applied sciences

MDPI

Article

No pictures, please: Using eXplainable Artificial Intelligence to demystify CNNs for encrypted network packet classification

Ernesto Luis-Bisbé [1], Víctor Morales-Gómez [1], Daniel Perdices [1], and Jorge E. López de Vergara [1,*]

[1] Department of Electronic and Communication Technologies, School of Engineering, Universidad Autónoma de Madrid, Spain

* Correspondence: jorge.lopez_vergara@uam.es; Tel.: +34-91-497-22-59 (J.E.L.d.V.)

Featured Application: The results of this work can be applied to improve machine-learning-based network packet classification.

Abstract: Real-time traffic classification is one of the most important challenges for both Internet service providers and users, because the right traffic policing and planning allow a proper optimization of the network resources. However, there is no perfect solution for this problem, due to the grade of complexity of modern traffic. Nowadays, Convolutional Neural Networks (CNNs) are believed to be the miraculous solution for network packet classification of encrypted traffic. Nevertheless, given the obscure nature of deep learning, an appropriate explanation could not be easily obtained on how the model is detecting each traffic category. In this paper, we present an analysis on some popular CNN-based models for network packet classification, focusing on how the model works, how it was implemented, trained, and tested. By using eXplainable Artificial Intelligence (XAI), we are able to extract the most important regions of the models and extract some reasoning to justify their decisions. Moreover, in the process, we look for possible flawed methodologies that can lead to data leakage or an unrealistic performance evaluation. The results show that CNNs mainly focus on the packet length to make a decision, which is definitely a waste of resources. As we also check, the same could also be implemented with simpler machine learning models, such as decision trees. Our findings indicate that poor experimental protocols result in an unrealistic performance evaluation. Moreover, XAI techniques are of great help in the assessment of the model, showing that CNNs do not detect significant features in encrypted payloads apart from packet length.

Keywords: Convolutional Neural Networks; eXplainable Artificial Intelligence; Network Packet Classification; model evaluation; GradCAM.

Citation: Luis-Bisbé, E.; Morales-Gómez, V.; Perdices, D.; López de Vergara, J.E. No pictures, please: Using eXplainable Artificial Intelligence to demystify CNNs for encrypted network packet classification. *Appl. Sci.* **2024**, *1*, 0. https://doi.org/

Received: May 3, 2024
Revised: June 14, 2024
Accepted: June 19, 2024
Published:

1. Introduction

With the rise of the Internet usage, proper management of the network traffic in modern ISPs has become a challenge of paramount importance. Categorizing each network packet on the fly is key to proper decision-making. For many applications, such as quality of service policing or traffic shaping, routing might be more efficient if the type of data consumed is known. For example, in Internet telephony, latency is key and dropped packets are acceptable, whereas in a batch download latency is acceptable, but there must be guarantees that all information is received at least once. This packet classification cannot be done offline, i.e. after the network flow has already been completed, but rather online, i.e. while the first packets of the flow are passing in real time through the network equipment. However, as the Internet has become a fundamental part of our society, privacy issues have arisen, causing the advent of the encryption of network traffic. Encryption poses yet another challenge to traffic classification, as classical methods such as those mentioned below do not make sense any longer. Several methods have been tested for real-time network packet classification over the years: initially, the rudimentary use of port-based

classification [1] (with a basis on TCP&UDP well-known port numbers); later, the use of Deep Packet Inspection (DPI) using pattern matching and detecting protocol anomalies [2]; and most recently, algorithms based on statistical features and machine-learning-based techniques such as Support Vector Machine (SVM) [3], or deep neural networks such as Convolutional Neural Networks (CNNs) [4,5]. In particular, CNNs are at this moment a well-known solution for traffic classification, because their ability to find spatial correlations between earlier and later parts of the data is useful in network traces, where the data might be correlated spatially to a high degree, due to the relation between contiguous bytes. In fact, CNNs are becoming increasingly relevant, due to their reported high accuracy values, but they have a major drawback: they generally behave as black boxes, because the statistical features detected by the neural network are hard to be explained outside the context of the predictor itself. Thus, we decided to study in depth the capabilities of CNNs using a public network traffic dataset, focusing on whether it is feasible to extract usable data from the model. By analyzing state-of-the-art models and their methodologies for training and testing, we are able to identify common practices, keys to success, and possible malpractices, such as data leakage.

Compared to previous works, our methodology includes the following main contributions: firstly, we need to make sure that the separation between packets in the train and validation sets prevents data leakage. This is, we ensure we do not train and test with different packets of the same flows, in order to prevent overfitting and avoid bias in the model. While this is an obvious practice in AI, it is not as simple as it sounds when doing network packet processing, because packets of the same connection can cause data leakage if they are incorrectly split. Secondly, we want to gain a deeper understanding of what CNNs are actually watching. The lack of explainability or interpretability is a major concern for their potential feasibility and deployment in real-world scenarios. We expect to hold the model to the same standards of privacy, unbiasedness, and reliability as other standards in the industry. Once we have the right information, we have built a reference CNN model for network traffic classification. From the aforementioned dataset, we have filtered the headers of the packets in order to manipulate just encrypted payload data and avoid using IP addresses and ports. After training, we analyzed the performance and the explainability. Regarding explainability, we must study how to extract data directly from a deep neural network. For this, we have applied techniques used in other research fields, such as image and video classification, to create a useful representation of the reasoning behind the classification of network packets. To make our methodology applicable to all contexts, we used an algorithm that is universally applicable to all CNNs in order to further understand them, and developed several modifications in the model in order to see how the behavior may change depending on the added or removed features.

As a result, we focus on understanding important factors for model performance. First, we analyze characteristics of the data that might be relevant. A good example of this is that data leakage leads to overfitting when packets of the same flow are both in training and test set. We also analyze the different parameters of the model that impact performance. Finally, the decision is analyzed using eXplainable Artificial Intelligence (XAI) techniques, so that we try to understand how the features are built and what they focus on.

The rest of the paper is structured as follows. Next, section 2 shows the related work, in order to contextualize our research. Then, section 3 explains how available datasets have been processed, trying to avoid the bias that is present in many prior network classification works. In section 4, we present the different components of a CNN architecture to classify network packets. After that, section 5 presents gradient analysis as a way to explain the convolutional models. In this way, section 6 applies this technique to real data, so we can understand what the neural network has generalized from the encrypted traffic it has trained with. Finally, section 7 concludes the paper, providing a summary of the main results and contributions.

2. Related work

The problem of network traffic classification has been extensively studied, and many works, including books, articles, and deep research, have already approached this matter. As such, this section covers the papers of interest for our research. First, we present state-of-the-art works that are relevant in the scope of this study. Second, we analyze an extensive set of CNN-based models to deepen our knowledge on how CNN models are applied in network packet classification.

2.1. State of the art

Machine learning techniques have been progressing and evolving since the mid-nineties, but the first proposals to use modern machine-learning techniques for network traffic classification became popular around the 2000s. A relevant example is SVMs, appearing around 2002 [6].

Nowadays, XAI has gained a pivotal role in the use of modern artificial intelligence (AI) in the research community [7], because the difficulties to interpret deep learning models prevents their use in real scenarios, as the need for accountability, responsibility and transparency of the decisions arises. Additionally, the unknown properties and behavior of the model make further improvements much harder to grasp, as we do not have proficient metrics that help us define better hyperparameters.

