4-1-2023

Darknet traffic classification and adversarial attacks using machine learning

Nhien Rust-Nguyen
San Jose State University

Shruti Sharma
San Jose State University

Mark Stamp
San Jose State University, mark.stamp@sjsu.edu

Recommended Citation

Nhien Rust-Nguyen, Shruti Sharma, and Mark Stamp. "Darknet traffic classification and adversarial attacks using machine learning" *Computers and Security* (2023). https://doi.org/10.1016/j.cose.2023.103098

Computers & Security 127 (2023) 103098

Contents lists available at ScienceDirect

Computers & Security

journal homepage: www.elsevier.com/locate/cose

Darknet traffic classification and adversarial attacks using machine learning

Nhien Rust-Nguyen, Shruti Sharma, Mark Stamp*

Department of Computer Science, San Jose State University, United States

ARTICLE INFO

Article history:
Received 12 June 2022
Revised 22 October 2022
Accepted 9 January 2023
Available online 14 January 2023

Keywords:
Darknet
Classification
Adversarial attacks
Convolutional neural network
Auxiliary-Classifier generative adversarial network
Random forest

ABSTRACT

The anonymous nature of darknets is commonly exploited for illegal activities. Previous research has employed machine learning and deep learning techniques to automate the detection of darknet traffic in an attempt to block these criminal activities. This research aims to improve darknet traffic detection by assessing a wide variety of machine learning and deep learning techniques for the classification of such traffic and for classification of the underlying application types. We find that a Random Forest model outperforms other state-of-the-art machine learning techniques used in prior work with the CIC-Darknet2020 dataset. To evaluate the robustness of our Random Forest classifier, we obfuscate select application type classes to simulate realistic adversarial attack scenarios. We demonstrate that our best-performing classifier can be degraded by such attacks, and we consider ways to effectively deal with such adversarial attacks.

1. Introduction

Most of us are familiar with the Internet and the World Wide Web (WWW, or web). We regularly access both using web browsers or other networked applications to share information publicly, guided by search engine indexing of the Domain Name System (DNS) over globally bridged Internet Protocol (IP) networks. This publicly accessible and indexed address space is known as the surface web or clearnet. In contrast, the WWW address space which is not indexed by search engines but still publicly accessible is known as the deep web. Private networks within the deep web or networks comprised of unallocated address space are known as darknets and collectively termed the dark web. Fig. 1 illustrates the relationship between these layers of the Internet.

The dark web is reached by an overlay network requiring special software, user authorization, or non-standard communication protocols (Demertzis et al.). Many darknets afford users anonymity during communication and thus facilitate criminal activities, including hacking, media piracy, terrorism, human trafficking, and child pornography (Branwen et al., 2015; Sarwar et al., 2021). Researchers have illuminated darknet traffic with machine learning and deep learning techniques, to better identify and inhibit these criminal activities. The research presented in this paper strives to contribute by promoting accurate classification of traffic features from the well-studied CIC-Darknet2020 (Lashkari et al., 2020) dataset, which is a collection of traffic features from two darknets, namely, The Onion Router (Tor) and a Virtual Private Network (VPN). This dataset also includes corresponding traffic generated over clearnet sessions using the same applications.

We consider a wide variety of classic machine learning techniques, as well as modern neural networking architectures. We also represent traffic features as grayscale images and apply image-based deep learning architectures as classifiers, namely, Convolutional Neural Networks (CNN) and Auxiliary-Classifier Generative Adversarial Networks (AC-GAN). To assess the issue of extreme class imbalance within the CIC-Darknet2020 dataset, we explore data augmentation—specifically, we consider both the generative network of our AC-GAN model, as well as Synthetic Minority Oversampling Technique (SMOTE). Our results show that Random Forest is the most effective among the models tested, both for classifying traffic type and for classifying the underlying application types. We also find SMOTE is beneficial for fine tuning our models, the best of which is a Random Forest (RF).

Having established baseline classification performance, we consider the robustness of our RF classifier in some detail by approaching the problem of darknet traffic detection adversarially. From the perspective of an attacker, we obfuscate the application classes in an attempt to evade detection. As a proof-of-concept, we apply an encoding scheme to transform class features using probability analysis of the CIC-Darknet2020 dataset. We strongly

* Corresponding author.
E-mail address: mark.stamp@sjsu.edu (M. Stamp).

https://doi.org/10.1016/j.cose.2023.103098

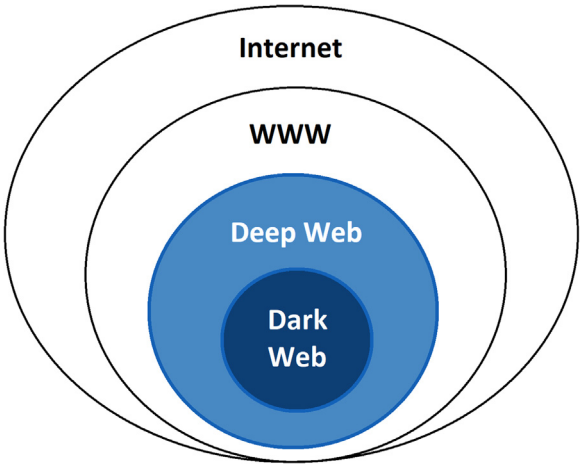

Fig. 1. Layers of the Internet (Demertzis et al., 2021).

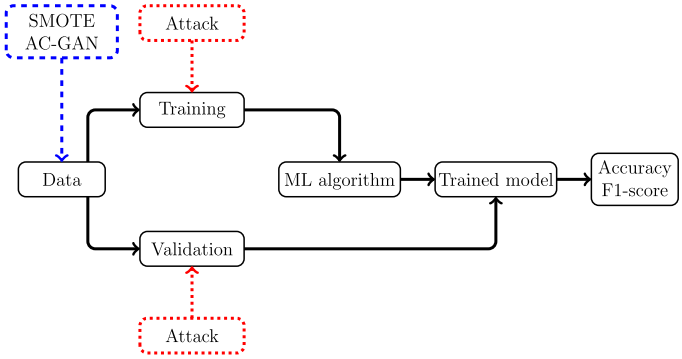

Fig. 2. Overview of experiments.

correlate the resulting RF confusion with our obfuscation technique for three attack scenarios, assuming few limitations for traffic modification. We then assess the strength of our obfuscation technique with one defense scenario, by which we demonstrate that we can restore the performance of the RF classifier despite duress. We find that sufficient statistical knowledge of network traffic features can empower either the classification or obfuscation tasks.

A high-level overview of our experiments is provided in Fig. 2. After some limited initial data cleaning, for each experiment, we partition the dataset under consideration into training and validation sets. In the base case, a specific machine learning model is trained, based on the training set, with the validation set used to compute accuracy and F1-score statistics. As mentioned above, we consider data augmentation using SMOTE. We also conduct experiments using the generator module of AC-GAN to produce synthetic data, which can be viewed as another form of data augmentation. The three adversarial attack scenarios mentioned above assume that the attacker can manipulate the training data, the validation data, or both. In Section 4.2, we discuss these attack scenarios in detail, and explain why they are realistic threats.

The remainder of this paper is structured as follows. Section 2 gives a brief background discussion of Tor and VPN, and considers related work on darknet traffic detection. Section 3 describes the dataset used in our experiments and outlines our experimental methodology. Section 4 provides background knowledge on the machine learning techniques used in our experiments and gives implementation details. Section 5 discusses the results of our experiments. Lastly, Section 6 summarizes our research and considers possible directions for future work.

2. Background

In this section, we first discuss the two broad categories of data in our dataset, namely, Tor and VPN traffic. Then we discuss the most relevant examples of related work.

2.1. The onion router

Initially, The Onion Router (Tor) was a project started by the United States Navy to secure government communication. Since 2006, Tor has become a nonprofit with thousands of servers (called relays or relay nodes) run by volunteers across the world (Tor Project History). Tor clients anonymize their TCP application IP addresses and sessions keys, sending encrypted application traffic through a network of relays (Sarkar et al., 2020). An example client application is the Tor Browser, which allows users to browse the web anonymously.

Tor generally selects a relay path of three or more nodes and encrypts the data once for each node using temporary symmetric keys. The encrypted data hops from relay to relay, where each relay node only knows about the previous node and the next node along the path. This design makes it difficult to trace the original identity of Tor clients. Each relay removes a layer of encryption, so that by the last relay, the original data is forwarded to the intended destination as plaintext. Tor then deletes the temporary session keys used for encryption at each node, so that any subsequently compromised nodes cannot decrypt old traffic (Dingledine et al., 2004).

2.2. Virtual private networks

Virtual Private Networks (VPN) are used to ensure communication privacy for individuals or enterprises, and can serve to separate private address spaces from the public Internet. VPN software disguises client IP addresses by tunneling encrypted communications through a trusted server, which acts as a gateway or proxy to route client traffic to the broader network space. Client data is anonymized behind VPN server credentials before being forwarded to an intended destination, which may be either public or private. Any response traffic is sent back through the VPN server over the encrypted connection for the client to decrypt, ensuring anonymity between the client and recipient. Third parties, such as Internet Service Providers (ISP), will only see the VPN server as the destination of client communications. There are many forms of VPN. Some operate at the network layer, others reside at the transport or application layer (Venkateswaran, 2001).

2.3. Related work

Several researchers have considered the problem of detecting darknet traffic. However, there are limited public darknet datasets available. The CIC-Darknet2020 dataset used in the experiments reported in this paper was generated by Lashkari et al. (2020). This dataset was also used in prior research, including (Demertzis et al.; Iliadis and Kaifas 2021; Sarwar et al. 2021), and it has become a well-known darknet traffic dataset due to its accessibility. In their research, Lashkari et al. (2020) grouped Tor and VPN together as darknet traffic, while non-Tor and non-VPN were grouped as benign traffic (clearnet). They created 8×8 grayscale images from 61 select features and used Convolutional Neural Networks (CNN) to classify samples in the dataset. Their CNN model achieved an overall accuracy of 94% classifying traffic as darknet or benign and 86% accuracy classifying the application type used to generate the traffic. The application traffic was broadly labeled as browsing, chat, email, file transfer, P2P, audio streaming, video streaming, or VOIP.

The research reported in Sarwar et al. (2021) consisted of classifying traffic and application type by combining a CNN and two

other deep-learning techniques: Long Short-Term Memory (LSTM) and Gated Recurrent Units (GRU). They addressed the issue of having an imbalanced dataset by performing Synthetic Minority Oversampling Technique (SMOTE) on Tor, the minority traffic class. They used Principle Component Analysis (PCA), Decision Trees (DT), and Extreme Gradient Boosting (XGBoost) to extract 20 features before feeding the data into CNN-LSTM and CNN-GRU architectures. Their CNN layer was used to extract features from the input data, while LSTM and GRU did sequence prediction on these features. CNN-LSTM in combination with XGBoost as the feature selector produced the best F1-scores, achieving 96% classifying traffic type and 89% classifying application type.

