the NLP domain can be applied to ENTC. Also, Chen *et al.* [8] converted raw bytes of traffic flow to complete 2-D images and leveraged computer vision techniques for DL-based ENTC. Similarly, there are significant works on TL methods that are not specific to ENTC. For example, Chen and Moschitti [18] investigated several TL approaches in NLP. They first implemented a base model with a few categories in the output layer, then applied TL to construct a target model with additional output classes. In order to mitigate a phenomenon known as catastrophic forgetting, where a model that learns to classify new classes forgets how to classify old ones, they proposed a “Neural Adaptor”.

The authors in [18] assume that the training data for the base model is not available for the target model, and the only data available for the target model is mixed with new and old classes. Due to the high relevance of their approach to ours, we incorporate their approaches with additional variations in our experiments for DCA. However, unlike our work, the authors in [18] did not address how to discover new classes since they focused on TL to known target domains.

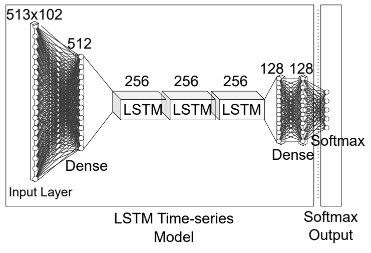
More recently, Zhu and Li [19] proposed a framework for emerging new labels, named SEEN, which is similar to our work. SEEN combined three components: (i) a novel class detector, (ii) a classifier, and (iii) a method for updating the model with the novel class. Unlike our approach, the novel class detection in this framework is done automatically using a tree-based ensemble for anomaly detection. The SEEN framework also used a Label Propagation-based algorithm as the classifier, while our framework uses a DL model. Their primary limitation is that they assume that only a single novel class can be added at a time, therefore, their framework is unable to accommodate for multiple novel classes at the same time. We will use a framework for detecting new classes, which is inspired by SEEN. However, unlike SEEN, our AL approach for labeling novel classes can be used to simultaneously label and train on multiple new classes.

III. Methodology

In this section, we discuss the methodology for our approach. First, we describe our proposed protocol-agnostic ENTC models. Second, we expose the various TL methods used to extend the output dimension of ENTC models for



(a)



(b)

Fig. 1: High-level architectures of DL models with outputs

DCA. Finally, we show how the ENTC and TL methods are integrated into an AL framework, named FSTC.

A. Protocol-Agnostic Encrypted Network Traffic Classification

We propose two different protocol-agnostic approaches for ENTC, each with their own input features and DL model architecture. These separate models and features could be merged into a singular DL model to benefit from both approaches, such as in [2], as these input features are independant of any specific protocol and available for any encrypted network traffic. We assume a flow to be a series of packets that share the source IP, destination IP, source port, destination port, and protocol.

(i) Input features: We utilize two different types of input features, namely standard flow statistics and flow time-series information.

(a) Statistical features: For each flow, we use the standard statistical features available from CICFlowMeter and used in the ISCXVPN2016 dataset [12]. These features include, but are not limited to, duration of the flow, and mean, max, min and standard deviation of forward and backward inter-arrival times. A total number of 23 features are employed.

(b) Time-series features: Similar to [2], for each flow, inter-arrival times, size, and direction are calculated for 1024 packets. Inspired from signal processing, we also compute the Short Time Fourier Transform (STFT) on the time series to help the model detect pattern changes in the input. The STFT parameters of the number of points to overlap between segments is set to 30, the length of each segment is set to 32, and polar coordinates are used.

(ii) Model architectures: We leverage two model architectures each designed for one of the input features. The simpler and more efficient model uses MLP to classify application protocols in encrypted network traffic, while the more sophisticated model uses stacked LSTM to classify services in a more recent encrypted traffic protocol.

(a) MLP model: The high-level architecture of the MLP model is shown in Figure 1a. The values in each layer were set via hyper-parameter search. We use the MLP model with and without the "Softmax output layer" in our TL approaches for MLP model in Section III-B.

(b) LSTM model: The high-level architecture of the LSTM model is depicted in Figure 1b. The LSTM model with and without the "Softmax output layer" are used in our TL approaches for the LSTM model in Section III-B.

(iii) Training strategies: To enable a fair comparison of our results using the LSTM model with the flow time-series model in [2], we use the same training strategy as in [2]. We use an up-sampling strategy to overcome the problem of imbalanced classes in the datasets.

B. Transfer Learning for New Traffic Classes

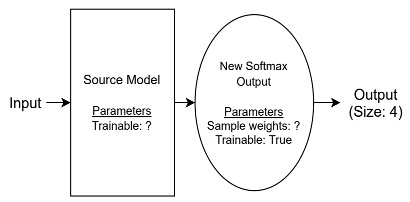
In this section, we present our TL approaches for DCA. We evaluate our TL methods on both the MLP and LSTM models. These approaches can be applied to any source model, other than the ones presented in this paper, as long as they have a Softmax output layer.