Different authors have applied a wide variety of machine learning techniques to network traffic classification. Different datasets, models, applications and methodologies have been used, such as Multilayer Perceptron, CNN, Recurrent Neural Networks (RNNs), Autoencoders, and other types of layers [8–11]. Currently, deep learning has established itself as one of the main tools in the area of network traffic classification. However, few studies have addressed how to classify the traffic in real-world situations. The research made by [12] must be taken as a warning sign: we must develop stronger tools to further prove and test the models, as most of them heavily underperform on real scenarios due to a lack of generalization and a misjudgment in the true capabilities of the model. Additionally, these techniques are sometimes applied to non-encrypted packets, which is not representative of the nowadays networks. Our work focuses only on encrypted TLS payloads, so neither the header nor the data format can leak information about the corresponding label. This will be discussed extensively in section 3.

Authors in [4], one of the first references on using deep learning for traffic classification, show how to detect malicious traffic using CNNs. They process the network packets to display them as 2D images, and it is observed that, depending on the type of application, they can see specific patterns. This is promising because it means that, even if the traffic is encrypted, there might be a way to classify it according to the application.

The works in [13,14] show that such techniques used in other fields, such as image or video classification, can be applied in network and service management, bringing a new perspective into some challenges in the area, and opening up the entire field for a different point of view. Similarly, we have applied and extended XAI techniques, reaffirming their strengths and addressing the state-of-the-art challenge of understanding traffic classification.

Previous studies have demonstrated the ability of neural networks to achieve high levels of accuracy in the classification of encrypted and non-encrypted traffic. However, a significant challenge remains in the application of such techniques in real-world environments where low latency and high processing rates are required. In such scenarios, waiting for the arrival of a sufficient number of packets to classify a flow is not feasible and classification must be performed on individual packets in real-time.

To address this challenge, recent literature has proposed a method whereby network traffic packets are processed and displayed as 2D grayscale images, which can reveal discernible patterns depending on the type of application. This approach is particularly noteworthy as it enables classification of encrypted traffic based on the source application. Consequently, it presents an opportunity for Internet Service Providers (ISPs) to enforce

service differentiation policies while ensuring the protection of user data. In order to harness these unique patterns, CNNs are employed as an image classification model. CNNs have been extensively investigated in recent years for their ability to effectively solve real-world image classification problems.

One advantage of using neural networks accelerated by GPU (Graphics Processing Unit) or Deep Learning Processor Units is the parallelization of the classification process. By working with individual packets, in which no information needs to be saved between flows, it is possible to create pipelines of work that greatly accelerate the process. However, there is a challenge that must be addressed. To enable these GPUs to operate at the highest performance, batches of images ranging from 64 to 256, depending on the architecture used, are required [15]. This has the disadvantage that one must wait to receive the next packets to classify when their information is not necessary in the classification process.

If real-time classification is desired, latency is to be minimized. In that case, Deep Learning Processing units on FPGAs or other converged accelerators are required. Otherwise, image batches would need to be reduced, leading to compromised classification system bandwidth. Therefore, these accelerators are essential for real-time network packet classification. With this approach, individual packets can be classified in real-time without extra latency due to the batching effect or compromising system bandwidth, leading to significant improvements in classification performance [16].

In contrast to the aforementioned challenges in the field of traffic classification using CNNs, we note that authors in [17] noticed how flow characteristics might affect their network classifier. Specifically, their method utilizes both flow information to contextualize the network traffic. By incorporating such information, they are able to leverage the power of temporal data and overcome the limitations of traditional packet classification approaches. Overall, their approach represents a significant advancement in the field, offering promising possibilities for improving the accuracy and reliability of network traffic classification.

2.2. Comparative analysis

To comprehend the significance of traffic classification using convolutional networks, a thorough review of 20 articles published between 2017 and 2022 was conducted. Table 1 provides a summary of this state-of-the-art analysis, comparing our work with previous research.

The literature review is focused on several key aspects of network traffic classification, including the classification of services or applications and if the traffic was encrypted, the dataset analyzed, the network architecture utilized, and whether packet processing was conducted on the fly or by waiting to generate flow statistics. Additionally, the review assessed how protocol headers were handled and how packets that did not meet the minimum size of the neural network input were padded. The analysis also examined whether the reviewed articles aimed to reduce bias by class balancing and manage data leakage by separating training streams from testing streams.

In the comparative analysis, we have studied the authors' handling of bias and data leakage. We observed that several studies report high accuracy performance on testing datasets, with some achieving as high as 99%. However, a significant proportion of these studies lack a detailed description of the experimental protocol and dataset handling process, raising concerns regarding possible flow bias or data leakage. Except for us and the work in [21], the rest of studies cited in Table 1 either do not specify how train-test split was done ('?'), or give hints that data leakage might have happened ('×'). Additionally, in many cases, it is not clear how the packet-to-image conversion is performed, with no information provided on whether headers are eliminated or if images are padded to obtain the desired dimensions. Moreover, we note that some studies suffer from imbalanced classes or omit this information altogether. Finally, the use of flow statistics instead of independent packets presents a challenge for real-time traffic classification in actual networks, as on-the-fly processing is required.

Table 1. Review of recent publications on traffic classification.

Paper	Year	Architecture	Dataset	Output	Online	No flow data leakage
Ours.	2024	1D-CNN, 2D-CNN	ISCX VPN	Services	✓	✓
[18]	2023	CIL CNN	MIRAGE19	Mobile apps	×	?
[19]	2022	CNN	ISCX nonVPN	Services	×	×
[20]	2022	Decision Tree	ISCX VPN & nonVPN	Heartbleed attacks	✓	?
[21]	2022	1D-CNN	MIRAGE COVID CCMA 2022	Mobile apps	×	✓
[22]	2022	1D-CNN, DBN	ISCX VPN & nonVPN	VPN vs nonVPN & Services	✓	×
[23]	2022	FDCNN	ISCX VPN & nonVPN	VPN vs nonVPN & Services	✓	×
[12]	2021	2D-CNN	ISCX nonVPN	Applications	✓	?
[17]	2021	2D-CNN	ISCX VPN & nonVPN	Both	×	?
[24]	2021	GADCN	USTC TFC2016	Applications	×	?
[10]	2020	1D-CNN+tSNE	Own	Services	×	?
[25]	2020	SAE & 1D-CNN	ISCX VPN & nonVPN	Applications	✓	?
[26]	2020	CNN, LSTM, SAE, HAN	WIDE, USTC	Applications	×	?
[27]	2019	1D-CNN	ISCX VPN & nonVPN, ISCX 2012 IDS	Services & Malware	×	?
[28]	2019	1D-CNN, 2D-CNN, 3D-CNN	ISCX VPN & nonVPN	Applications	✓	×
[29]	2017	CNN	Own (RedIRIS)	Services	×	?
[30]	2017	PNN	NOCSET, MOORESET	Applications	×	?
[31]	2017	2D-CNN	USTC TFC2016	Malware	✓	?
[32]	2017	1D-CNN	ISCX VPN & nonVPN	VPN vs nonVPN & Services	✓	?
[33]	2016	k-NN	Own (VPN)	Applications	×	?

In the articles we have reviewed, a wide variety of open datasets has been utilized. ISCX VPN-nonVPN [33] is the main one for traffic classification, used by [17,19–23,25,27,28,31]. Other ones are ISCX 2012 IDS [34], USTC-TFC2016 [31], MIRAGE19 [18], and MIRAGE-COVID-CCMA-2022 [21]. However, some examples of lab-created traffic and private datasets captured in different regions have been observed [10,26,29,30]. The aforementioned datasets contain network captures in the form of `pcap` files, where the label of the transmitted application or service is indicated in the title. Among these datasets, ICSX VPN-nonVPN [33] is consistently the most utilized, as it is the largest and has a wide variety of services and applications. It has been selected as a benchmark dataset in articles aiming to provide comparisons between their results and the rest of the research community.