The study (Iliadis and Kaifas, 2021) focused on just traffic type from the CIC-Darknet2020 dataset. They used *k*-Nearest Neighbors (*k*-NN), Multi-layer Perceptron (MLP), RF, DT, and Gradient-Boosting Decision Trees (GBDT) to do binary and multi-class classification. For binary classification, they grouped the data into two classes, namely, benign and darknet, similar to Lashkari et al. (2020). For the multi-class problem, they used the original four classes of traffic type (Tor, non-Tor, VPN or non-VPN). They found that RF was the most effective classifier for traffic type, yielding F1-scores of 98.7% for binary classification and 98.61% for multi-class classification.

Using the same dataset, the authors of (Demertzis et al.) further broke down the application categories into 11 classes and used Weighted Agnostic Neural Networks (WANN) to classify the data. Unlike regular ANNs, WANNs do not update neuron weights, but rather update their own network architecture piece-wise. WANNs rank different architectures by performance and complexity, forming new network layers from the highest ranked architecture. Their best WANN model achieved 92.68% accuracy on application layer classification.

The UNB-CIC Tor and non-Tor dataset, also known as ISCX-Tor2016 (Lashkari et al., 2017), was used by Sarkar et al. (2020) to classify Tor and non-Tor traffic using Deep Neural Networks (DNN). They built two models, DNN-A with 3-layers and DNN-B with 5-layers. DNN-A classified Tor from non-Tor samples with 98.81% accuracy, while DNN-B achieved 99.89% accuracy. For Tor samples, they built a 4-layer Deep Neural Network to classify eight application types. This model attained 95.6% accuracy.

In another study, Hu et al. (2020) generated their own dataset, capturing darknet traffic across eight application categories (browsing, chat, email, file transfer, P2P, audio, video and VOIP) sourced from four different darknets (Tor, I2P, ZeroNet, and Freenet). They used a 3-layer hierarchical approach for classification. The first layer classified traffic as either darknet or normal. In the second layer, samples classified correctly as darknet were then classified by their darknet source. The third layer then classified application type for each of the darknet sources. The techniques (Hu et al., 2020) used for classification include Logistic Regression (LR), RF, MLP, GBDT, Light Gradient Boosting (LightGB), XGBoost, LSTM, and DT. Their hierarchical method attained 99.42% accuracy in the first layer, 96.85% accuracy in the second layer and 92.46% accuracy in the third layer.

Table 1 provides a summary of the prior work presented in this section. We note that the research in Iliadis and Kaifas (2021), Lashkari et al. (2020), Sarwar et al. (2021) use the same dataset that we consider in this paper.

3. Methodology

The primary goal of this research is to improve upon the state-of-the-art classification of darknet traffic by exploring the performance of Support Vector Machines (SVM), Random Forest (RF), Gradient-Boosting Decision Trees (GBDT), Extreme Gradient Boosting (XGBoost), *k*-Nearest Neighbors (*k*-NN), Multilayer Perceptron (MLP), Convolutional Neural Networks (CNN), and Auxiliary Classifier Generative Adversarial Networks (AC-GAN) as classifiers. We experiment with different levels of SMOTE during a preprocessing phase, oversampling the minority classes of the CIC-Darknet2020 dataset to assess the effects of data augmentation and class balance on classifier performance. We also consider using the AC-GAN generator for data augmentation, but we find that it is ineffective for this purpose. We experiment with representations of the darknet traffic features as 2-dimensional grayscale images for CNN and AC-GAN. Then we test the robustness of our best-performing classifier in obfuscation scenarios, which serve to simulate adversarial attacks, assuming both the perspectives of an attacker and defender.

In our adversarial attacks, we apply statistical knowledge of the dataset to obfuscate specific data features, disguising one or more classes as others. We explore three scenarios whereby we either obfuscate the training data, the validation data or both. Obfuscating just the validation data simulates an attack scenario in which traffic data is disguised while our classifier is yet unaware of the attack, and thus we can only apply previously trained models without a chance to learn from the obfuscation. Obfuscating just the training data simulates a scenario in which an attacker has accessed our training data to poison it, such that we train our classifier with malformed assumptions or outright malicious supervision. A third scenario supposes we collect some of the obfuscated traffic data before training our classifier, and thus have a chance to update our classification models to detect obfuscated validation data.

3.1. Dataset

The CIC-Darknet2020 dataset (Lashkari et al., 2020) is an amalgamation of two public datasets from the University of New Brunswick. It combines the ISCXTor2016 and ISCXVPN2016 datasets, which capture real-time traffic using Wireshark and TCPdump (Gil et al., 2016; Lashkari et al., 2017). CICFlowMeter (Lashkari, 2018) is used to generate CIC-Darknet2020 dataset features from these traffic samples. Each CIC-Darknet2020 sample consists of traffic features extracted in this manner from raw traffic packet capture sessions. CIC-Darknet2020 consists of 158,659 hierarchically labeled samples. The top level traffic category labels consist of Tor, non-Tor, VPN, and non-VPN. Within these top level categories, samples are further categorized by the types of application used to generate the traffic. These type subcategories are audio-streaming, browsing, chat, email, file transfer, P2P, video-streaming, and VOIP. Table 2 details the applications that are used to generate each type of traffic at the application level.

3.2. Preprocessing

The CIC-Darknet2020 dataset has samples with missing data, more specifically, feature values of "`NaN`". We remove samples with these values in our data cleaning phase. As shown in Table 3, there are significantly less Tor samples compared to the other traffic categories. Prior work using this dataset eliminated CICFlowMeter the flow labels, namely, `Flow Id`, `Timestamp`, `Source IP` and `Destination IP`. The `Flow Id`, and `Timestamp`, which are also eliminated in our research as well. However, to obtain as much information as possible from the CIC-Darknet2020 dataset, we separate each octet of the source and destination IP addresses into their own feature columns. Preliminary tests run on the dataset with and without these IP octet features indicate an improvement in the performance of the classifiers when this IP information is retained. Thus our dataset contains 72 features total after this preprocessing step.

Table 1
Summary of previous work.

Work	Dataset	Problem considered	Techniques	Results
Demertzis et al. (2021)	CIC-Darknet2020	Only examines 11 application types	WANN	92.68% accuracy
Hu et al. (2020)	Self-generated	Hierarchial approach: Layer 1: darknet vs clearnet Layer 2: Tor, I2P, ZeroNET and FreeNET Layer 3: 8 application types	LR, RF, MLP, GBDT, LightGB, XGB, LSTM, DT	Layer 1: 99.42% accuracy Layer 2: 96.85% accuracy Layer 3: 92.46% accuracy
Iliadis and Kaifas (2021)	CIC-Darknet2020	Only examines traffic type Binary: darknet vs clearnet Multiclass: 4 traffic types	kNN, MLP RF, DT, GB	Binary: 98.7% F1-score Multiclass: 98.61% F1-score
Lashkari et al. (2020)	CIC-Darknet2020	Binary: darknet vs clearnet Multiclass: 8 application types	CNN	Binary: 94% accuracy Multiclass: 86% accuracy
Sarkar et al. (2020)	ISCXTor2016	Binary: Tor vs non-Tor Multiclass: 8 application types within Tor	DNN	Binary: 99.89% accuracy Multiclass: 95.6% accuracy
Sarwar et al. (2021)	CIC-Darknet2020	4 traffic types 8 application types	CNN-LSTM, CNN-GRU	Traffic: 96% F1-score Application: 89% F1-score

Table 2
CIC-Darknet2020 application classes (Lashkari et al., 2020).

Application class	Applications considered
Audio-Streaming	Vimeo and YouTube
Browsing	Firefox and Chrome
Chat	ICQ, AIM, Skype, Facebook and Hangouts
Email	SMTPS, POP3S and IMAPS
File Transfer	Skype and FileZilla
P2P	uTorrent and Transmission (BitTorrent)
Video-Streaming	Vimeo and YouTube
VOIP	Facebook, Skype and Hangouts

Table 3
Samples per traffic category.

Traffic Type	Samples
Non-Tor	93,357
Non-VPN	23,864
Tor	1393
VPN	22,920

Table 4
Samples per application category.

Class	Application Type	Samples
0	Audio-Streaming	18,065
1	Browsing	32,809
2	Chat	11,479
3	Email	6146
4	File Transfer	11,183
5	P2P	48,521
6	Video-Streaming	9768
7	VOIP	3567

The CIC-Darknet2020 dataset was scaled by min-max normalization, which applies the equation

$$\texttt{normalizedValue} = \frac{(\texttt{value} - \texttt{min})}{(\texttt{max} - \texttt{min})}$$

to every value in each feature column. Note that this serves to scale the feature values between 0 and 1. We also apply min-max normalization to our IP octet feature columns.

3.2.1. Data balancing

The CIC-Darknet2020 dataset does not have balanced sample counts among traffic and application classes, as shown in Tables 3 and 4. To explore the effect of reducing this imbalance on the classification task, we oversample each minority class using SMOTE. SMOTE interpolates linearly between feature values to produce new samples (Bhagat and Patil, 2015). We experiment with the following levels of oversampling: 0% (no SMOTE), 20%, 40%, 60%, 80% (partial SMOTE), and 100% (full SMOTE). SMOTE is performed on all classes with less than the oversampling threshold as compared to the class with the largest sample count. Note that 100% SMOTE results in an equal number of samples for each class, while lower thresholds of SMOTE result in an equal number of samples among only those classes which are oversampled.

3.2.2. Data representation

SVM and RF both use each the dataset samples in their original format, which is a 1-dimensional array. However, we reshape each sample to be 2-dimensional for CNN and AC-GAN. Intuitively, the data is reshaped as 9×9 grayscale images, where each of our 72 features is represented as a single pixel with the remaining pixels produced by zero padding. The pixels are ordered as their respective features appeared in the CIC-Darknet2020 dataset, starting at the top left corner of the image as shown in Fig. 3, where each row represents samples from an application class, color-coded for readability.

Both CNN and AC-GAN convolve local structures within the 2-D images, so adjacent pixels play an important role in classification. Therefore, we experiment with strategies to reorder the data to achieve better performance. We order the pixels by feature importance—as determined by our Random Forest classifier—starting at the top left corner of the image, and also reorganize the pixels spiraling outward from the center of the image. This latter strategy tends to group pixels with larger values toward the center of each image, as shown in Fig. 4.