(i) TL without Neural Adapters: The first group of approaches for TL is presented in Figure 2. The source model is the already trained model with optimal weights, to which we want to add additional classes. To apply DCA, we need to duplicate the source model and copy its weights. Here, we presume the source model has 3 output classes and the goal is to expand it to 4. The choice of these values is due to the dataset we used. The first approach (*cf.*, Figure 2a) is named "Replace the whole output method". As the name implies, it replaces the old output layer with 3 classes with a completely new output layer with 4 classes. The "Trainable" parameter in a layer decides whether a layer's weights can be updated or not. The source model has the "Trainable: ?" parameter, *i.e.*, whether the weights are fixed or not. The "Sample weights" parameter controls how weights are sampled. If true, the new Softmax layer's weights are sampled from a Normal distribution calculated from the weights of the source model's Softmax output layer.

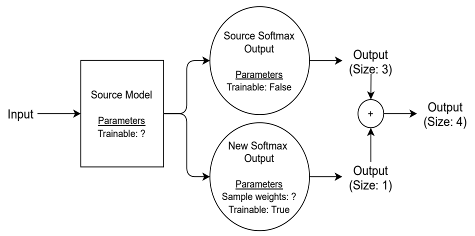
The second TL approach (*cf.*, Figure 2b), inspired from [18], concatenates a new output to the existing Softmax output layer in the source model. This is known as the "Add new dimension output method". The "Trainable" parameter of the Softmax output layer in the source model is set to False, as we know that these outputs already have good performance for classifying points for those 3 classes. The Softmax output layer outputs only for the new class with the "Trainable" parameter set to "True", and the input to the layer is the last hidden layer of the model. The two outputs concatenated create a combined model output with 4 classes.

The third and final approach (*cf.*, Figure 2c), uses the same underlying method as the second approach, but provides more degrees of freedom for classification on the new class by adding a dense layer between the last hidden layer and the output. For this layer, the parameter of "Sample weights" is configurable. This is called the "Add new dimension output with dense layers method".

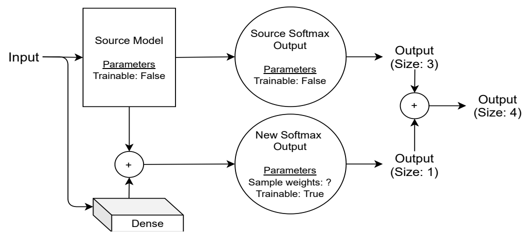
(ii) TL with Neural Adapters: The neural adapter method duplicates the source model to create a new model with re-initialized weights. The source model has fixed weights and the new model has trainable weights. The "BLSTM neural adapter method" (*cf.*, Figure 3b) is the neural adapter from [18] with a minor change. Despite having a retraining phase with both new and old classes, there is still a chance that some catastrophic forgetting will happen. However, after retraining, the TL model



(a) Replace the whole output method

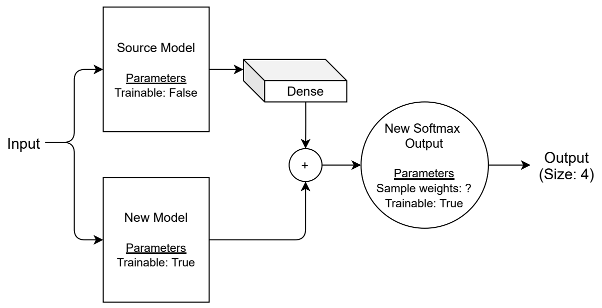


(b) Add new dimension output method

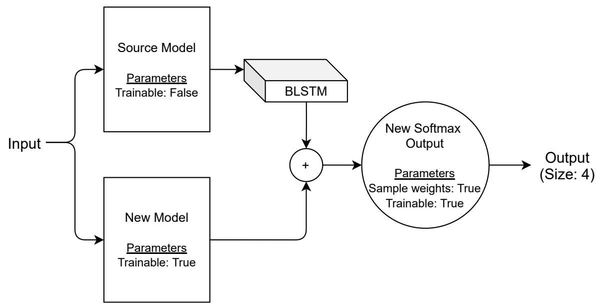


(c) Add new dimension output with dense layers method

Fig. 2: TL methods without neural adapters for adding a new class to the output of source model with 3 classes



(a) Dense neural adapter method



(b) BLSTM neural adapter method

Fig. 3: Approaches with neural adapter for adding a new class to the output of source model with 3 classes

will be significantly better at classifying the new class. The method uses a BLSTM layer to further mitigate the knowledge forgetting (*i.e.*, catastrophic forgetting) learned in the source model when the model is retrained to preserve the context information in the source model. Another neural adapter approach, the "Dense neural adapter method", is depicted in Figure 3a. This approach is basically same as the BLSTM approach but instead of a BLSTM, it uses a dense layer. This can be used to measure the improvements from using a BLSTM to better assess the BLSTM approach. With a dense layer instead of BLSTM, the total parameters and overhead of TL is significantly reduced. The major improvement our methods have over the BLSTM method used in [18] is that the new models have re-initialized weights in the beginning.

C. Active Learning for Dynamic Category Adaptation

Our proposed AL framework, FSTC, integrates both the classifiers from Section III-A and the TL approaches outlined in Section III-B with AL. The framework requires two models, the original traffic classification model, and a clone of the traffic classification model for use in TL. The algorithm is composed of three main steps: (i) identify network traffic most likely to be new classes, (ii) prompt a human to label the traffic, and (iii) use the labeled traffic to update a model.