Most of the time the convolutional network architecture was complemented by another type of ML technique or DL methodology, some examples we have identified are tSNE (t-distributed Stochastic Neighbor Embedding) [10], k-NN (K-Nearest Neighbor) [33], SAE (Sparse Autoencoder) [25], RNN (Recurrent Neural Network) [26], GADCN (Generative Adversarial Deep Convolutional Network) [24], FDCNN (Frequency Domain CNN) [23], CIL (Class Incremental Learning) [18], Decision Tree [20], DBN (Deep Belief Network) [22] and Bayesian fusion [17].

Some articles, such as [17–19], studied utilize convolutional neural networks to extract information from statistics generated with various packets of the same flow.In some cases, these statistics are used to generate images, while in other cases, classic machine learning algorithms are utilized for classifying, which are excluded in the study conducted here, due to fundamental differences with the explainability of CNN classification.

In other studies on traffic classification with machine learning, it has proven to be an effective way of classification, but it also has several disadvantages. The first challenge is the difficulty of classifying the first packets of the session, which cannot be classified due to the lack of statistics. In some studies, this is resolved by waiting for the first five packets to

generate the necessary inputs. Another problem introduced is the increase in packet latency due to classification, as the classifier must wait for the arrival of the first packets. Finally, computational and memory costs are added because real-time data structures that allow indexing of aggregated packet data must be generated, which can be a real challenge in a network at 10 Gbit/s or higher rates. To solve these problems in a real high-rate network, classification systems that work on the fly must be sought. Mainly, packet information should be used without seeking correlation with the information of its network flow. One way to do this is to convert the packet into an image and process it with the convolutional network, as for example in [31,32].

3. Initial data processing: avoiding model bias

The very first task we have to solve when working with AI is data pre-processing. Our main goal is to extract features adapted to network packet classification and build models with useful and standardized information. It is also essential to manage the bias in our data, so that the predictions are not only realistic but also accurate and fair. This section details how data pre-processing was handled, the processes, decisions and developments that have been made to ensure, to the best of our ability, that the model is capable of classifying traffic in a real network, avoiding overfitting or model bias.

The dataset used in this paper is the experimentation dataset ISCXVPN2016 [33], composed of 25 GBytes of captured data traffic divided in 152 capture files. It is grouped in 42 labels, depending on the application. For our study, we will group these labels into general categories representing traffic classes [35], such as Video (Vimeo, Facebook Video, and YouTube), Chat (Facebook chat, Skype or ICQ), Web E-mail (GMail), File Transfer (FTP) or Audio (Skype). We take this decision because there is no need to look at the Over-The-Top (OTT) service provider to apply fair QoS policies.

Being a multiclass classification problem, choosing always the same class in a balanced-classes situation would yield an average accuracy of $\frac{1}{\#\text{ of classes}}$. According to the current literature, some systems reach values close to 90% in class averages [25,27,36]. However, one of the main problems detected is that these papers do not make clear the experimental protocol, e.g. how they split the training and test packets. In the case of network traffic, it is especially important to split them correctly, to avoid undesired biases. For example, performing a temporal split on a traffic capture of 100 packets would mean that we take the first 70 for training and the last 30 for testing. In that case, we might be training the model on recognizing the start of the connection phase but not the closing part. Thus, the model will exhibit strong biases in a real environment. Then, will this problem be solved if the training and test data are taken randomly? Not necessarily. By using random packets, you are probably using packets from the same flows or sessions in the training and test, causing data leakage. If part of the test information leaks into the training set, the model does not learn to generalize but to detect those related packets by identifiers such as IP addresses, port numbers, or domain names. This is not a realistic case, as it encourages the neural network to learn particular literals of our dataset instead of general conditions. Even if IP addresses and ports are discarded, there might be information in the data that identifies the flow, such as session identifiers or encryption algorithms and keys.

In order to give the full context on how to avoid these issues, an overview of the whole process will be made from the time the traffic trace is obtained until it is classified by the neural network. The first thing we will do is to filter from each traffic capture those packets that do not contain payload of the application, for example ACKs, DNS and other protocols. This is done to prevent the neural network to train with packets that can be classified by well-known port in a real application or with DPI techniques. Then, we must extract from each traffic capture the individual packets and convert them to binary files. In this step, we will also discard Ethernet, IP, and TCP headers, since they do not contain payload from the application, and we want to avoid training with data that depend on the network conditions of the capture. We can take advantage of this process to group the packets by flows by using the quintuple (source IP, destination IP, source port,

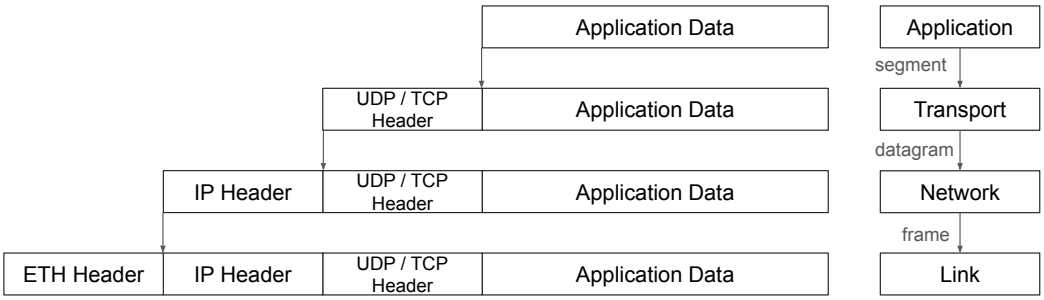


Figure 1. Protocol headers

destination port, transport protocol). This does not mean that statistics will be extracted, simply that the packets of the same flow will be labeled. This method was chosen to ensure the independence of traffic flows and to separate the data into training and testing sets, avoiding data leakage. This approach leverages the similarity among flows of the same application (see, for instance, Figure 10). Thus, the objective is to train and test with packets of different flows in order to avoid overfitting and data leakage. Next, these individual packets must be converted into images so that they can be used to train the neural network. Each byte of payload information will be converted to a pixel, and then the pixels will be arranged according to the desired dimension.

The final step is to generate a convolutional neural network that will process these images, classify them and extract the most probable application value. With this label, we could use different QoS policies to improve the experience in the network. With this process, we have achieved that from each packet an application label is extracted individually, without the need to load in memory the rest of the flow data. It is important to work with individual packets and not with network flows to gain speed and reduce latency, because, in a high-speed system, grouping data extracted from a traffic flow can be time-consuming, and it would affect the performance of a latency-critical network application such as VoIP, video calls or live gaming.

3.1. From packets to images

As explained above, we start with the processing of the protocol headers. As shown in Figure 1, a typical network packet includes ETH, IP, TCP, and UDP headers containing information specific to the network conditions at the time of packet capture. In our model, we propose discarding these headers to isolate the payload, thereby retaining only the "Application Data" bytes. This task is performed using a data preprocessing step done using Tshark, available in the GitHub repository link attached to the paper 1. The process is summarized in the paper, but it is a simple filter. These headers should be discarded for several reasons. The first one is that, due to the number of servers in a cloud platform, they do not necessarily provide reliable information on the user's application data (e.g. different Google services share the same IP addresses) [37]. Therefore, a traffic classification would not be correctly made according to the applications. The second reason is that they can easily generate false positives in the neural network model prediction, since it could learn to differentiate an application by using a specific IP address, a specific port or even unrelated TCP parameters such as the window size. Additionally, this is not necessary in the neural network model, as a known port system would more efficiently leverage port information and avoid over-fitting. In addition, other TCP parameters such as the ACK or SYN sequence numbers would not necessarily be used to classify traffic. Once the packet headers have been filtered, the payload binary data has to be transformed into an image or another data structure that accepts a convolutional neural network as input. To transform the data into images, we use each byte of information in the packet as a pixel and represent that value as a grayscale value, i.e. 0 is a black pixel and 255 is a white pixel. This process will help to normalize the dataset and could help in detecting patterns in the image representation of the packets.