3.2.3. Data augmentation experiment

We experimented with AC-GAN as an alternative to SMOTE, with the goal of generating realistic artificial samples that can be

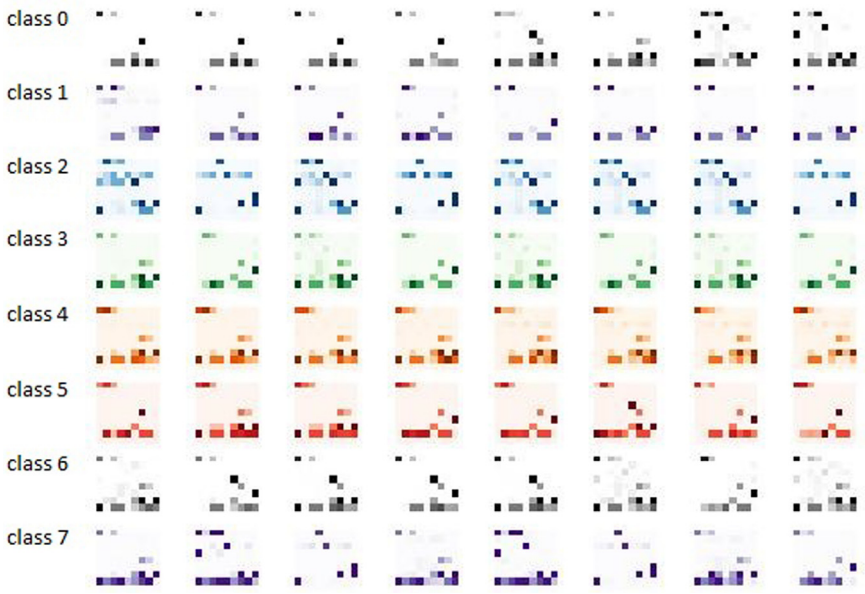

Fig. 3. Data as 2-D images in original order.

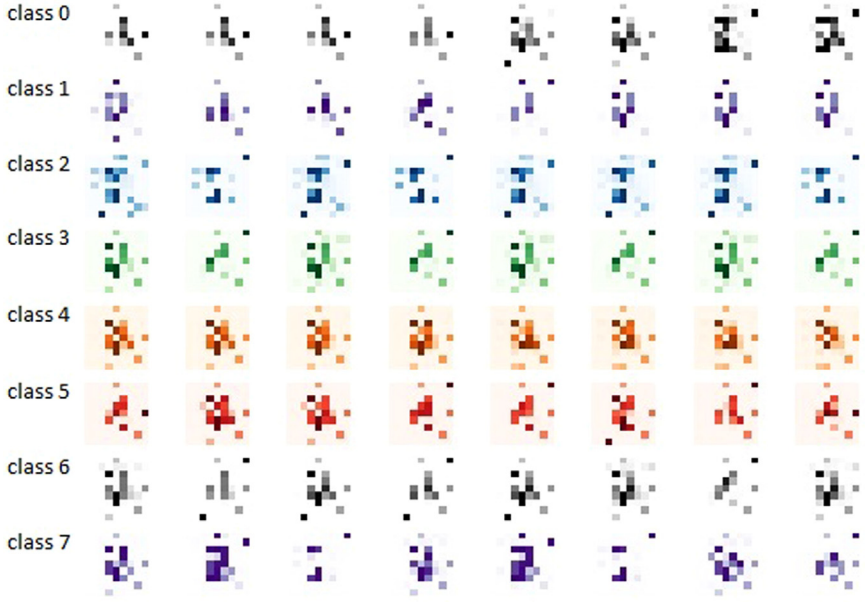

Fig. 4. Data in 2-D sorted by RF feature importance and centered.

used to augment our dataset. Again, we use data augmentation to address the issue of class imbalance. However, we abandoned this approach as we found that the fake images generated by AC-GAN are consistently detectable by a CNN model with accuracy ranging from 99% to 100%. We believe that the failure of our AC-GAN to produce realistic fake images is due to the depth of the AC-GAN neural network architecture, which is constrained by the input image size. In any case, we were unsuccessful in our attempt to use AC-GAN to augment our data.

An example of four fake samples compared to real samples can be found in Fig. 5. The fake samples in this figure may appear to be useful but, again, a CNN can distinguish the fake from the real with essentially 100% accuracy. This clearly shows that from a machine learning perspective, the fakes samples are not sufficient for data augmentation.

3.3. Evaluation metrics

In our experiments, we use accuracy and F1-score to measure the performance of each classifier. Accuracy is computed as the total number of correct predictions over the number of samples tested. The F1-score is the weighted average of precision and recall metrics, which is better for unbalanced datasets like CIC-Darknet2020. Similar to accuracy, F1-scores fall between 0 and 1, with 1 being the best possible. The F1-score is computed as

$$\texttt{F1} = 2 \times \frac{(\texttt{Precision} \times \texttt{Recall})}{(\texttt{Precision} + \texttt{Recall})}$$

Precision calculates the ratio of samples classified correctly for the positive class, while recall measures the total number of positive samples that were classified correctly. Precision and recall are com-

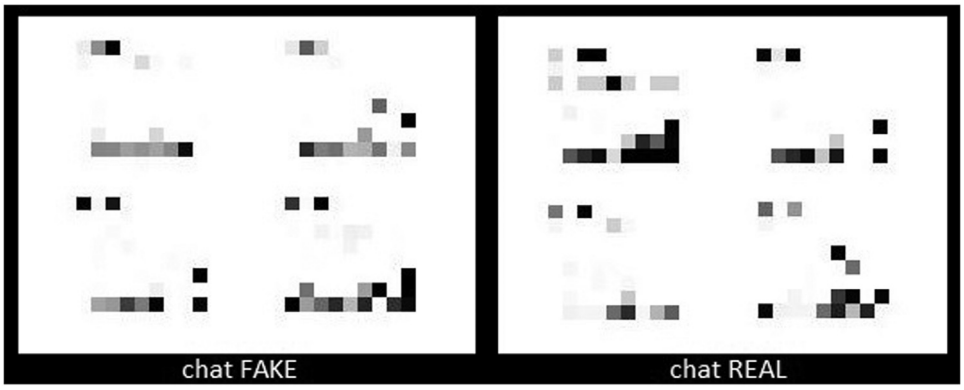

Fig. 5. Chat class (4 fake and 4 real examples).

Table 5
Computing resources used in experiments.

Computing hardware	Experiments
CPU: 8-core Intel(R) CORE(TM) i7-8550U @ 1.80GHz	SVM, RF, GBDT, XGBoost, *k*-NN, MLP
CPU: 12-core Intel(R) Xeon(R) W-10855M @ 2.80GHz	Obfuscation
GPU: NVIDIA Quadro RTX 5000	CNN, AC-GAN
	SMOTE

puted as

$$\texttt{Precision} = \frac{\texttt{True Positives}}{(\texttt{True Positives} + \texttt{False Positives})}$$

and

$$\texttt{Recall} = \frac{\texttt{True Positives}}{(\texttt{True Positives} + \texttt{False Negatives})}$$

respectively.

4. Implementation

This section details the implementation of the experiments that we mentioned in Section 3. All experiments are coded in Python. The `Imblearn` library (imblearn) is used to implement SMOTE to balance the dataset, while the package `Scikit-learn` (Scikit-learn: Machine Learning in Python) is employed to run most of the experiments, with the exceptions being that the `Tensorflow` and `Keras` libraries are utilized to implement CNN and AC-GAN. From the `Scikit-learn` library, the `metrics` module is used to evaluate the F1-scores and accuracy of the classifiers and the `StratifiedKFold` function is applied to perform 5-fold cross validation. Graphs are generated with the `Matplotlib` and `Seaborn` libraries, with the exception of the confusion matrices and bar graph, which are typeset directly in LaTeX using PGFPlots.

All experiments in this research are executed on one of two personal computers, as detailed in Table 5. We exploit a graphics processing unit (GPU) in the second computer to decrease the training time of our more computationally demanding experiments, that is those using neural networks to process 2-D image representations.

4.1. Overview of classification techniques

This section briefly describes the machine learning and deep learning concepts that we apply to classification in our experiments. These include the boosting techniques of GBDT and XGBoost, as well as *k*-NN, MLP, SVM, RF, CNN, and AC-GAN.

4.1.1. Boosting techniques

Boosting is a general technique where a collection of weak classifiers are combined to produce a stronger classifier. Gradient-Boosting Decision Trees (GBDT) assign weights to decision trees based on residuals (i.e., gradient calculations). Extreme Gradient Boosting (XGBoost) is a slightly modified—and highly efficient—implementation of the GBDT technique. XGBoost has performed well in numerous machine learning competitions (Synced, 2017).

In our GBDT experiments, we employ the log loss function, while the learning rate is $\alpha = 0.1$ and the number of estimators is 100. For our XGBoost experiments, the learning rate is selected to be $\alpha = 0.3$, the maximum depth of the trees is 6, and we employ uniform sampling.

4.1.2. *k*-Nearest Neighbors

As the name suggests, in *k*-Nearest Neighbors (*k*-NN), samples are classified based the *k* nearest samples in the training set. There is no explicit training required in *k*-NN, and hence no algorithm can be simpler, at least in terms of training. In spite of—or, perhaps, because of—its simplicity, there exist strong error bounds for *k*-NN. However, the technique is sensitive to local structure and, in particular, for small values of *k*, overfitting is common. Based on small-scale experiments, we use $k = 5$ for all *k*-NN experiments reported in this paper.

4.1.3. Multilayer perceptron

Multilayer Perceptrons (MLP) are feedforward networks that generalize basic perceptrons to allow for nonlinear decision boundaries. This is somewhat analogous to the way that nonlinear SVM generalize linear SVMs. In a sense, MLPs are the simplest useful neural networking architecture, and hence they are sometimes referred to simply as Artificial Neural Networks (ANN). In our MLP experiments, we use an architecture with 100 hidden layers, rectified linear unit (ReLu) activation functions, the Adam optimizer, and a learning rate of $\alpha = 0.0001$.

4.1.4. Support vector machines

Support Vector Machines (SVM) are supervised machine learning models frequently used for classification. An SVM attempts to find one or more hyperplanes to separate labeled training data while maximizing the margin of the decision boundaries between classes. The data must be vectorized into linear feature sets, but non-linear data can also be encoded with some success. Scaling the feature values across training samples allows coefficients of the hyperplanes (weights) to be ranked by relative importance. SVMs rely on the so-called kernel trick to map data into a higher dimensional space, which can yield nonlinear decision boundaries in the input space. The idea behind the kernel trick is that in higher dimensions, it is generally easier to find hyperplanes to separate

classes (Stamp, 2022). For our research, we perform preliminary tests to determine the best kernel for our dataset, with the result being the Gaussian radial basis function (RBF).