The first step is accomplished using the models in the framework. By interpreting the output of the model as a set of probabilities, we can construct a *confidence score* by taking the maximum over the probability of all the classes. If the model strongly believes that an input belongs to a certain class, then the probability for that particular class would be high. However, if the model is unsure of the class the input belongs to, then none of the probabilities for any of the classes should be high. The data points with the lowest confidence scores would then be prompted to be labeled in the next step.

The second step is taking the lowest confidence score data points to be labeled by a human. We assume that labeling data points is expensive, hence, only a limited budget b of data points can be labeled. Our framework utilizes both the original ENTC model as well as the cloned TL model to sort data points by confidence score into two lists, one for each model. Then, we take the first b instances of each list and send them to the human for labeling. The least confident points in the original ENTC model mainly serve to find instances of all new classes, while the least confident points in the cloned TL model are for finding the classes that the TL model does not recognize. This is because the original model may be less confident about one particular new class, and thus, the number of instances returned for any other new class may not be sampled if we rely exclusively on the original ENTC model. However, once the TL model has learned the new class detected by the original model, it will increase confidence in that class, so the least confident points returned by the TL model will likely be of a different new class, mitigating this effect.

The final step is to fit the labeled instances to the TL model. We first check if a new class is present in the labeled instances from the human. If there is, then we use one of the DCA techniques in Section III-B to extend the model. Finally, we use the labeled data points to update the model by backpropagation. During the update, we also include a sample of labeled data for old classes to mitigate the effects of catastrophic forgetting, as detailed in Algorithm 1.

Our AL framework has two different variations. The first is the full dataset variation, where the first batch of data points consists of the entire dataset. Each iteration excludes points chosen to be labeled from the batch in the next iteration. The maximum number of iterations must be specified for this variation. The full dataset variation focuses specifically on the least confident points of the models from the entire dataset, which should be the points that are the most critical for improving the accuracy of the TL model.

The second variation is the streaming dataset variation, where the dataset is broken into batches according to a batch size, and the algorithm loops through every batch. The batch size must be specified for this variation. This achieves a more balanced sampling of the data, as the least confident points are sampled from each batch, increasing the diversity of the sampled points. Furthermore, unlike the static dataset variation, the streaming dataset variation can easily be applied in scenarios with continually increasing data, which is true in many real-world production environments where network traffic logs continue to accumulate over time.

Algorithm 1 The Four Seasons Classifier Algorithm

```
$D$: Dataset
$D_{old}$: Small labeled dataset of existing/old classes
$M_{orig}$: Original ENTC model
$M_{tl}$: Cloned version of the ENTC model
$b$: Budget

 1: Initialize empty list $LabeledSamples$
 2: for $TrainingBatch$ in $D$ do
 3:     Initialize empty list $ConfidenceScoreOrig$
 4:     Initialize empty list $ConfidenceScoreTL$
 5:     for $output$ in $M_{orig}(TrainingBatch)$ do
 6:         Append($ConfidenceScoreOrig$, $\max(output)$)
 7:     for $output$ in $M_{tl}(TrainingBatch)$ do
 8:         Append($ConfidenceScoreTL$, $\max(output)$)
 9:     $I_1 \leftarrow$ TakeFirst(Argsort($ConfidenceScoreOrig$), $b$)  ▷ Get the indices of first $b$ least confident samples according to original model
10:     $I_2 \leftarrow$ TakeFirst(Argsort($ConfidenceScoreTL$), $b$)  ▷ Get the indices of first $b$ least confident samples according to TL model
11:     $I \leftarrow$ Union($I_1, I_2$)
12:     $D_{sample} \leftarrow TrainingBatch[I]$  ▷ Store data points corresponding to the indices in $I$ to $D_{sample}$
13:     Append($LabeledSamples$, SendToLabel($D_{sample}$))
14:     $UpdateSamples \leftarrow$ Union($LabeledSamples$, $D_{old}$)  ▷ Train the labeled samples along with old data to mitigate catastrophic forgetting
15:     UpdateModel($M_{tl}$, $UpdateSamples$)
```

We also experiment with adapting the AL algorithm to do unsupervised DCA, which we call "Unsupervised FSTC". This approach is similar to the SEEN framework [19] in that it assumes only one new class is presented at a time. Thus, we can assume that all the least confident points belong to one new class and label them as such. Then, we can use these points to fit the model to perform unsupervised DCA.

To implement "Unsupervised FSTC", we use the full dataset variation, but only perform a single iteration. We do not want to sample by batches since this will increase the likelihood of the model picking points that do not belong to the new class. Since the original ENTC model is not updated for each iteration, there is no need to iterate through the dataset more than once because the resulting selected points will be identical. We only use the original classifier to generate the confidence score instead of both classifiers. Finally, we can assume that all the instances selected are new class instances and update the model accordingly. As before, we mix these labeled instances with a labeled sample of old data to mitigate the effects of catastrophic forgetting. The algorithm is detailed in Algorithm 2.