However, images tend to have fixed sizes, whereas network packets can vary in size: from minimal sizes of 40 bytes to the maximum of the MTU (usually 1500 bytes), even for each class (e-mail, chat, video, etc.). Thus, we need all sequences and images to have a fixed size because CNN libraries usually need a fixed input size. When a packet does not reach the size to be transformed as an image, missing data should be filled in. Two solutions are proposed: the first would be to fill in the rest of the packet with zeros (zero padding); the second would be to fill in with random values (random padding) [12]. We have used the first alternative, as the intention of the model is to alter the packets only what is needed to be able to be fed into the model, based on input perturbation techniques. Given the different packet sizes, filling with random values would introduce an undesirable noise in the model. In fact, we confirmed that this random padding produced worse performance than zero padding. Packets without payload data (e.g. TCP SYN or ACK segments) are taken out of the dataset, as they would be pitch-black images. This packet-to-image process can be performed for grayscale 1-dimensional (1D) or 2-dimensional (2D) structures [38], although in other studies, so-called 3-dimensional (3D) architectures have also been considered [28]. By 3D images, they mean images with RGB color gamut, not grayscale. This choice of the number of dimensions may affect on the performance, both in terms of accuracy of the model and in terms of throughput of the system. It is expected that a neural network can take advantage of image acceleration hardware architectures to process in parallel the 2D and 3D images. Moreover, in these cases it must be considered that an artificial dimension is created in the packets, where the information is originally sequential, and a row of pixels does not have to be related to the one immediately above or below it, as it will be discussed in section 5. In the work by Zhang et al. [28], they presented a comparative analysis of 1D, 2D and 3D convolutional neural networks (CNNs) for traffic packet classification. The results showed that 1D CNN achieves the highest precision and F1-score, which is consistent with previous findings. Interestingly, 3D CNN closely matches the performance of 1D CNN and even surpasses that of 2D CNN, despite the potential increase in complexity and spatial information introduced by the 3D images. Furthermore, that paper provides insight into the resource usage, speed, and bandwidth of the three cases. It is observed that, as expected, the addition of dimensions to the convolutional input layer leads to improved speed performance of the classifier.

Regarding the size of the packet, it could be considered that larger image size will result in better accuracy results, due to the increase of information about each packet that is provided to the neural network. In our preliminary results, we have observed that increasing the image size from 28×28 pixels to 32×32 pixels significantly improves neural network performance in packet recognition tasks, with accuracy increasing from 56% to 66%. However, it is important to note that further increases in image size beyond 32×32 pixels may not necessarily lead to performance improvements. For example, we found that 38×38 pixel images produced the same 66% accuracy. In this work, a size of 1024 pixels has been used, i.e. square images of 32×32 pixels, so that we do not exceed the typical MTU of 1500 bytes. We chose empirically this size because smaller images yield worse performance, while larger ones do not exhibit any noticeable improvement.

3.2. Organizing traffic flows

To avoid any data leakage from packets of the same flow in training and test datasets, we propose a folder structure that allows having the packet images of the same flow in a common path. This will be useful for testing with independent inputs in the model. As shown in Figure 2, a folder structure with 3 levels was utilized. At the first level, all the folders with different traffic captures from the dataset are organized, totaling 42 groups of applications and services. Within these folders, at the second level, all flows that comprise the traffic capture are stored. This means that each capture can contain a varying number of flows, and each flow can have a different number of packets. At the third and final level, each flow is divided into 1D and 2D images. We did not consider higher dimension images, as state of the art did not show substantial improvements. These images are stored in the

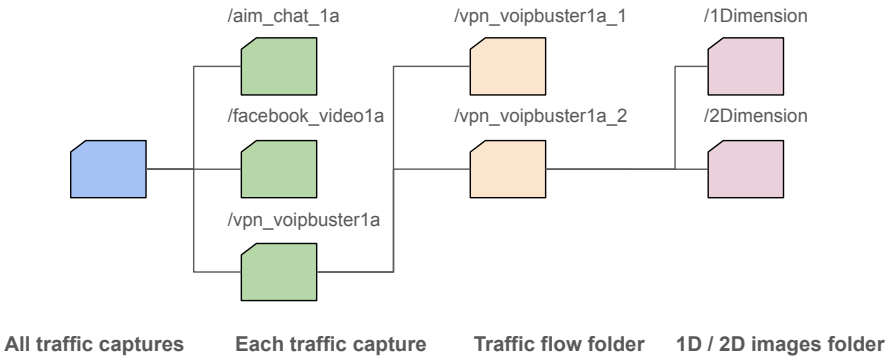


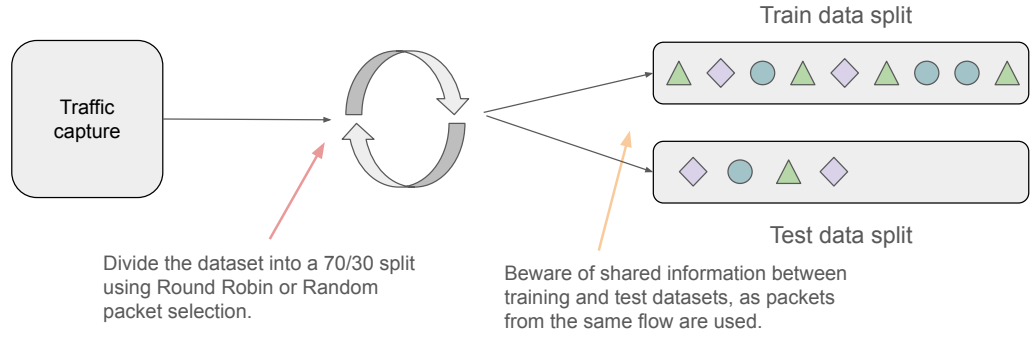
Figure 2. Proposed folder structure for traffic classification

respective folders and are used as input files for model training or testing. A problem with this system of grouping images by flows is that in some cases, such as file transfer type applications, there will be very few flows but with thousands of images each. However, in other cases we may have many small streams with few images. Our proposed solution is limiting how many packets are used per traffic flow, so that there are not large differences in the number of packets of each flow. Similarly, the work in [29] developed traffic classifiers with high accuracy percentage that used only the first 20 packets of each flow. However, this may introduce bias in the training, since connection closure will not be observed in all flows.

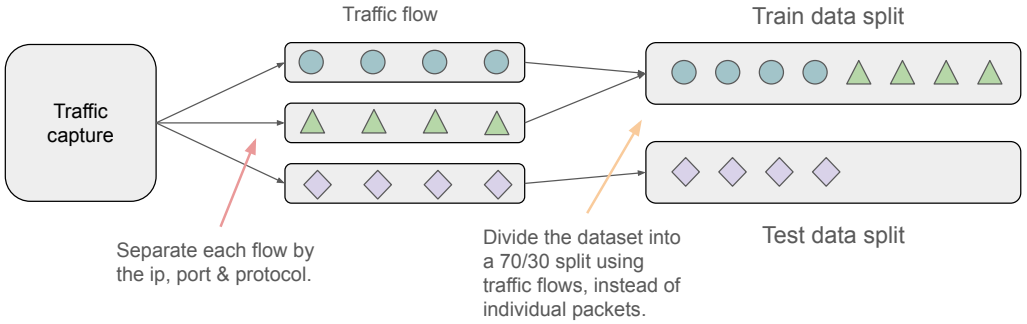
Figure 3a illustrates the standard dataset division process as described in the reviewed literature. In this scenario, each packet from a traffic capture (represented as colored geometric shapes) is divided between train and test sets either using a round-robin algorithm or randomly. This approach can lead to data leakage during model testing, as packets from flows used in training share information. Figure 3b depicts our proposed methodology, which includes grouping packets into flows to ensure no information is shared between the train and test splits. This prevents overfitting and results in a model that should perform more consistently with what is expected in a real network environment. Hereafter, we will be naming this phenomenon 'flow dependence'.