4.1.5. Random forest

Random Forest (RF) is an ensemble method that generalizes Decision Trees (DT). While a DT is a simple and efficient classification algorithm, it is highly sensitive to variance in the training data and hence prone to overfitting. RF compensates for these deficiencies by generating many subsets of the dataset, then randomly selecting features (with replacement) and trains a DT for each subset. This process is called bootstrapping. To classify, RF takes the majority vote from all resulting DT in a process called aggregation. Together bootstrapping and aggregation is referred to as bagging (Misra and Li, 2020; Stamp, 2022). RF also enables us to rank the importance of features based on the mean entropy within the component DTs. Feature importance tells us how influential each feature is when classifying samples with the RF. Based on small-scale experiments, we found that the default hyperparameters in `Scikit-learn` yielded the best results; see (sklearn.ensemble.RandomForestClassifier) for the details.

4.1.6. Convolutional neural networks

Convolutional Neural Networks (CNN) are a unique type of neural network that focus on local structures, making them ideal for image analysis. CNNs are composed of an image input layer, convolution and pooling layers and a fully-connected output layer that produces a vector of class scores. Convolutional and pooling layers are the fundamental components of any CNN architecture. In convolutional layers, the output of the previous layer (or the raw image in the initial convolutional layer) is convolved with randomized filters to produce local structure maps that are joined to create the output of the layer. In the convolutional process, the filter windows slide across the input image, thus emphasizing local structure, and providing a degree of translation invariance. The components of each filter are learned when training a CNN. Pooling layers decrease total training time by reducing the dimensionality of the resulting feature maps, concentrating effort on the most signifcant features (Convolutional Neural Networks for Visual Recognition; Lashkari et al. 2020). For this research, we use `max` pooling.

Our CNN architecture is based on that described in (Lashkari et al., 2020). We experiment with various hyperparameters, testing all combinations of the following in a grid search.

- Initial number of convolution filters (9, 32, 64, 81)
- Filter size (2×2, 3×3)
- Percentage dropout (0.2, 0.5)
- Number of nodes in the first dense layer (72, 256)

All these architectures yield accuracies within the range of 86% to 88% when classifying application type. Therefore, we select the architecture that produces the highest accuracy. Our select CNN architecture is illustrated in Fig. 6. Note that we use Adam for our optimizer and sparse categorical cross entropy for our loss function.

Dropout is a common technique used to combat overfitting in neural networks with fully-connected layers. However, it is found to be not as effective with convolution layers. A better regularization technique for CNN is to "cut out" sections of the input images. Such cutouts force CNN to learn from the other parts of an image during training, which tends to activate filters that would otherwise atrophy. It is comparative in effect to dropouts except that it operates on the input stage rather than the intermediate layers (DeVries and Taylor; Li et al. 2021). We implement cutouts by creating feature masks of equivalent size to our input image. We experiment with different cutout sizes including 2×2, 3×3, and 4×4 and randomize the position of the cutout within the mask. Refer to Fig. 7 for some examples of masks with 3×3 cutouts. Our cutout experiments are discussed in detail in Section 5.2, below.

4.1.7. Auxiliary-classifier generative adversarial network

Generative Adversarial Networks (GAN) are comprised of two neural network architectures—a generator and a discriminator—that compete in a zero-sum game during training. The generator takes noise from a latent space as input and produces images that feed into the discriminator. The discriminator is given both real and generated images and is tasked to classify them as either real or fake. The discriminator error is then fed back into the generator to improve its image generation. AC-GAN is an extension of this base GAN architecture, taking a class label as additional input to the generator while predicting this label as part of the discriminator output. The objective of the AC-GAN generator is to minimize the ability of the discriminator to distinguish between real and fake images and also maximize the accuracy of the discriminator when predicting the class label (Mudavathu et al., 2018; Nagaraju and Stamp, 2021). Besides using the AC-GAN generator in data augmentation experiments, we also explore the secondary class prediction output of the discriminator as a classifier.

Our AC-GAN architecture is inspired by the `ImageNet` model described in (Odena et al., 2017). However, since that architecture was built for image sizes 32×32 or larger, we modify that architecture to accommodate our 9×9 image size by reducing the number of convolutional and transposed convolutional layers in the discriminator and generator, respectively.

We fine-tune our AC-GAN hyperparameters by experimenting with the following.

- Latent space size (81, 100)
- Initial number of convolution filters (15, 40, 64, 192, 202, 384, 500, 1500)
- Number of nodes in the first dense layer (31, 81, 128, 384, 405, 768, 1000, 3000)
- Filter size (3×3, 5×5)
- Stride size (2×2, 3×3)

We observe accuracies within the range of 70% to 73% when classifying application type with these hyperparameters. The best-performing architecture with the shortest runtime duration is used in this research; Tables 6 and 7 detail our generator and discriminator architecture, respectively.

We feed training data to our AC-GAN model in batches of 64 samples. Batch normalization (BatchNorm) layers are applied between convolutional layers to regularize the training gradient step size. BatchNorm is thought to smooth local optimization steps and stabilize training, thereby accelerating convergence of GAN models (Santurkar et al., 2018).

4.2. Adversarial attacks

Our adversarial attacks rely on obfuscation, which serves to disguise application classes based on applied probability analysis. We select application classes to disguise as other classes based on minimum and maximum sum statistical distance between all class features, as specified in Algorithm 1.

We also select a third class transformation to perform based on maximal classifier confusion, whose sum statistical distance between class features is notably low, but not the minimum between classes. We ensure our class transformation can be decoded by encoding features with a deterministic algorithm, given here as Algorithm 2. We impose no additional restrictions on feature transformation.

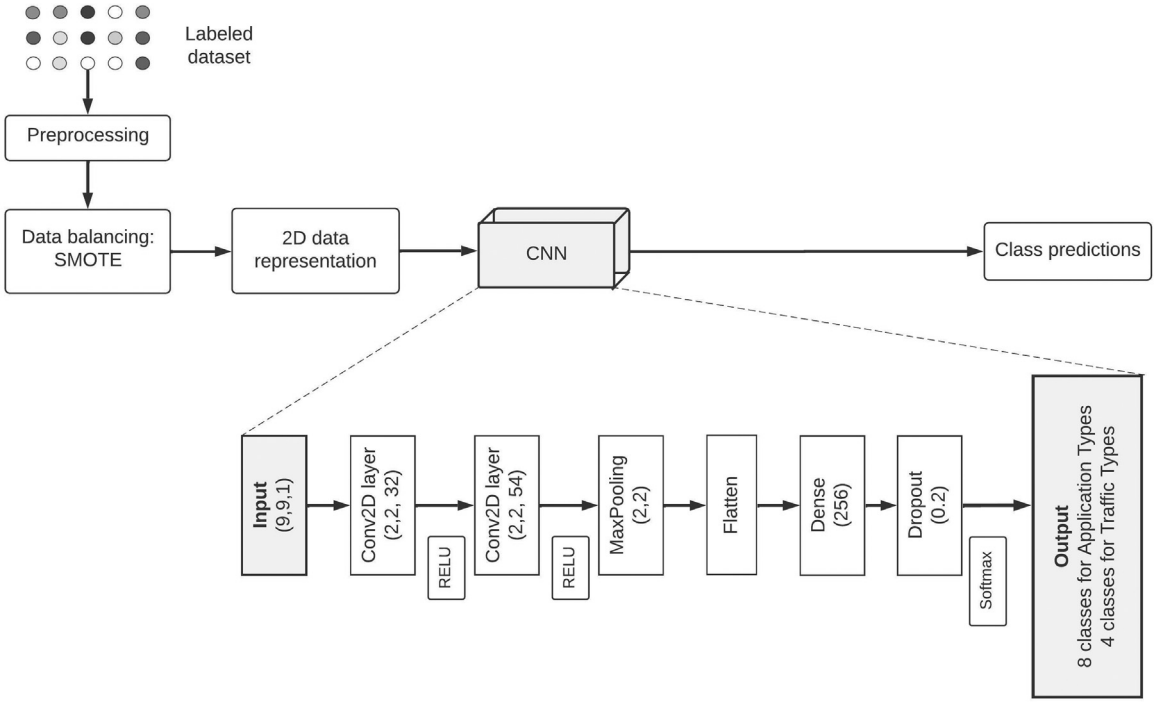

Fig. 6. CNN architecture.

Table 6
AC-GAN generator architecture.

Layer Operation	Kernel	Strides	Depth	BN	Activation
$1 \times 1 \times 100$ Input A (Latent Space)					
Dense A			405		ReLU
$1 \times 1 \times 1$ Input B (Feature Noise)					
8×32 Class Embedding for B			256		
Dense B			1		
Merge A + B			406		
Conv2DTranspose	5×5	3×3	202	✓	ReLU
Conv2DTranspose	5×5	3×3	1		Tanh

Table 7
AC-GAN discriminator architecture.

Layer Operation	Kernel	Strides	Depth	BN	Dropout	Activation
9x9x1 Input (Image)						
Conv2D	3×3	2×2	32		0.5	Leaky ReLU
Conv2D	3×3	1×1	64	✓	0.5	Leaky ReLU
Conv2D	3×3	2×2	128	✓	0.5	Leaky ReLU
Conv2D	3×3	1×1	256	✓	0.5	Leaky ReLU
Flatten						
Dense			1			Sigmoid
Dense			8			Softmax
Leaky ReLU Slope						0.2
Weight Initialization						Gaussian ($\sigma = 0.02$)
Optimizer						Adam ($\alpha = 0.0002, \beta_1 = 0.5$)

We start by generating normalized histograms of feature values per class to assess the probability at which values occur within each class. To decide which classes to obfuscate, we examine the sums of the distances between feature probability distributions from each class to each other class. We use the `cdist` function of the `scipy` Python library to calculate the Euclidean distance between probability distributions. This provides an estimate of the overall difference between classes while considering all feature probability distributions. In the case of application type, this yields the 8×8 array in Table 8, where the Class numbers correspond to those in Table 4, above.

From Table 8, we observe that class 0 is most different from class 5 and class 3 is most similar to class 7. We pick the classes with the minimum and maximum sum of statistical distances between features, changing class 0 (audiostreaming) to class 5 (P2P) and class 3 (email) to class 7 (VOIP). We also examine the confusion matrix for our best-performing classifier, RF, which is shown in Fig. 8. RF is observed to be most confused between class 2 (chat) and class 3 (email), so we decide to additionally obfuscate class 2 with class 3. We arbitrarily choose to transform lower numbered classes to higher numbered classes, e.g., disguising class 2 as class 3 instead of class 3 as class 2.

Table 8
Statistical distances between pairs of application classes.