DL Model	Dataset	Accuracy (%)	W. Avg F1 (%)
Akbari *et al.* [2]	ISCXVPN2016 [12]	65	64
MLP Model		69	69
Akbari *et al.* [2]	QUIC [16]	99.37	99.47
LSTM Model		97.9	97.9

TABLE I: Accuracy and weighted average F1-score of models

Algorithm 2 Unsupervised Four Seasons Classifier Algorithm

```
$D$: Dataset
$D_{old}$: Small labeled dataset of existing/old classes
$M_{orig}$: Original ENTC model
$M_{tl}$: Cloned version of the ENTC model
$b$: Budget

1: Initialize empty list $ConfidenceScoreOrig$
2: for $output$ in $M_{orig}(D)$ do
3:     Append($ConfidenceScoreOrig$, $\max(output)$)
4: $I \leftarrow$ TakeFirst(Argsort($ConfidenceScoreOrig$), $b$)  ▷ Get the indices of first $b$ least confident samples according to original model
5: $D_{sample} \leftarrow D[I]$  ▷ Store data points corresponding to the indices in $I$ to $D_{sample}$
6: $LabeledSamples \leftarrow$ Label($D_{sample}$, $newClass$)  ▷ Label all points in $D_{sample}$ with the new class
7: $UpdateSamples \leftarrow$ Union($LabeledSamples$, $D_{old}$)  ▷ Train the labeled samples along with old data to mitigate catastrophic forgetting
8: UpdateModel($M_{tl}$, $UpdateSamples$)
```

IV. Evaluation

In this section, we present details regarding the employed datasets. We then discuss the performance of the protocol-agnostic ENTC models with all 5 classes. After that, we evaluate the performance of our TL approaches when we add the fourth class to a three-output class source model and compare it to the base model without TL. Finally, we use the framework to add the two remaining classes to the source model, and analyze the performance by comparing it to the base model's performance without TL and dynamic category adaptation.

A. Protocol Agnostic Encrypted Network Traffic Classification

(i) Dataset description: We employ two publically available datasets in our evaluations, namely ISCXVPN2016 [12] and QUIC [16] datasets. For both datasets, we use 80% of the data for model training and the remaining 20% for validation.
(a) ISCXVPN2016 dataset: This dataset includes the statistical features described in Section III-A. We use 5 types of applications as the output classes: CHAT, STREAMING, VOIP, P2P, and FT. Each application traffic is encrypted and routed through OpenVPN.
(b) QUIC dataset: This dataset includes the time-series features described in Section III-A. There are 5 type of applications as the output classes: GoogleDoc, GoogleDrive, GoogleMusic, GoogleSearch, and YouTube. The QUIC protocol uses UDP and brings the encryption to the transport-layer.

Moreover, it is more comprehensively encrypted in comparison to the TLS protocol.

(ii) Performance Evaluation: When comparing the performance of the "MLP Model" with a "Statistical sub-model" that encapsulates the statistical section of the model from [2], we achieved a 4% increase in accuracy and a 5% increase in F1-score with our modified approach for dense layers. The results of the two models are shown in Table I.

The performance results of our "LSTM Model" along with the flow time-series model from [2] are also presented in Table I. When compared to each other, we can see that our "LSTM Model" lags behind by 1.47% and 1.57% in accuracy and F1-score, respectively. This indicates that STFT does not help to improve performance, possibly due to not preserving all the information present in the time-series input features.

B. Transfer Learning in DL for New Traffic Classes

We now evaluate the two groups of approaches explained in Section III-B for our two base models. Because of the imbalanced dataset, the accuracy metric would not be a fair metric for evaluating the model's performance. Therefore, the performance metrics evaluated in this section are weighted average F1-scores, unless specified otherwise. Weighted average F1-score is a weighted metric that better reflects the performance of a model across all classes. The retraining phase of all TL methods use Adam optimizer with a learning rate of 0.001, and 10 epochs.

(i) MLP Base Model: The results for the first group of TL approaches (*cf.*, Section III-B) is summarized in Table II. The "sample size" hyper-parameter in the Tables is the total number of available flows for the retraining phase of the TL model. We used only a small part of the available flows for retraining the model on the new class. The reasoning behind this is that it is likely that emerging new traffic classes have less available flows to train the model. The number of available flows for the new traffic class is the equal to the $samplesize/divide$, with other flows being from old traffic classes. Sampling is done randomly from the old and new class traffic data in the dataset. The decision for the best values of these hyper-parameters is dependent on the scenario that these TL approaches and source models are being used in. Therefore, we examined the effects of changing the proportions.

The results for the second group of TL approaches (*cf.*, Section III-B) are presented in Table III. For each approach in both groups, all possible combinations of parameters of "Trainable" and "Sample weights" were tested to examine their effects on the performance. The "-" values for "Trainable" and "Sample weights" in the tables means that they are either fixed in the model's design and not configurable or are not applicable. We also tested the performance of 3 different proportions of available flows from the new class to the existing classes. The first column shows the performance of TL with sample size: 120, divide: 4. The second column (*i.e.*, sample size: 90, divide: 3) has a sample that reduces the proportion of samples from existing classes while having the same number of new class samples as the previous column. The third column (*i.e.*, sample size: 120, divide: 3) increases the proportion of samples from the new class while preserving the total number of samples from the first column. The maximum value of the "Average" column for each TL method is colored in green. If the first and second maximum values of "Average" column for a approach are very close (*i.e.*, less than 3% difference), both of them are marked.