With these considerations, we can perform a desired data-leakage-free train-test split. Suppose we want to divide the data set between 70% training files and 30% test files. In a first approach, the first 70% of the flows of the trace are assigned to the training dataset, and the last 30% are assigned to the test dataset. The advantage of this method is that it is easy to implement and debug. The main disadvantage is that the training flows will be the first of the capture and, in case there is a relationship among the flows, some over-fitting would be introduced. An alternative is to perform a round-robin system between the flows. In this way, the first 7 flows would be for training, the next 3 for testing, then 7 again for training, etc. This method corrects the possible over-fitting introduced due to the order in the flows. The last proposal consists of randomly choosing 70% of the flows in training and 30% in test. This method is very difficult to debug and check for errors. On the other hand, there will be no problem in the order of flows and their relation and over-fitting.

In summary, this section has detailed the process on how to avoid bias in the classification of encrypted traffic packets using convolutional networks. Our assumption was that the current literature and authors were not giving enough effort in the handling and pre-processing of the payload data. Now, with the given preliminary results, we have shown that overfitting and model bias might be common in the literature. Our aim in the next sections will be to study how is the neural network processing the input data in order to give transparency in the decision-making and audit these black-box-like classification systems.



(a) Independent flows



(b) Dependent flows

Figure 3. Comparison between both methods for dataset split.

4. Model Definition

After reviewing all the publications of Section 2, we have come up with a general schema of a convolutional architecture, depicted in Figure 4. It is divided in two parts:

1. **Feature extraction**: in the very first part, the packets are inputted to the network, either as 1D or as 2D tensors. Then, they are fed into a convolutional layer. This convolutional layer uses small squared kernels (typically using an odd number, such as 3 or 5, as kernel size) to apply a convolution with an optional activation function, such as sigmoid or ReLU. The output of this layer is as many images as the number of kernels. However, this convolutional layer lacks a reduction of dimensionality for a proper feature extraction. To perform a reduction of the size of the image, a max-pooling layer is typically employed. This layer consists of reducing the dimension by 2 or 3 on each axis by summarizing the information of squared patches of pixels into a single pixel that represents the maximum of the patch. One could have considered other pooling operations, such as the minimum or the mean, but usually the maximum works better when using ReLU activations. These steps are repeated several times (D times in Figure 4) until the image is small enough to have captured few but interesting features.
2. **Feature analysis**: once we have extracted features using convolutional mechanisms, the model just employs simple dense layers using standard activation functions to provide a final output. For this, we have to flatten the images by just concatenating them or using an RNN. By doing this, we get an analysis of the features that is concluded with the final layer that provides a probability by using a softmax activation function. Additionally, to avoid overfitting, weight regularizers and dropout mechanisms between each layer can be added during training.

With this architecture, we have built a generic CNN to be evaluated. The final selection of hyperparameters was made after a tuning process, for the best-in-class results of the model, so it would benefit the most the model we are trying to optimize. In particular,

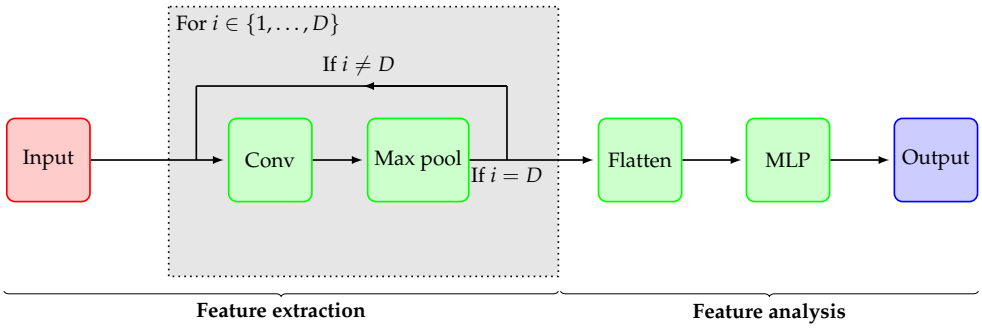


Figure 4. General CNN architecture with D convolutional stages.

Table 2. Model parameters

Common parameters	
Optimizer	Adam
Kernel Size	3
Dropout Rate	0.5
Activation	ReLU
Loss func.	Cross-entropy
Epochs per training	50
1D-model	
Max-Pool Size	2
Number of Conv2D stages	2
Number of fully-connected stages	2
Number of LSTM stages (when used)	2
2D-model	
Max-Pool Size	2x2
Number of Conv2D stages	4
Number of fully-connected stages	4
Number of LSTM stages (when used)	2

Table 2 describes the 1D-CNN and the 2D-CNN models, respectively. Both of them use just two convolutional stages ($D = 2$) since the size of the tensors after them is already small enough to do feature extraction. All activation functions, but the one of the output, are ReLU, given its good performance and well-known characteristics to avoid any vanishing gradient issue. Since the size of the image is divided by two in each convolutional stage, we are multiplying the number of kernels by two in each stage, i.e. we have many smaller images and each of them focuses on a particular feature of the packet. For instance, for the 2D-CNN model, we have that the input is one image of 32×32 pixels and the output of the feature extraction is 64 images of sizes 6×6 (or 8×8 if we had considered a convolution layer with border padding).

5. Gradient analysis

As we have seen, proper separation in flows is needed in order to avoid overfitting or data leakage and reaching a better overall performance. This conclusion was made with the network behavior analysis when exposed to data of the same flow. In order to prevent this type of problem in the future, we looked for a way to get an explanation of the behavior of the models. For this purpose, we apply GradCAM, known as gradient-weighted class-activation mapping. This XAI algorithm was presented initially by [39] and used for image-based classification, captioning, and visual question answering model, such as ResNet [40]. However, this algorithm is applicable to a wider variety of model families, such as any convolutional neural network (CNN) model with fully-connected layers, with

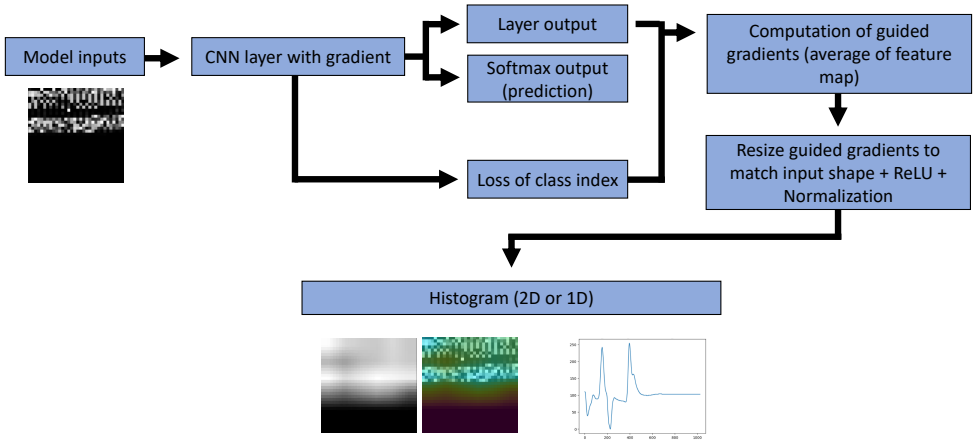


Figure 5. Histogram computation diagram in GradCAM.

structured output or multi-modal inputs. GradCAM is a generalization of CAM for CNN architectures [41].