Class	0	1	2	3	4	5	6	7
0	0	25.129	18.709	21.041	23.656	28.195.9	18.371	21.903
1	25.129	0	23.518	21.958	12.884	12.098	16.728	23.623
2	18.709	23.518	0	9.841.75	22.613	25.294	18.408	9.901
3	21.041	21.958	9.841.75	0	21.51	23.021	18.031	6.859.6
4	23.656	12.884	22.613	21.51	0	15.651	14.605	23.211
5	28.195.9	12.098	25.294	23.021	15.651	0	21.085	24.451
6	18.371	16.728	18.408	18.031	14.605	21.085	0	20.089
7	21.903	23.623	9.901	6.859.6	23.211	24.451	20.089	0

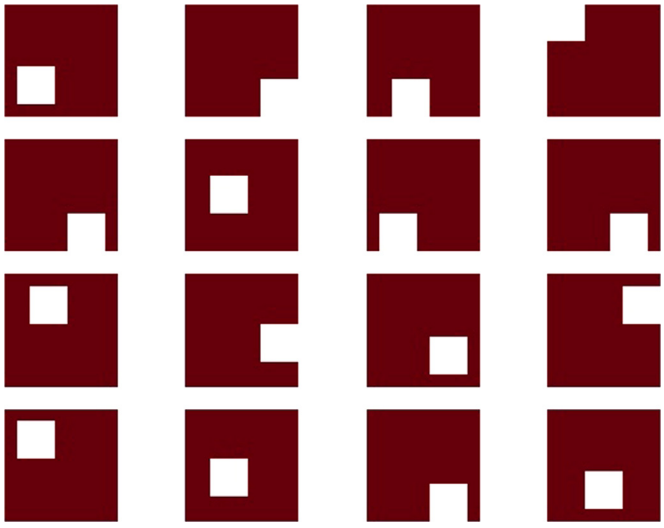

Fig. 7. Examples 9×9 images with 3×3 cutouts.

Algorithm 1 Class feature probability distributions

```
procedure COMPARE(classA,classB)
    bins = some discrete bins partitioning values 0 to 1   ▷ We use 100 bins
    A = classA feature probability distributions
    B = classB feature probability distributions
    classDistance = 0
    for each distributionA, distributionB in A, B do
        featureDistance = cdist(distributionA, distributionB)▷ Euclidean
        classDistance += featureDistance                    ▷ Manhattan sum
    end for
end procedure
```

Our obfuscation algorithm first calculates the difference in class probability distributions (DCPD) for each feature between the two classes under consideration, where the classes are denoted as A and B, and sorts each distribution from maximum to minimum. Intuitively the index of each DCPD maximum corresponds to each feature value most probably belonging to the positive class A while the minimum corresponds to each feature value most probably belonging to the negative class B. To obfuscate a sample, we then transform individual feature values by subtracting the difference in bin thresholds between the original feature bin and a target bin for obfuscation. To choose target bins for this transformation, we create a 1-to-1 map of the sorted indices of each DCPD with a reverse sort of the same DCPD. This ensures a transformed sample feature could be decoded later given the feature DCPD for a class obfuscation vector. An example visualization of the DCPD bin mapping for the transformation of the most common feature 0 values from class 2 to class 3 is provided in Section 4.2.1, below.

Reversing the 1-to-1 bin map facilitates decoding of obfuscated class sample feature values back to their original values. To do this we add back the same difference in bin thresholds which we subtracted earlier, thus applying each feature DCPD between known classes as a decoder key to undo an expected class obfuscation for a particular feature. To test this method of class obfuscation, we performed the three adversarial attacks summarized in Table 9, with RF as the classifier.

Algorithm 2 Disguise one class sample as another class sample

```
procedure OBFUSCATE(sample,classA,classB)   ▷ To decode, reverse A and B
    bins = some discrete bins partitioning values 0 to 1   ▷
We use 100 bins
    A = classA feature probability distributions
    B = classB feature probability distributions
    for each featureValue at featureIndex in the sample do
        featureBin = the bin which contains featureValue
        DCPD = A[featureIndex] - B[featureIndex]
        AtoB = sorted DCPD from maximum to minimum
        BtoA = sorted DCPD from minimum to maximum
        oldBin = where AtoB[oldBin] == featureValue   ▷
Red arrows in Figure 9
        newBin = BtoA[oldBin]                  ▷ Black arrows in Figure 9
        newValue = featureValue - (bins[oldBin] - bins[newBin])
        sample[featureIndex] = newValue
    end for
end procedure
```

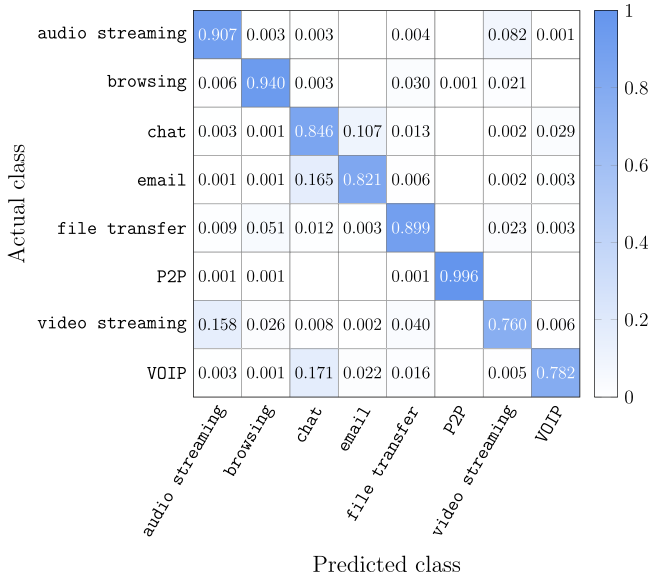

Fig. 8. Best RF results for application classification.

4.2.1. An obfuscation example

To illustrate Algorithm 2, we will walk through a simple example where we are given a sample from class 2 and we want to transform this sample to look more like class 3. Let us start with the first feature, feature 0. We note the value of this feature for class 2; call this value v. Suppose, for example, that $v = 0.178142$. We allocate 100 equal-width bins ranging from 0 to 1, so that

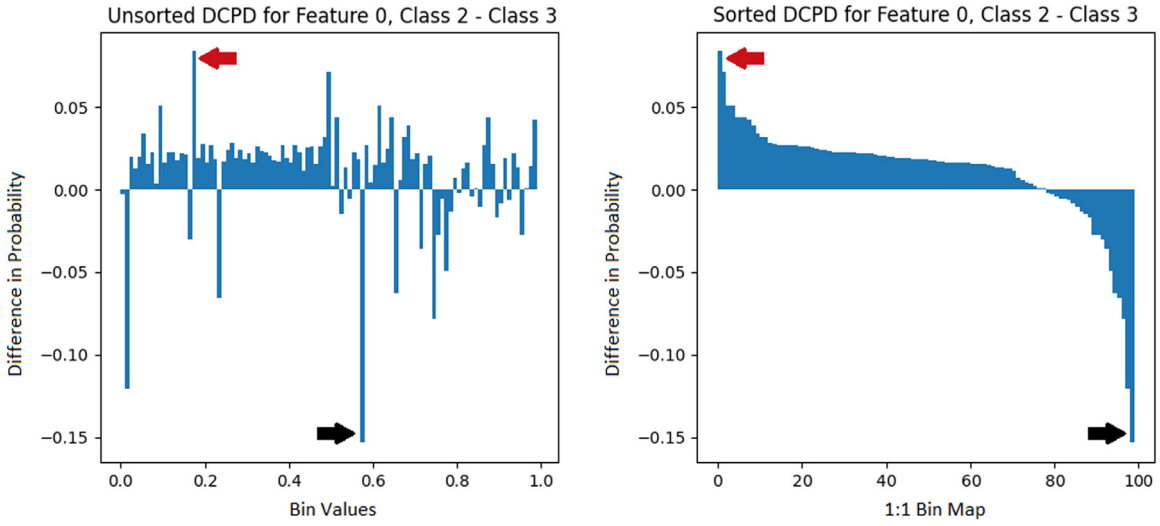

Fig. 9. Visualization of obfuscation example.

Table 9
Attack scenarios.

Scenario	What is obfuscated?		Scenario description
	Training data	Validation data	
1		✓	Simulates a novel attack where we apply an outdated model for classification
2	✓		Simulates an attack on our training data, poisoning the classifier
3	✓	✓	Simulates a novel defense where we train our model on some obfuscated data

bin b_0 corresponds to values 0.00 to 0.01 and so on. Given the value of v, we find the bin that v falls into. The value $v = 0.178142$ is in bin b_{17}, which contains values between 0.17 to 0.18. Bin b_{17} is indicated by the red arrows in Fig. 9. We then flip the sorted DCPD index at b_{17} to locate our target bin, indicated by the black arrows in Fig. 9. This target bin b_{58}, which contains values between 0.58 to 0.59. To obfuscate, we subtract the difference between b_{17} and b_{58} from v. In this example, our new transformed value is

$$\widetilde{v} = 0.178142 - (0.17 - 0.58) = 0.588142$$

which falls into the target bin b_{58}. We repeat this for all the features to transform the sample from class 2 to class 3.

Note that this obfuscation technique is designed to maximize the effectiveness of a simulated adversarial attack. Our approach ignores practical limitations on the ability of attackers to modify the statistics of the data. Hence these simulated attacks can be considered worst-case scenarios, from the perspective of detecting darknet traffic under adversarial attack.

5. Results and discussion

In this section, we consider a wide range of experiments. First, we determine which of the three 2-D image representation techniques discussed in Section 5.1 is most effective. Then we consider the use of cutouts, which can serve to reduce overfitting and improve accuracy in CNNs. We then turn our attention to the imbalance problem, with a series of SMOTE experiments. We conclude this section with an extensive set of experiments involving various adversarial attack scenarios.

5.1. Data representation experiments

We evaluate CNN and the AC-GAN discriminator given different 2-D pixel representations of the data features. All of our 2-D representations of the data are of size 9×9, where each pixel is a feature. The pixels in the original representation follow the order that the features appear in the CIC-Darknet2020 dataset. We hypothesize that grouping the pixels together would have a positive effect on the performance of our classifiers since convolutions operate on local structures. Our results show that CNN performs best when the pixels are sorted by RF feature importance and then grouped together at center of the image. However, this is not true for the AC-GAN discriminator. AC-GAN does better using the original data representation, contrary to our hypothesis. Table 10 shows the results for these experiments.

5.2. Cutout experiments

Initially, our CNN model is able to achieve 88% accuracy classifying application type within 15 epochs. However, we notice that overfitting starts to occur the longer we run our model. To reduce overfitting, we apply cutouts to the training data. We experiment with different cutout sizes: 2×2, 3×3, and 4×4. We observe that cutouts allow our CNN to train for a longer period of time without overfitting. The loss graphs in Fig. 10 show how the CNN model overfits after 20 epochs in the original execution but does not overfit with cutouts. There is little difference in the effects of applying 2×2 compared to 3×3 cutouts. Both delay overfitting at the same rate and the accuracies for both linger at 88%. Notably, we witness a 1% decrease in accuracy with 4×4 cutouts. As our images are only 9×9 pixels, a 4×4 cutout likely deletes too much information from the image, negatively affecting the accuracy. While cutouts address the issue of overfitting, we find that more training does not significantly improve the performance of CNN on the dataset under consideration. Thus, we do not employ cutouts in the CNN results reported below.