By looking at the maximum general performance of all the models (*i.e.*, green cells in "Average" column), we can infer that the two neural adapter methods (*cf.*, Table III) perform better than all the approaches without neural adapters (*cf.*, Table II). It can be seen that on average, the "BLSTM neural adapter method" performs around 2% better than the "Dense neural adapter method", which confirms our assumption that a BLSTM layer has higher performance than a Dense layer in a neural adapter as it can retain context information in the source model. However, this performance boost is not that significant, so perhaps for some models using a Dense neural adapter would be sufficient because of the lower number of parameters it has in comparison to the more complex BLSTM neural adapter. The BLSTM neural adapter method in [18] is also presented in Table III. The average performance of their model is 38%, which is 47% lower than the performance we achieve with our "BLSTM neural adapter method". This indicates that our modified approach for the new model we presented has a significant performance boost over the unmodified model in [18].

Regarding the TL approaches without neural adapters (*cf.*, Table II), the "Replace the whole output method" is the best-performing, achieving the two top "Average" scores of 78.33% and 77.33%. In this approach, the output Softmax layer is replaced with a different layer with more outputs. Due to the fact that the data is a mixture of existing and new class, the weights are tuned for both classes and thus the model gets good results distinguishing between the different traffic classes. The other two approaches were not nearly as performant, with "Average" scores of 33.66% and 31% for "Add new dimension output method" and "Add new dimension output with dense layers method", respectively. The "Add new dimension output method" approach is from [18], while the "Add new dimension output with dense layers method" is its modified version. One possible explanation is that since there is only one new class but the data is a mixture of new and existing classes, the model wants to update the weights of the Softmax output layer for all classes. However, as the model's output layer has only one output dimension with weights that are not fixed (*i.e.*, only the new Softmax output layer), the model cannot be trained properly. Therefore, we see degraded performance.

As a comparison, base model results are also provided in Tables II and III. The base model is the "MLP Model" that has been trained from scratch with four classes on the entire dataset without any TL methods. The weighted average F1-score for this model is 86%. By comparing this score to the neural adapter TL approaches, we can see that they are very

Approach	Model name	Parameters		sample size: 120, divide: 4	sample size: 90, divide: 3	sample size: 120, divide: 3	Average
		Trainable	sample weights	W. average F1-score (%)	W. average F1-score (%)	W. average F1-score (%)	(%)
Replace the whole output method	swap_output_layer	TRUE	TRUE	**82**	72	78	77.33
	swap_output_layer_notrain	FALSE	TRUE	36	17	**72**	41.66
	swap_output_layer_nosampleweights	TRUE	FALSE	82	74	79	78.33
	swap_output_layer_notrain_nosampleweights	FALSE	FALSE	37	54	**58**	49.66
Add new dimension output method	concat_output_layer	TRUE	TRUE	31	32	31	31.33
	concat_output_layer_notrain	FALSE	TRUE	31	31	31	31
	concat_output_layer_nosampleweights	TRUE	FALSE	38	30	33	33.66
	concat_output_layer_notrain_nosampleweights	FALSE	FALSE	31	31	31	31
Add new dimension output with dense layers method	concat_with_hidden_notrain	-	TRUE	18	31	31	26.66
	concat_with_hidden_notrain_nosampleweights	-	FALSE	31	31	31	31
N/A	MLP Statistical Model with four classes (base model)	-	-	-	-	-	86

TABLE II: Performance evaluation of the first group of TL approaches for DCA using the "MLP Model"

Approach	Model name	Parameters		sample size: 120, divide: 4	sample size: 90, divide: 3	sample size: 120, divide: 3	Average
		Trainable	sample weights	W. average F1-score (%)	W. average F1-score (%)	W. average F1-score (%)	(%)
Dense neural adapter method	dense_neuraladapter	-	TRUE	85	82	79	82
	dense_neuraladapter_nosampleweights	-	FALSE	85	86	80	83.66
BLSTM neural adapter method	blstm_neuraladapter	-	-	86	85	84	85
Chen and Moschitti [18] BLSTM neural adapter	chen_blstm_neuraladapter	-	-	36	38	40	38
N/A	MLP Statistical Model (base model)	-	-	-	-	-	86

TABLE III: Performance evaluation of the second group of TL approaches for DCA using the "MLP Model"

close. The "BLSTM neural adapter method" has an "Average" score of 85% and the "Dense neural adapter method" has a score of 83.66%. These results show that our TL approaches with neural adapters can achieve similar performance with only a small portion of data for a new class and a limited number of old data samples. We are able to achieve the same results as the base model that trained on a much larger number of samples. For the TL approaches without neural adapters, the "Replace the whole output method" has an "Average" of 78.33%, which is roughly 7.77% lower than the base model. However, with parameter tuning, we can achieve an 82% weighted average F1-score for this model (*cf.*, row one of Table II in bold font), which is only 4% lower than the base model. We also consider this a success because not only does the model have all of the advantages of the TL neural adapter methods, the number of model parameters for this method is way lower than the neural adapter methods, which makes it extremely fast and lightweight.