The main concept behind GradCAM is that deeper representation on a CNN capture higher-level visual construct, and convolutional layers naturally retain spatial information, which is lost in fully-connected layers. As such, the last convolutional layers should have a spatial representation of the features extracted from the input, so the algorithm uses the gradient information flowing into the last convolutional layer of the CNN to identify important regions considered by the model for the decision. We will now briefly explain the algorithm, visually shown in Figure 5.

To obtain the class activation map or heatmap, we first need to find in our model the CNN layer that we want to extract the gradient from. In Figure 4, this is the convolutional layer when $i = D$. As a result, we get (i) the inputs of the model X; (ii) the output $A = [A^1, \ldots, A^k]$ of the CNN layer, which consists of k tensors of the same size; and (iii) the output y of the softmax activations from the model. As an observation, GradCAM focuses on the feature extraction part of the model, not in the feature analysis part. In that sense, GradCAM is agnostic to the type of convolutional model as long as the input is fed into some type of CNN that retains spatial information, so that the extracted heatmap is spatially correlated to the input X.

We fed the input X, the bytes from the packet payload, into the model, while we retain information of the gradients of y with respect to A computed with backpropagation, i.e. $\frac{\partial y}{\partial A}$.

These gradients are used together with A to compute a heatmap for each class. Finally, the heatmaps are normalized and scaled up to the original size of the input X. Usually, the last step is made when a representation with both the original image and the activation map is desired. However, for some cases where we want to extract the data as-is, we can skip the normalization step.

In addition to the standard use of GradCAM, that is, to detect relevant areas of information for the model for a certain input, you can also obtain explanations that highlight support for regions that would make the model change the prediction. This is done by using the gradient of the opposite of the score of each class with respect to A. This is called counterfactual explanations, and further reinforces the insight the standard model might give. These counterfactual explanations are very useful in non-binary classification problems, as they show what features are they seeing to determine that a possible packet does not belong to a category. Merging both results, we get both the most important features the model considers for the prediction, positive and negative, as shown in Figure 6.

More technically speaking, to obtain the heatmap L^c, we have to study the relation between each output tensor A^k of the last convolutional layer and the output of a class y^c, which is just the c component of vector y. For each element of A^k, we compute a measure of how that element contributes to y^c. In particular, two options can be used: (i) relying on the gradients, i.e. for the i-th element of the tensor

$$g_c^k(i) = \frac{\partial y^c}{\partial A_i^k}; \tag{1}$$

(ii) or use guided gradients, i.e. for the i-th element of the tensor

$$g_c^k(i) = \begin{cases} \dfrac{\partial y^c}{\partial A_i^k} & \text{if } A_i^k > 0 \text{ and } \dfrac{\partial y^c}{\partial A_i^k} > 0 \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

By using the first, we just consider the gradients in every part of A^k, while in the second approach we just take into consideration elements whose value is positive, i.e. they would trigger the ReLU activation, and whose gradient is also positive, i.e. contribute towards class c in the classification decision.

Using either of the previous gradients, we compute the average score of A^k,

$$a_k^c = \underset{i}{\text{mean}}\, g_c^k(i), \tag{3}$$

which represents how much A^k contributes to class c. Moreover, it is worth noting that this works for both 1D, 2D or n-dimensional CNNs, since we compute the gradient at each element of A^k and they are averaged all together ignoring their original position or the 2D/3D structure of A^k.

After obtaining the averaged gradient score, a partial linearization is made to only keep the parts that produce a positive response, this is,

$$L^c = ReLU\left(\sum_k a_k^c A^k\right) = \max\left(0, \sum_k a_k^c A^k\right). \tag{4}$$

Similarly, (3) can be modified to obtain counterfactual explanations, i.e. zones that discard a class, contributing negatively to y^c. This is done by computing the gradient of $-y^c$. Similarly to (1),

$$\bar{g}_c^k(i) = \frac{\partial(-y^c)}{\partial A_i^k} = -\frac{\partial y^c}{\partial A_i^k} = -g_c^k(i), \tag{5}$$

so essentially $\bar{a}_k^c = -a_k^c$ when using gradients. However, for guided gradients, there is no relevant relation between these two quantities:

$$\bar{g}_c^k(i) = \begin{cases} -\dfrac{\partial y^c}{\partial A_i^k} & \text{if } A_i^k > 0 \text{ and } \dfrac{\partial y^c}{\partial A_i^k} < 0 \\ 0 & \text{otherwise.} \end{cases} \tag{6}$$

In both cases, the weights of each A^k can be computed using the previous gradients

$$\bar{a}_k^c = \underset{i}{\text{mean}}\, \bar{g}_c^k(i), \tag{7}$$

yielding the final expression for the counterfactual relevance map:

$$\bar{L}^c = ReLU\left(\sum_k \bar{a}_k^c A^k\right). \tag{8}$$

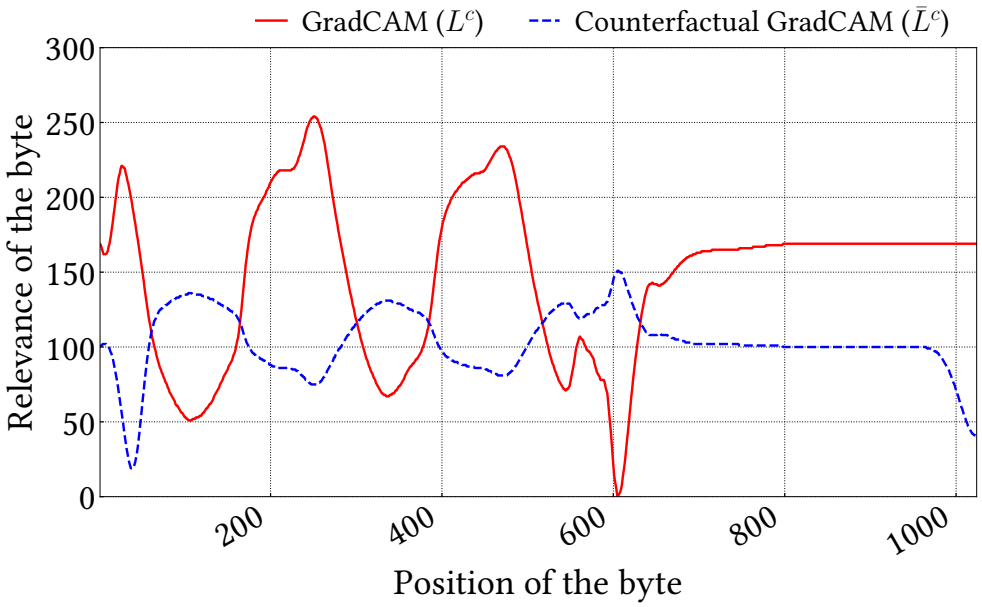


Figure 6. Relevance values given by GradCAM-1D when using both standard and counterfactual analysis.

Together, L^c and $\bar{L}^c$ explain the decision of the model. On the one hand, L^c describes the regions that contribute to choose class c. On the other hand, $\bar{L}^c$ describes which regions hinted not to choose class c.

Another consideration that must be made is that on multi-class applications like this one, this technique only presents data relevant to the predicted class. For example, if a packet has been identified as "Chat", the information that GradCAM will be able to provide will be related to whether that packet has been identified as "Chat" and not other classes.

GradCAM has been wildly tested and validated in computer vision problems (such as weakly-supervised localization, segmentation, class discrimination, the identification of bias in datasets, image captioning or visual question answering) [41–43], but the same principles can be applied to a network management problem involving CNNs, like the one at hand in this article.

The specific implementation of this algorithm for the model used in network flow classification focuses on the gradients obtained in the feature extraction part. Therefore, despite RNNs or MLPs can be used to find other correlations in the feature analysis, GradCAM will not consider them. Nevertheless, it still gives a valid metric to understand the general behavior of the convolutional model, as the later layers can only work with the extracted features.