5.3. SMOTE Experiments

We compare the performance of our classifiers with various levels of SMOTE, performing SMOTE to oversample the training data before training each classifier for both cases, that is, traffic type and application type. The results from these experiments appear in Tables 11 and 12, respectively, where the best result for each

Table 10
2-D data representation results.

	CNN		AC-GAN	
	Accuracy	F1-scores	Accuracy	F1-score
Original	0.889	0.887	0.753	0.738
Shaped with RF feature importance	0.890	0.887	0.753	0.731
Shaped with RF feature importance and centered	0.891	0.889	0.742	0.729

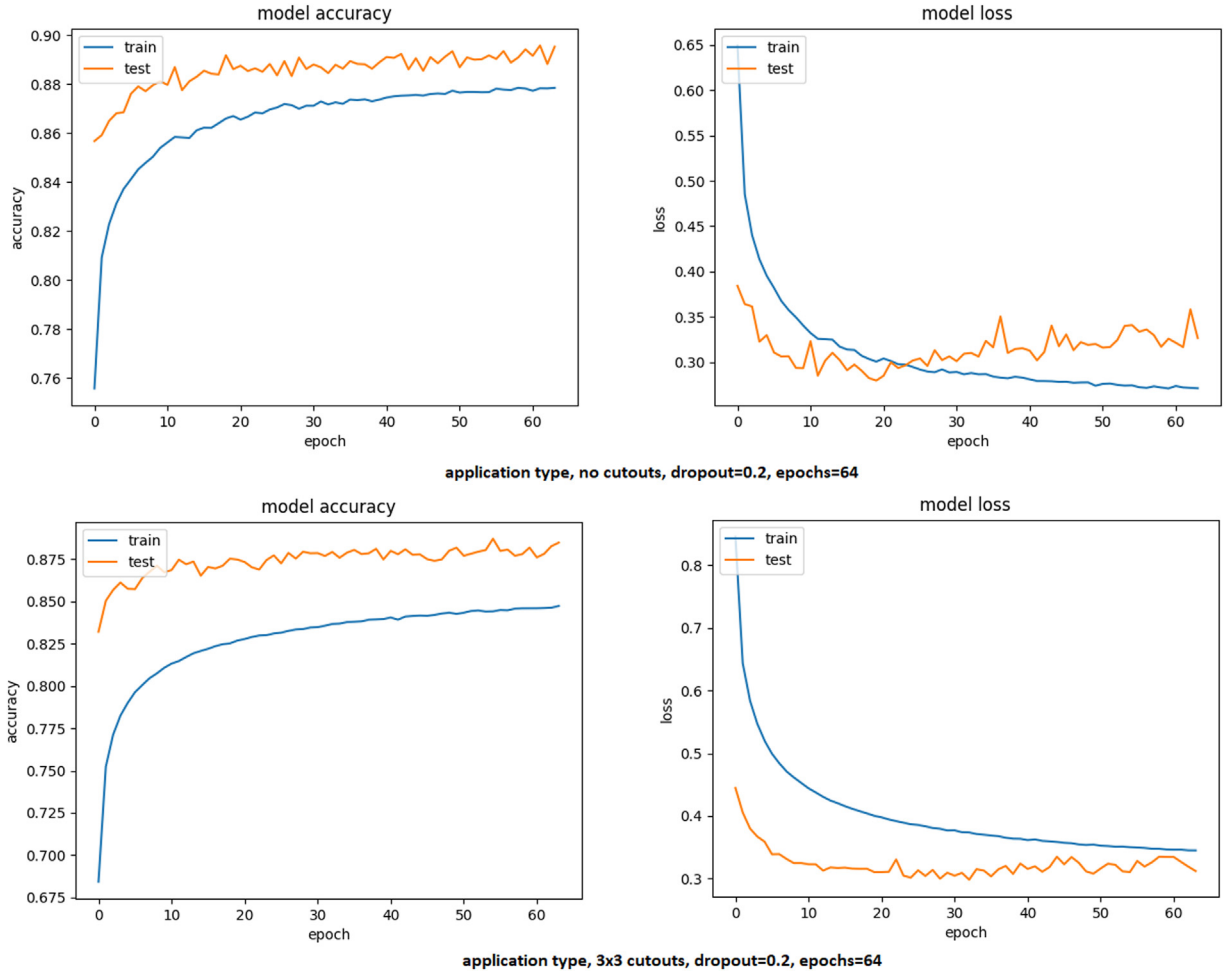

Fig. 10. The effects of cutouts on overfitting for CNN.

Table 11
Traffic classification F1-scores at various SMOTE levels.

Learning	SMOTE percentage					
technique	0%	20%	40%	60%	80%	100%
GBDT	0.961	0.961	0.960	0.960	0.958	0.958
XGBoost	0.983	0.983	0.982	0.980	0.977	0.975
k-NN	0.884	0.884	0.881	0.875	0.871	0.868
MLP	0.821	0.821	0.850	0.788	0.676	0.744
SVM	0.986	0.993	0.993	0.993	0.993	0.993
RF	0.998	0.998	0.998	0.998	0.998	0.998
CNN	0.998	0.995	0.995	0.995	0.996	0.995
AC-GAN	0.974	0.980	0.984	0.986	0.987	0.987

SMOTE level is boxed. We observe that reducing class imbalance using SMOTE does not have a large effect on the performance of most of the classifiers. With the exception of the MLP traffic classification experiments, SMOTE only affects the F1-score by about 1% to 2% in each case. Note also that the MLP results are the poorest in every case. We conclude that for the problem under consideration, SMOTE is of some value for fine tuning models.

Our RF model without SMOTE outperforms the state-of-the-art F1-scores for both traffic and application classification tasks. We observe a 1.1% improvement for traffic classification as compared to Iliadis and Kaifas (2021), where they also found RF to be their best classifier. The study (Iliadis and Kaifas, 2021) only classified traffic type, thus no application type performance is available for comparison. For application classification, our RF model achieved a 3.2% increase over (Sarwar et al., 2021). In addition, our CNN model outperformed the CNN results in Lashkari et al. (2020) by 2.8% and is within 0.2% of the more complex and costly CNN-LSTM results in Sarwar et al. (2021). We are only able to compare classification results for application type with (Lashkari et al., 2020) because they approach traffic type classification as a binary problem while we address it as a multiclass problem. Table 13 summarizes the best performance of our classifiers in comparison to relevant prior work, where the best results in the Traffic and Application columns are boxed. Overall, RF is our

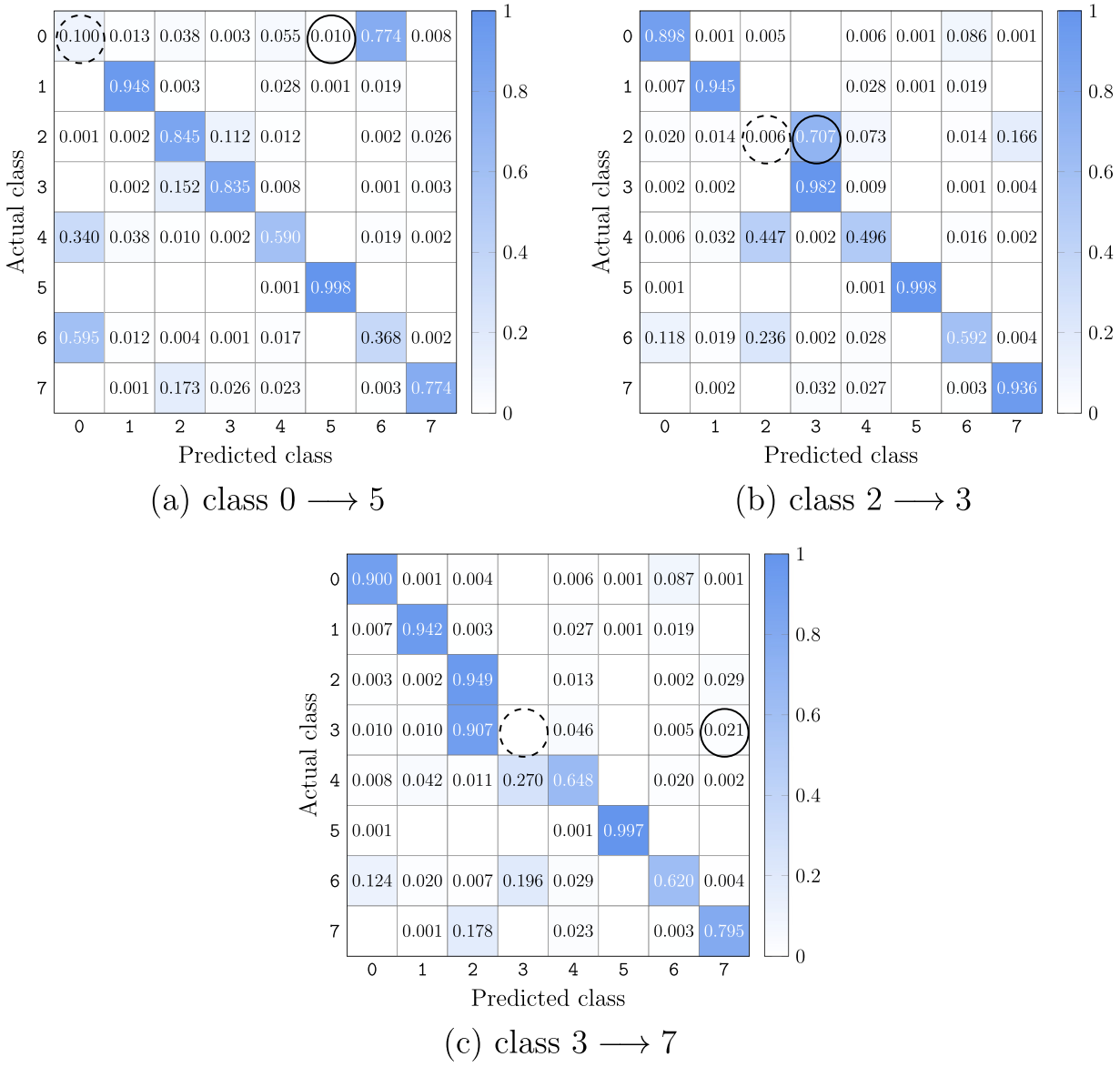

Fig. 11. Confusion matrices for attack scenario 1.