(ii) LSTM Base Model: The "LSTM Model" performance results are summarized in Tables IV and V. The poor-performing approaches from the "MLP Model" were excluded. We also included the accuracy metric since the QUIC dataset used for this model is more balanced.

The best method here is the "Replace the whole output method" which does not use a neural adapter approach. The average accuracy and weighted average F1-score for the best approach are 72.3% and 63.8%, respectively. Compared to the base model, which is the "LSTM Model" trained from scratch using all the data for four classes, it lags behind by around 26.7% in accuracy and 35.2% in weighted average F1-score. It is important to note that the "LSTM Model" is very complex and costly to train from scratch. Moreover, the QUIC protocol is also one of the most sophisticated algorithms for encrypted communication in the Internet. Therefore, considering the tiny amount of data used to train the new class and the low training overhead with this TL approach, the results are particularly relevant for resource-constrained use cases.

The "BLSTM neural adapter method" comes in at the second place with an approximately 10% performance gap compared with the "Replace the whole output method". The BLSTM neural adapter in [18] also shows the same performance as our "BLSTM neural adapter method". Therefore, the catastrophic forgetting that was solved by our model's modified approach in the "MLP Model" does not have that much of an effect here. The reason behind this may be that the source model itself already uses 3 stacked LSTM layers (*cf.*, Figure 1b), with the first and second one already acting as a BLSTM layer in the "BLSTM neural adapter method". This may also be the reason why the "Replace the whole output method" performs well when using "LSTM Time-series Model" as the source model.

C. Active Learning for Dynamic Category Adaptation

We present the results of three different experiments utilizing our AL learning framework: (i) Static dataset AL, (ii) Streaming dataset AL, and (iii) Unsupervised learning using the AL framework. Each of these experiments are conducted using the "MLP Model", and the three highest-performing techniques presented in Section IV-B. For the AL experiments, we test the model's ability to adapt both one and two additional classes. For the two-class case, we have one experiment that utilizes the same parameters as the one-class case, and another experiment which adjusts various hyper-parameters to optimize for the two-class case, which is denoted "2 (Adjusted)". The results of the experiments are detailed in Table VI.

(i) Static dataset AL: The static dataset AL variation samples a budget b of points from the entire dataset using both the original classifier as well as the TL classifier. In the evaluation of the static dataset AL, we decided to use the following AL hyper-parameters: a budget size of 3, and 5 iterations of the dataset. Since we sample from both classifiers, this means we sample 6 points per iteration. Hence, the total number of instances to be prompted by a human to label is 30. We can then add a sample of 90 old data points in order to match the most optimal ratio derived in the previous section, which is a sample of 120 data points of which 1/4 of them are new classes. We disable sample weights for all models and train for 10 epochs. For the two-class adjusted case, our model uses 10 iterations instead of 5. All other parameters remain the same.

Approach	Model name	Parameters		W. average F1-score (%)	Accuracy (%)
		Trainable	sample weights		
Replace the whole output method	lstm_swap_output_layer	TRUE	TRUE	36.9	52.1
	lstm_swap_output_layer_notrain	FALSE	TRUE	4.4	11.5
	lstm_swap_output_layer_nosampleweights	TRUE	FALSE	65.8	71.9
	lstm_swap_output_layer_notrain_nosampleweights	FALSE	FALSE	63.8	72.3
N/A	LSTM Time-series Model with four classes (base model)	-	-	99	99

TABLE IV: Evaluation of the first group of of TL approaches for DCA using the "LSTM Model"

Approach	Model name	Parameters		W. average F1-score (%)	Accuracy (%)
		Trainable	sample weights		
Dense neural adapter method	lstm_dense_neuraladapter	-	TRUE	3.1	11.5
	lstm_dense_neuraladapter_nosampleweights	-	FALSE	18.1	30.5
BLSTM neural adapter method	lstm_blstm_neuraladapter	-	-	51.21	62.84
Chen and Moschitti [18] BLSTM neural adapter	lstm_chen_blstm_neuraladapter	-	-	51.2	62.84
N/A	LSTM Time-series Model with four classes (base model)	-	-	99	99

TABLE V: Evaluation of the second group of TL approaches for DCA using the "LSTM Model"