In the next section, we will discuss how GradCAM was applied for both 1D-CNNs and 2D-CNNs and what are the explanations of the decisions of the models.

6. Results

In this section, we present the results of our experiments grouped in four categories. First, we analyze the performance of our model compared to the state of the art. Second, we focus on the data, and how some characteristics of the input data may alter significantly the performance. Third, the same process is repeated for the model and its parameters. Last, we apply XAI to understand the decision of the model.

Table 3. Global metrics of the proposed models

	Precision	Recall	F-score
1D-CNN	0.845	0.86	0.852
2D-CNN	0.734	0.71	0.721

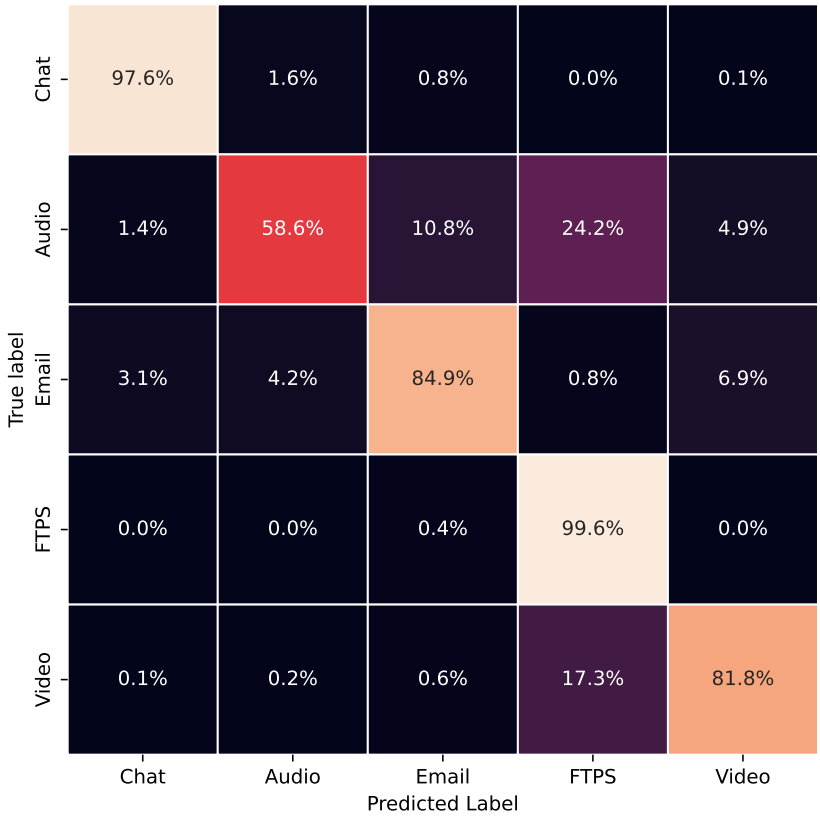


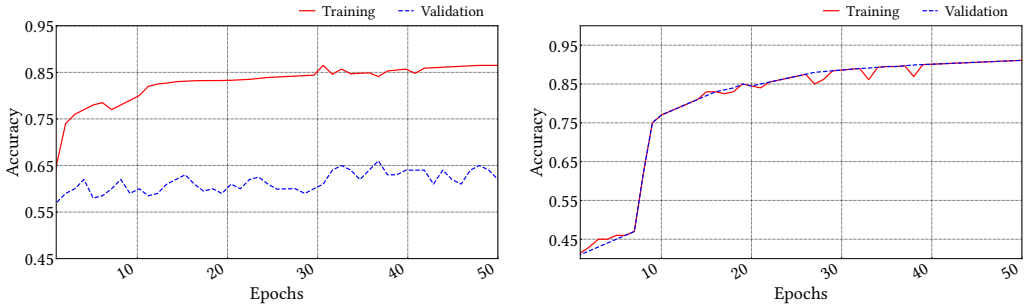
Figure 7. Final Confusion Matrix model using 1D input images and client-server flow direction separated.

6.1. Performance of the model

In section 4, we presented the CNN architecture and two models, 1D-CNN and 2D-CNN. Both of these networks, although they can be trained for any purpose, we are particularly interested in classifying packets in traffic classes—e.g., video or chat—for QoS policing purposes.

After dividing the dataset into 70% training and 30% test randomly, an accuracy of 90% was easily obtained with both models, comparable to state-of-the-art alternatives [25,27,36]. However, as we explained previously, we believe that packets of the same flow can only belong to either the training or the test set.

Consequently, we present in Table 3 the performance metrics of the 2D-CNN model using a train-test split with the flow division. All of them; precision, recall and F1-score; are around 85%, which is comparable with the existing architectures and other alternatives. The small 5% differences, as it is covered in next subsection, is due to the flow dependence. Furthermore, Figure 7 displays the confusion matrix for the 1D model after some data preprocessing. This shows that the error is mostly focused on the audio class, being this confused with e-mail and video classes.



(a) Independent flows **(b)** Dependent flows

Figure 8. Training and validation of the base model with dependent and independent flows.

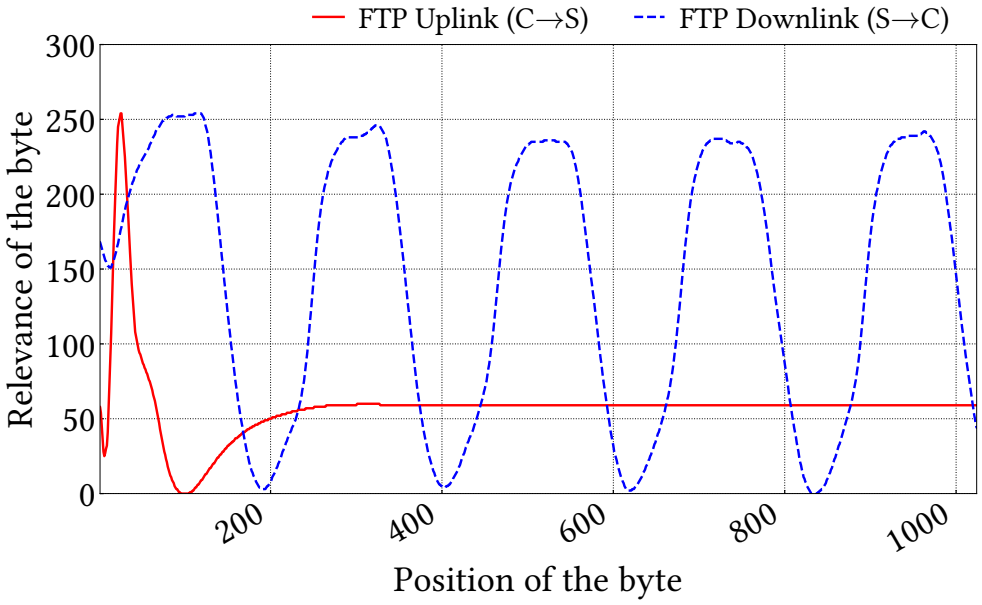


Figure 9. GradCAM histogram of FTP client vs server.

6.2. Understanding your data

There are some aspects of the data that are worth mentioning: the flow dependence and its effects, the effect of direction of the flow, and the padding of small packets.

In order to study the impact of flow dependence, we trained the same model with two different datasets. First, we trained the model with just a random split over the packets, as shown in the Figure 8b. Using a validation set of random packets, we managed to observe that both training and validation curves overlap, showing accuracy above 90%. On a first view, this would mean that the model is not overfitting, and the performance is optimal.