Table 12
Application classification F1-scores at various SMOTE levels.

Learning technique	SMOTE percentage					
	0%	20%	40%	60%	80%	100%
GBDT	0.840	0.840	0.840	0.838	0.837	0.835
XGBoost	0.893	0.890	0.888	0.887	0.885	0.885
k-NN	0.750	0.746	0.742	0.736	0.734	0.734
MLP	0.591	0.587	0.596	0.558	0.547	0.536
SVM	0.834	0.839	0.842	0.846	0.847	0.848
RF	0.922	0.920	0.921	0.921	0.920	0.920
CNN	0.887	0.883	0.883	0.887	0.888	0.885
AC-GAN	0.738	0.750	0.762	0.768	0.767	0.759

best-performing classifier and MLP and *k*-NN perform the worst. Also of note is the fact that the AC-GAN classifier is one of the best performing models in the traffic classification problem, but it performs relatively poorly in the application classification task.

5.4. Adversarial attack experiments

With improvement in the accuracy of darknet traffic detection by machine learning and deep learning techniques, it is realistic to anticipate that attackers will attempt to find ways to circumvent

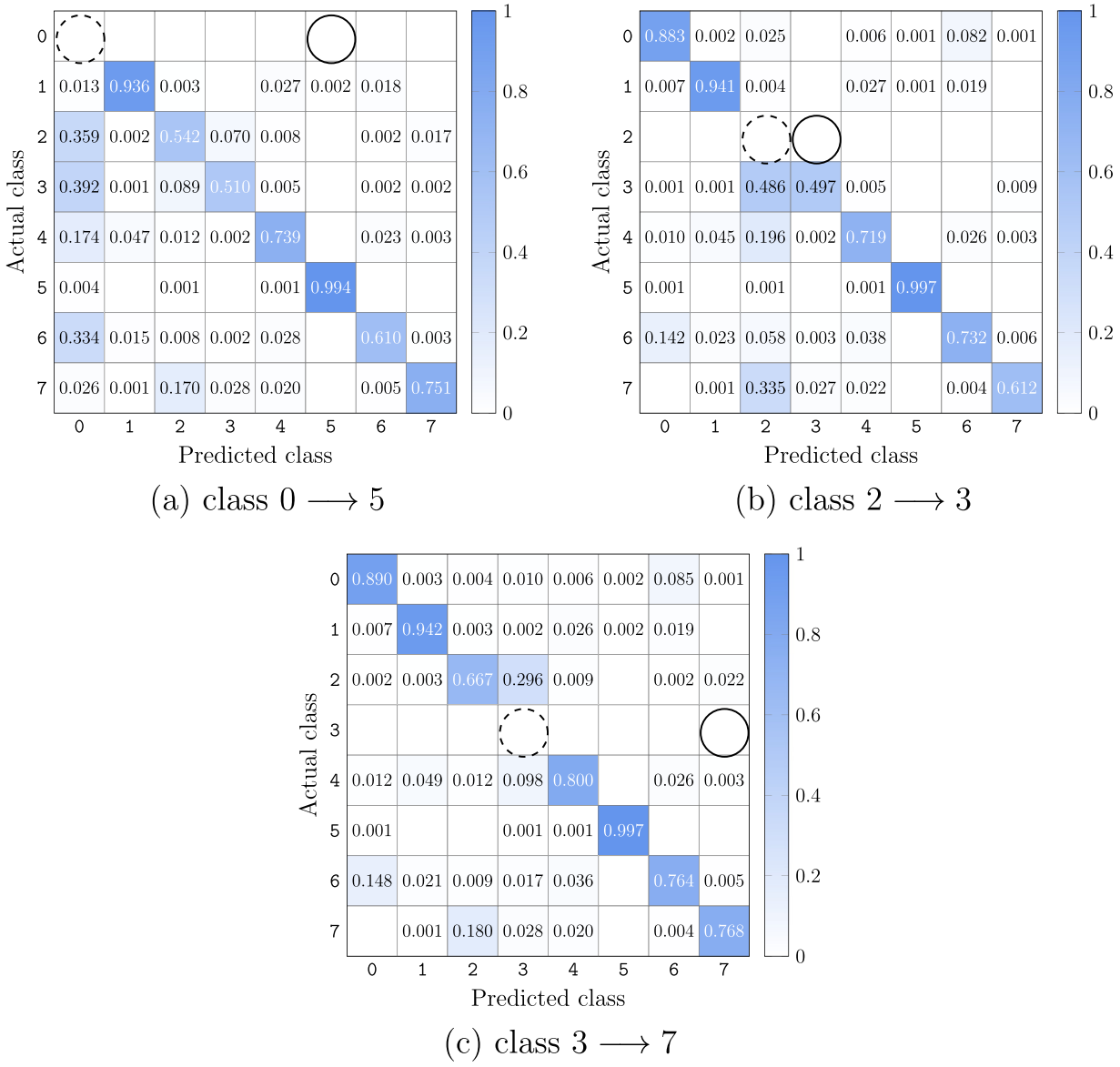

Fig. 12. Confusion matrices for attack scenario 2.

Table 13
Best F1-scores compared to prior work.

Source	Learning technique	Traffic	Application
Previous work	CNN-LSTM (Sarwar et al., 2021)	0.960	0.890
	RF (Iliadis and Kaifas, 2021)	0.987	–
	CNN (Lashkari et al., 2020)	–	0.860
Our results	GBDT	0.961	0.840
	XGBoost	0.983	0.893
	k-NN	0.875	0.750
	MLP	0.850	0.596
	SVM	0.993	0.848
	RF	0.998	0.922
	CNN	0.998	0.888
	AC-GAN	0.987	0.768

detection by modifying the profile of their application traffic. For example, someone pirating copyrighted media with P2P applications might disguise their illegal activity as VOIP traffic to avoid prosecution. We show obfuscation of traffic in this fashion can be accomplished by modifying traffic feature values, understanding that this process is most feasible and desirable at the application layer. Also, if an attacker were to discover the methods we use for classification and pollute our training data, then our classifiers could be compromised, allowing the attacker to avoid detection without modifying any of their traffic features.

For this experiment we assume the role of an attacker on the network, with the goal of modifying traffic features such that classes are incorrectly classified or entirely undetected. This could represent covert illegal activity that an attacker wishes to hinder the detection of, with common examples being P2P or file-transfer applications. Realistically, traffic features common to one application class could be modified at the application layer to appear more similar to other application classes. An attacker could do this by writing a custom overlay application to change various features, such as the number of packets sent, their communication intervals, port assignment, etc. In our experiments, we disguise class 0 as class 5 (originally the most different), class 2 as class 3 (the classes which most confused our RF classifier) and class 3 as class 7 (originally the most similar).

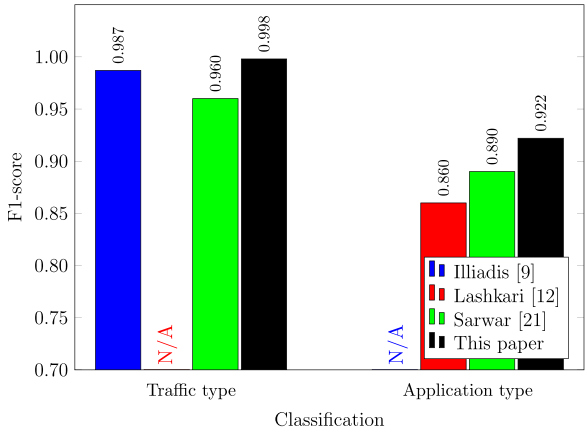

Fig. 13. Classification results compared to previous work.

In attack scenario 1, we train our RF classifier on the original application class data, then test the same model with an obfuscated class in the validation dataset. This represents a hypothetical scenario where an attacker modifies the traffic features of one class at the application layer, perhaps with an overlay application. We demonstrate that our method of obfuscation is able to defeat our best classifier in this scenario, significantly reducing detection of the obfuscated class, as well as overall classifier accuracy. Before obfuscation, RF classifies application classes with an accuracy of 92.3% without SMOTE. After obfuscation of the three class choices mentioned in the previous paragraph, the overall RF accuracy for application classification without SMOTE decreases to 80.8%, 85.4%, and 88.7%, respectively.

The confusion matrices in Fig. 11 show that RF consistently misclassifies each class we obfuscate, actually detecting no samples in the case of an obfuscated class 3, where the dashed circle indicates the class we are obfuscating and the solid circle indicates the class we intended it to appear as. However, RF did not misclassify classes 0 and 3 as the expected classes 5 and 7, respectively. Instead, the confusion matrices (a) and (c) in Fig. 11 reveal that RF mostly categorizes class 0 and 3 as class 6 and 2, respectively. It may be relevant that our obfuscation method does not account for any interdependence between traffic feature values, obfuscating each feature independently.

In attack scenario 2, we train our RF classifier with an obfuscated class in the training dataset, then test the model with the original application class data. We consider a hypothetical scenarios where an attacker entirely poisons our training data, perhaps by injecting malware into our database or by intercepting our traffic capture data stream. We find the attacker could prevent an entire class from being predicted by our best classifier when the training data for a class is entirely obfuscated. We see this trend in the all three confusion matrices in Fig. 12, where in each case, the dashed circle indicates the class we are obfuscating and the solid circle indicates the class we intended it to appear as. Notice that the entire row in the confusion matrix is zeroed out, indicating that the class was never predicted for classification by RF. Similar to attack scenario 1, the overall RF accuracy decreases to 82.0%, 86.5%, and 89.4% respectively, for application classification without SMOTE. As the obfuscated class is never considered for prediction by RF, in this scenario, we observe a lesser overall accuracy decrease as compared to attack scenario 1.

In attack scenario 3, we train our RF classifier with the same obfuscated class in both the training dataset and the validation dataset. We obfuscate only a small portion of the training data while still obfuscating all of the validation data for each of class 0, 2, and 3. We experiment with the percentage of training data we obfuscate. This represents a hypothetical scenario where the obfuscation algorithm has been obfuscating network traffic long enough to pollute a small portion of a network traffic population. A defender then updates the classifier to include this small portion of obfuscated class data at training time, with increasing exposure to the obfuscated data over time. As our dataset is split into 80% training data and 20% validation data, we decide to limit the training dataset exposure of obfuscated class data to 20% of the total training dataset. We choose to decrement this value logarithmically with three total sub-scenarios representing 0.2%, 2%, and 20% obfuscation exposure, expecting that with more exposure to the obfuscated class data, our classifier will adapt and outperform the obfuscation algorithm to correctly classify the obfuscated class in our validation dataset.

Table 14
Class accuracies for attack scenarios.