AL Variation	# New Classes	Method	Model name	W. average Precision (%)	W. average Recall (%)	W. average F1-score (%)
Static dataset AL	1	Replace the whole output method	swap_output_layer_nosampleweights	86	84	84
		Dense neural adapter method	dense_neuraladapter_nosampleweights	85	85	85
		BLSTM neural adapter method	bltsm_neuraladapter	86	86	86
	2	Replace the whole output method	swap_output_layer_nosampleweights	70	63	59
		Dense neural adapter method	dense_neuraladapter_nosampleweights	65	58	56
		BLSTM neural adapter method	bltsm_neuraladapter	67	64	61
	2 (Adjusted)	Replace the whole output method	swap_output_layer_nosampleweights	72	69	67
		Dense neural adapter method	dense_neuraladapter_nosampleweights	68	65	63
		BLSTM neural adapter method	bltsm_neuraladapter	71	67	65
Streaming dataset AL	1	Replace the whole output method	swap_output_layer_nosampleweights	86	84	85
		Dense neural adapter method	dense_neuraladapter_nosampleweights	85	85	85
		BLSTM neural adapter method	bltsm_neuraladapter	87	86	86
	2	Replace the whole output method	swap_output_layer_nosampleweights	71	68	67
		Dense neural adapter method	dense_neuraladapter_nosampleweights	71	64	63
		BLSTM neural adapter method	bltsm_neuraladapter	70	66	64
	2 (Adjusted)	Replace the whole output method	swap_output_layer_nosampleweights	68	65	63
		Dense neural adapter method	dense_neuraladapter_nosampleweights	71	68	67
		BLSTM neural adapter method	bltsm_neuraladapter	64	63	63
Unsupervised Learning	1	Replace the whole output method	swap_output_layer_nosampleweights	61	58	55
		Dense neural adapter method	dense_neuraladapter_nosampleweights	64	62	60
		BLSTM neural adapter method	bltsm_neuraladapter	73	60	63
Baseline (with 5 classes)	-	-	MLP Statistical Model	74	69	69

TABLE VI: Performance evaluation of the various AL variations

For one additional class, the "BLSTM neural adapter method" is the highest-performing model. We expected the "BLSTM neural adapter method" to perform the best as our results in Section IV-B show that method to be the best-performing overall. However, the other two methods have comparable performance. With two additional classes with non-adjusted settings, the BLSTM still performs quite well overall, but model performance was significantly decreased. This is likely because applying the "BLSTM neural adapter method" twice results in double the number of parameters that need to be trained, since both the source model and target model of the original model need to be cloned in order to create a new target model. After adjusting the iterations to increase the number of labeled samples, the "Replace the whole output method" becomes the most optimal. This may be due to the fact that this method produces a model with the least amount of parameters to be trained, and thus requires less data to train than other methods.

(ii) Streaming dataset AL: The streaming dataset AL variation samples a budget b of data points in batches of fixed size until the end of the dataset is reached. Unlike the static dataset AL variation, the number of iterations for the streaming dataset AL variation depends on the total size of the training dataset s_{train} and the batch size bs, *i.e.*, ceiling(s_{train}/bs). After reserving the sample of 90 old data points, the number of training instances was 1264. Thus, we set a batch size of 253 to ensure that the number of iterations is 5, which is the same as the number of iterations in the static dataset AL experiment. The other parameters are identical to the static dataset AL variation experiment. For the two-class adjusted case, our model uses a batch size of 127 instead, ensuring that the number of iterations is 10. This matches the number of iterations in the adjusted case of the static dataset AL variation.

For one additional class, the results show that the "BLSTM neural adapter method" is the best performing method yet again, having the highest weighted average precision, recall, and F1-score. However, the other two methods are within 1-2% for all three statistics. This result is also in line with the results from Section IV-B. For two classes, however, the "Replace the whole output method" is the clear winner. This may again be due to the fact that each application of the "BLSTM neural adapter method" causes the number of trainable parameters to double. Thus, the large number of parameters may be reducing the performance of the model. After adjusting batch size, the "Dense neural adapter method" has the highest efficacy in this case, which is surprising as the number of parameters is comparable to the "BLSTM neural adapter method". We speculate that the "Replace the whole output method" may not be robust to smaller batches as they increase the likelihood of sampling data that is not part of the new class.

(iii) Unsupervised learning: The unsupervised model samples a budget b of data points from the entire dataset using only the original classifier. Unlike the AL approaches, the unsupervised learning approach only samples a budget b of

points from the entire dataset once. Thus, to ensure that the number of labeled samples is identical to other experiments, we set b to be 30. The 30 data points sampled this way are labeled as a new class, As before, we combine this with the sample of 90 old data points before updating the model. Again, the "BLSTM neural adapter method" performs the best overall, achieving a weighted average F1-Score of 63%, although the "Dense neural adapter method" has a slightly higher recall.

Overall, both the static dataset AL and streaming dataset AL variations are very similar in performance in both the one-class and two-class cases. However, the streaming dataset AL variation only iterates through the dataset once, which reduces training time significantly, especially on large datasets. Additionally, the streaming dataset variation can be used in streaming environments where dataset is constantly growing. Hence, due to practical considerations, we consider the streaming dataset variation of "FSTC" to be the best AL approach. For one new class, the "BLSTM neural adapter method" seems to perform the best overall. However, with two new classes, the "Replace the whole output method" seems to perform more optimally as the number of parameters does not increase exponentially with each additional class, and thus the model converges faster with the addition of more classes.

Comparing the two-class cases with the performance of the base model, we find that the performance of both AL models are comparable to the base model, with only a slightly lower weighted average precision, recall, and F1-score. This is expected, since the AL models are trained on a much smaller dataset than the base model. The AL model only requires labeling 30 data points in the non-adjusted case, and 60 data points in the adjusted-case. Even if the sample of 90 data points from old classes are factored in, the resulting dataset only consists of 120-150 data points in total, whereas the full training set consists of 1264 data points. Thus, achieving similar performance to the base model is already quite impressive.

The performance of the unsupervised learning approach, on the other hand, is significantly worse compared to AL approaches, with a weighted average F1-Score that is lower by more than 20%. Therefore, we believe this approach may require further adjustment to meet performance milestones for learning new categories.