Scenario	Obfuscation	Overall Accuracy			Class Accuracy		
		(0,5)	(2,3)	(3,7)	0	2	3
No attack	–	–	–	–	0.907	0.846	0.821
1	–	0.808	0.854	0.887	0.100	0.006	0.000
2	–	0.820	0.865	0.894	0.000	0.000	0.000
3	20.0%	0.947	0.946	0.939	0.998	0.993	0.997
3	2.0%	0.935	0.939	0.891	0.958	0.921	0.120
3	0.2%	0.820	0.859	0.887	0.503	0.247	0.000

We find that 20% exposure of our obfuscation algorithm to the RF training data is sufficient for RF to predict the disguised classes with high accuracy, defeating our obfuscation technique as shown in Table 14. Note that the overall accuracies reported for attack scenario 3 are higher than our RF benchmark score of 92.2%. However, we modify the validation dataset in both attack scenarios 1 and 3, so the resulting accuracies of those scenarios cannot be directly compared to the results of prior work. Our results with lower exposure levels of 2% and 0.2% reveal a trend—of the classes tested, class 0 appears to be the most difficult for our algorithm to obfuscate, while class 3 appears to be the easiest to obfuscate. Class 2 is somewhere in between, providing a loose correlation to our metric of statistical distance between classes and the performance of our obfuscation algorithm. We observe this trend in Table 14 under Class Accuracy for attack scenario 3.

6. Conclusion and future work

In this research, we classified the CIC-Darknet2020 network traffic samples using a wide variety of classifiers. We classified based on four traffic classes and eight application classes, while fine tuning the classifier hyperparameters. We experimented with different levels of SMOTE to assess class imbalance in the dataset and explored 2-D representations of the traffic features for CNN and AC-GAN. We also approached the issue of darknet detection adversarially, from the perspective of an attacker hoping to confuse our best classifier. We demonstrated that we could effectively obfuscate application class traffic features. We then correlated the underlying statistics of the CIC-Darknet2020 dataset to the performance of this algorithm assuming specific hypothetical attack scenarios for added realism.

Among the tested machine learning classifiers, Random Forest was found to be the most proficient at classifying darknet traffic for both traffic and application types. It yielded 99.8% F1-score for traffic classification and 92.2% F1-score for application classification, outperforming the state-of-the-art studies on CIC-Darknet2020 (Iliadis and Kaifas, 2021; Sarwar et al., 2021). Figure 13 provides a visual comparison of our best results with those of prior work.

Our research was limited by the availability of darknet traffic datasets. We selected the CIC-Darknet2020 dataset because it is frequently cited and publicly accessible; however the dataset suffers from a substantial imbalance. We attempted to compensate for this class imbalance by generating artificial samples with AC-GAN and SMOTE. The artificial SMOTE samples marginally improved our classification results. Seeking to improve the quality of artificial samples, we assessed AC-GAN as a sample generator. However, our AC-GAN-generated samples were not useful for data augmentation purposes. An approach that future researchers might consider is to use clustering to group samples within a class, then train one GAN per cluster to generate samples. Other variations of GAN might also be better suited for multiclass sample generation and could conceivably generate more realistic samples.

We kept our obfuscations fairly basic, with the goal being to demonstrate that we could confuse our best classifier, with few restrictions imposed on the hypothetical attacker. Under more realistic attack scenarios, it may not be possible to so easily modify features which define darknets such as Tor and VPN, but it would be possible to obfuscate traffic features at the application layer such as those produced by CICFlowMeter analysis. We introduced a loose correlation to one statistical metric, an independent sum of distances between DCPD across all sample features. We noted that 2 out of the 3 classes we chose to obfuscate were misclassified not as the intended classes, but with a majority of predictions distributed among other classes. This results from the fact that our obfuscation metric does not account for the statistical relationship between more than two classes, nor does it account for any dependency between the CIC-Darknet2020 feature values.

There is much more remaining work that could be done to extend the adversarial obfuscation analysis presented this paper. Real traffic features could be modified on live network traffic (e.g., changing IP addresses, ports, packet lengths or intervals), or select features could be prohibited from modification during obfuscation, which is likely to be a realistic constraint. An even larger task is to explore the dependency between features in order to anticipate counterattacks. One possible avenue that future research could take with respect to the CIC-Darknet2020 dataset is to develop an obfuscation method to exploit Random Forest feature importance, or the weights of a linear SVM. This might better correlate the relationship between classifier response and dataset statistics. We only tested our obfuscation method using our best-performing classifier. It would also be interesting to explore how other classifiers respond to similar obfuscation techniques, so as to determine which classifiers are most robust to such attacks.

Author contribution

Mark Stamp proposed and guided the research, and edited the paper.

Nhien Rust-Nguyen performed the majority of the experiments, developed some of the key ideas used in this research, and wrote the first draft of the paper.

Shruti Sharma completed several of the experiments included in the paper.

Declaration of Competing Interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Mark Stamp reports financial support was provided by San Jose State University. Nhien Rust-Nguyen reports was provided by San Jose State University.

Data availability

Data will be made available on request.

References

Bhagat, R.C., Patil, S.S., 2015. Enhanced SMOTE algorithm for classification of imbalanced big-data using random forest. In: 2015 IEEE International Advance Computing Conference, pp. 403–408.

Branwen, G., Christin, N., Décary-Hétu, D., Andersen, R. M., StExo, Presidente, E., Anonymous, Lau, D., Sohhlz, Kratunov, D., Cakic, V., Buskirk, V., Whom, McKenna, M., Goode, S., 2015. Dark net market archives, 2011–2015. https://www.gwern.net/DNM-archives.

Convolutional Neural Networks for Visual Recognition, 2022. Convolutional neural networks for visual recognition. https://cs231n.github.io/convolutional-networks.

Demertzis, K., Tsiknas, K., Takezis, D., Skianis, C., Iliadis, L., 2021. Darknet traffic big-data analysis and network management for real-time automating of the malicious intent detection process by a weight agnostic neural networks framework. https://arxiv.org/abs/2102.08411.

DeVries, T., Taylor, G. W., 2017. Improved regularization of convolutional neural networks with cutout. https://arxiv.org/abs/1708.04552.

Dingledine, R., Mathewson, N., Syverson, P., 2004. Tor: the second-generation onion router. In: 13th USENIX Security Symposium (USENIX Security 04). https://www.usenix.org/conference/13th-usenix-security-symposium/tor-second-generation-onion-router.

Gil, G.D., Lashkari, A.H., Mamun, M., Ghorbani, A.A., 2016. Characterization of encrypted and VPN traffic using time-related features. In: 2nd International Conference on Information Systems Security and Privacy, pp. 407–414.

Hu, Y., Zou, F., Li, L., Yi, P., 2020. Traffic classification of user behaviors in Tor, I2P, ZeroNet, Freenet. In: 2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications, pp. 418–424.

Iliadis, L.A., Kaifas, T., 2021. Darknet traffic classification using machine learning techniques. In: 2021 10th International Conference on Modern Circuits and Systems Technologies (MOCAST), pp. 1–4.

imblearn, 2022. `imblearn` 0.0. https://pypi.org/project/imblearn/.

Lashkari, A. H., 2018. CICFlowmeter-v4.0 (formerly known as iscxflowmeter) is a network traffic bi-flow generator and analyser for anomaly detection. https://github.com/ISCX/CICFlowMeter.

Lashkari, A.H., Draper-Gil, G., Mamun, M.S.I., Ghorbani, A.A., 2017. Characterization of Tor traffic using time based features. In: 3rd International Conference on Information System Security and Privacy, pp. 253–262.

Lashkari, A.H., Kaur, G., Rahali, A., 2020. Didarknet: a contemporary approach to detect and characterize the darknet traffic using deep image learning. In: Proceedings of 10th International Conference on Communication and Network Security, pp. 1–13.

Li, J., Chang, H.-C., Stamp, M., 2021. Free-text keystroke dynamics for user authentication. https://arxiv.org/abs/2107.07009.

Misra, S., Li, H., 2020. Noninvasive fracture characterization based on the classification of sonic wave travel times. In: Misra, S., Li, H., He, J. (Eds.), Machine Learning for Subsurface Characterization. Elsevier, pp. 243–287.

Mudavathu, K.D.B., Rao, M.V.P.C.S., Ramana, K.V., 2018. Auxiliary conditional generative adversarial networks for image data set augmentation. In: 2018 3rd International Conference on Inventive Computation Technologies, pp. 263–269.

Nagaraju, R., Stamp, M., 2021. Auxiliary-classifier GAN for malware analysis.

Odena, A., Olah, C., Shlens, J., 2017. Conditional image synthesis with auxiliary classifier GANs. In: Proceedings of the 34th International Conference on Machine Learning. In: ICML, Vol. 70, pp. 2642–2651.

Santurkar, S., Tsipras, D., Ilyas, A., Madry, A., 2018. How does batch normalization help optimization? In: Proceedings of the 32nd International Conference on Neural Information Processing Systems, pp. 2488–2498.

Sarkar, D., Vinod, P., Yerima, S.Y., 2020. Detection of Tor traffic using deep learning. In: Proceedings of IEEE/ACS 17th International Conference on Computer Systems and Applications, pp. 1–8.

Sarwar, M.B., Hanif, M.K., Talib, R., Younas, M., Sarwar, M.U., 2021. Darkdetect: darknet traffic detection and categorization using modified convolution-long short-term memory. IEEE Access 9, 113705–113713.

Scikit-learn: Machine Learning in Python, 2022. `Scikit-learn`: machine learning in Python. https://scikit-learn.org/stable/index.html.

sklearn.ensemble.Random ForestClassifier, 2022. `sklearn.ensemble.RandomForestClassifier`. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html.

Stamp, M., 2022. Introduction to Machine Learning with Applications in Information Security, 2nd ed. Chapman and Hall/CRC, Boca Raton, FL.

Synced, 2017. Tree boosting with XGBoost — why does XGBoost win "every" machine learning competition?https://syncedreview.com/2017/10/22/tree-boosting-with-xgboost-why-does-xgboost-win-every-machine-learning-competition/.

Tor Project History, 2006. Tor project history. https://www.torproject.org/about/history/.

Venkateswaran, R., 2001. Virtual private networks. IEEE Potentials 20 (1), 11–15.

Nhien Rust-Nguyen received her master's in computer science in May 2022. Her research interests are in applications of machine learning and deep learning.

Shruti Sharma will received her master's in data science in December 2022. Her research interests are in applications of machine learning and deep learning.

Mark Stamp is a professor of computer science at San Jose State University. His primary research focus is on problems at the interface between information security and machine learning. He has published more than 150 research articles and textbooks in information security (Information Security: Principles and Practice, 3rd edition, Wiley, September 2021) and machine learning (Introduction to Machine Learning with Applications in Information Security, 2nd edition, Chapman and Hall/CRC, May 2022).