V. Conclusion

In this paper, we introduced two groups of novel TL approaches (*i.e.*, neural adapter and non-neural adapter) for DCA applied on two distinct models based on the state of the art. These models were evaluated using two public datasets with different protocol-agnostic input features. We showcased that our models can achieve similar results to baseline by leveraging TL using only 120 samples (*i.e.*, flows), only 30 of which corresponded to the new class. Our TL approaches used only around 7% of the samples from all classes and only 3.7% of the samples available for the new class during the retraining phase. We also proposed an AL framework and evaluated two different variations of AL, as well as unsupervised learning using a small modification of the AL approach. We found the streaming dataset AL to be the best due to practical considerations. Our framework achieves similar performance to the base model despite being trained on a dataset that is nearly 10x smaller.

For future work, we believe that our novel TL approach can be further improved when applied to a stacked LSTM model. Additional experimentation with weight transfer algorithms and sample weights can also be conducted to improve the performance of the TL methods. For the AL framework, additional experiments may be conducted on the performance of other class detection methods, such as using Isolation Forest [20] or clustering-based approaches to detect new classes.

References

[1] M. Lotfollahi, M. Jafari Siavoshani *et al.*, "Deep packet: A novel approach for encrypted traffic classification using deep learning," *Soft Computing*, vol. 24, no. 3, pp. 1999–2012, 2020.

[2] I. Akbari, M. A. Salahuddin *et al.*, "A look behind the curtain: Traffic classification in an increasingly encrypted web," *ACM on Measurement and Analysis of Computing Systems*, vol. 5, no. 1, pp. 1–26, 2021.

[3] N. Malekghaini, E. Akbari *et al.*, "Deep learning for encrypted traffic classification in the face of data drift: An empirical study," *Computer Networks*, vol. 225, p. 109648, 2023.

[4] N. Malekghaini, "Adapting to data drift in encrypted traffic classification using deep learning," Master's thesis, 2023. [Online]. Available: http://hdl.handle.net/10012/19058

[5] P.-O. Brissaud, J. Francçis *et al.*, "Transparent and service-agnostic monitoring of encrypted web traffic," *IEEE Transactions on Network and Service Management*, vol. 16, no. 3, pp. 842–856, 2019.

[6] S. Rezaei, B. Kroencke, and X. Liu, "Large-scale mobile app identification using deep learning," *IEEE Access*, vol. 8, pp. 348–362, 2019.

[7] A. Khajehpour, F. Zandi *et al.*, "Deep inside tor: Exploring website fingerprinting attacks on tor traffic in realistic settings," in *International Conf. on Computer and Knowledge Engineering*, 2022, pp. 148–156.

[8] Z. Chen, K. He *et al.*, "Seq2img: A sequence-to-image based approach towards ip traffic classification using convolutional neural networks," in *IEEE Big Data*, 2017, pp. 1271–1276.

[9] N. Malekghaini, E. Akbari *et al.*, "Data drift in dl: Lessons learned from encrypted traffic classification," in *IFIP Networking*, 2022, pp. 1–9.

[10] W. Wang, M. Zhu *et al.*, "End-to-end encrypted traffic classification with one-dimensional convolution neural networks," in *IEEE Int. Conf. on Intelligence and Security Informatics*, 2017, pp. 43–48.

[11] S. Rezaei and X. Liu, "Deep learning for encrypted traffic classification: An overview," *IEEE Comm. Magazine*, vol. 57, no. 5, pp. 76–81, 2019.

[12] G. Draper-Gil, A. H. Lashkari *et al.*, "Characterization of encrypted and vpn traffic using time-related features," in *International Conf. on Information Systems Security and Privacy*, 2016, pp. 407–414.

[13] G. Sun, L. Liang *et al.*, "Network traffic classification based on transfer learning," *Computers & electrical engineering*, vol. 69, pp. 920–927, 2018.

[14] F. Shang, S. Li, and J. He, "Improved application of transfer learning in network traffic classification," in *Journal of Physics: Conference Series*, vol. 1682, no. 1, 2020, p. 012011.

[15] A. S. Iliyasu and H. Deng, "Semi-supervised encrypted traffic classification with deep convolutional generative adversarial networks," *IEEE Access*, vol. 8, pp. 118–126, 2019.

[16] S. Rezaei and X. Liu, "How to achieve high classification accuracy with just a few labels: A semi-supervised approach using sampled packets," *arXiv preprint arXiv:1812.09761*, 2018.

[17] Y. Shi, D. Feng *et al.*, "A natural language-inspired multilabel video streaming source identification method based on deep neural networks," *Signal, Image and Video Processing*, vol. 15, pp. 1161–1168, 2021.

[18] L. Chen and A. Moschitti, "Transfer learning for sequence labeling using source model and target data," in *AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, 2019, pp. 6260–6267.

[19] Y.-N. Zhu and Y.-F. Li, "Semi-supervised streaming learning with emerging new labels," in *AAAI Conf. on AI*, 2020, pp. 7015–7022.

[20] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest," in *2008 Eighth IEEE International Conference on Data Mining*, 2008, pp. 413–422.