On the other hand, when training and validating the system with independent flows, performance change drastically. While the accuracy in training was about 90%, independent flows of the validation set showed that the real number is more around 65%. This is, the model is not learning to generalize the specific per-class characteristics of the packets, but learning fixed patterns that are related to the particular flows or servers involved. This overfitting due to data leakage renders the model incapable of handling real-life situations where all data might be coming from systems not present in the training data.

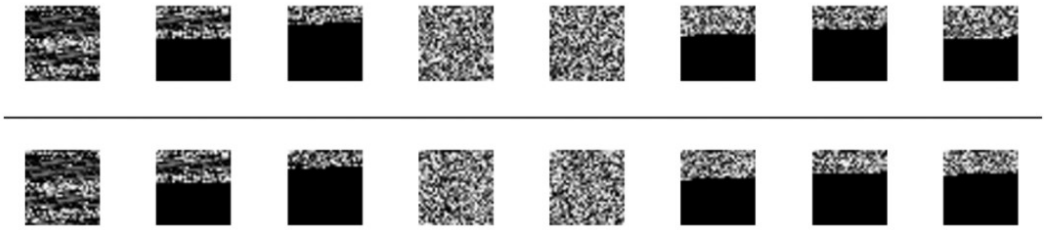
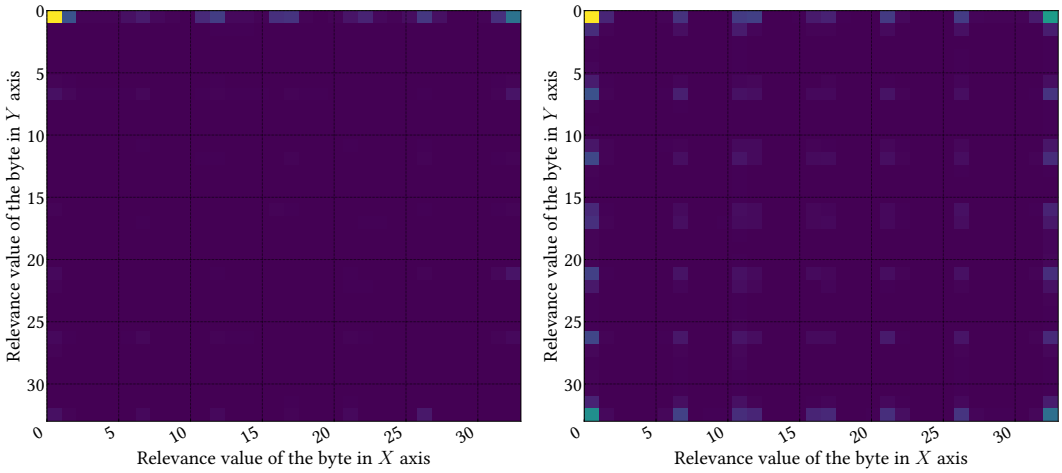


Figure 10. Payload images of the packets in two different Chat flows.



(a) E-mail **(b)** FTP

Figure 11. 2D histograms of the relevance obtained with GradCAM-2D for two classes

Second, we analyze the difference depending on the direction of the flow. Figure 9 shows the Grad-CAM relevance for the class FTP depending on the direction. Both lines show the relevance of each byte of the packet, denoted by its position, when processed by the CNN, showing differences in uplink (client to server) and downlink (server to client) for FTP traffic. Due to the asymmetry of the roles of client and server in this case, uplink relevance is focused on the first bytes of the packets, where request should have the most relevant part, while downlink relevance is more spread in higher bytes. This makes clear that bidirectional flows should not be mixed, since they can have different and confusing behaviors if considered together.

Third, we study the impact of padding, this is, filling with extra data when the packet does not reach the desired size of 1024. This can be done in two ways: (1) using random padding, i.e., adding random bytes at the end of the packet; or (2) using zero padding, i.e., adding 0 bytes to fill the rest of the packet. Although both options might be reasonable, they hide the attribute of the size. If we use the first option, we are considering that the size of the packet is not an important attribute, and we are hiding that information adding data to maintain the entropy as it were encrypted. In contrast, the second option exposes the end of the packet, since models should be able to notice the position of a final position of zeros.

In this case, we believe that packet size is an important attribute that intuitively can help identify traffic classes, since we expect that larger packets should belong to data transfers and small one to other delay-sensitive services. Therefore, we only consider zero padding, as shown in Figure 10, the bottom of 2D images is filled with black pixels.

6.3. Understanding the model

With all the previous observations about the data, we are now able to understand better what the model might be doing. In this point, we study what are the difference between both models and how some parameters may affect the performance.

As we mentioned in previous sections, both models, 1D-CNN and 2D-CNN, perform similarly. As other studies highlighted [28], the difference between models is noticeable but minor. When using a 1D approach, we assume that the sequence of bytes is spatially correlated, i.e., close bytes might have a relation. On the other hand, 2D models do consider this relation adding an extra relation with bytes that are before or after a fixed offset. Although this might happen, we did not observe it when plotting data and 2D relevance. Figure 11 depicts the relevance of each position of the 2D representation to showcase the absence of strong 2D relations. Two examples are shown: for the class E-mail (Figure 11a), there is no clear relation; and for the class FTP (Figure 11b), there is some spurious 2D relation that happen to be periodical but with a small period of length of five to eight bytes. Additionally, this 2D metric is sensitive to the width of the image, which affects the generality if, for some reason, this offset changes depending on the environment.

Apart from some hyperparameters that can be optimized and depend on the amount of data such as the number of convolutional stages in the feature extraction, we noticed that the amount of convolution padding (different from the packet padding) might have an effect on the data. Convolution padding is an important parameter of convolutional layers, because the borders of the signal are not considered if we do not add a padding for the convolution. For 1D, this means adding extra data at the beginning and the end, while for the 2D, this happens every time we reach the borders of the image. Figure 12 represents the effect of convolution padding length for the relevance of the 1D model. Previous works [44] had already used convolution padding successfully. Different types of border padding were tested, but the use of "same" padding (replicating the last value in the border) yielded the best results, as the model did not lose spatial data, unlike padding with zeros or the max value. Another test was made using a padding with an arbitrary number of pixels in the test set, so it would permanently modify the length size of all the packets, and the classification capabilities of the model were reduced significantly to the extent that it could not work as intended. Other works also see that packet size is a relevant metric when analyzing performance of classifiers made with CNNs [12]. Upon testing different widths, we observed a clear soften of the borders of the relevance when using padding larger than 1 pixel, which may be the result of giving the gradient further space to develop, as per Figure 12. We settled on using 2-pixel padding here onwards, as it provided generic enough patterns to analyze and giving a clear outline identified for that packet position in the flow.

6.4. Understanding the decision

After studying the data and the model, we can explore the classification results using XAI techniques. In this section, we use GradCAM relevance for the different classes of the dataset to analyze the feature extraction procedure of CNNs in encrypted packets. After understanding the features, we provide a simplified ML model that achieves a similar performance with much less complexity.

First, the relevance per sample is computed and displayed in left-hand side of Figure 13. All of them but Figure 13g display a similar behavior: they are smooth functions with some random noise. In order to model this, we provide a median function to act as centroid of the groups that, by nature, should be less noisy. Right-hand side of the figure represents the same median behavior as representative of the relevance of each byte together with the histogram of packet lengths.

Figure 13a shows the relevance L^c and counterfactual relevance $\bar{L}^c$ according to Grad-CAM for class Chat. In general, $\bar{L}^c$ is always above L^c except for an environment centered in 300 bytes, where L^c increases and surpasses $\bar{L}^c$. As shown in Figure 10, packets in this class have different sizes, and their images are filled with zeroes depending on such sizes. Thus, when looking at Figure 13b, we observe that this point is no random, but the mode of

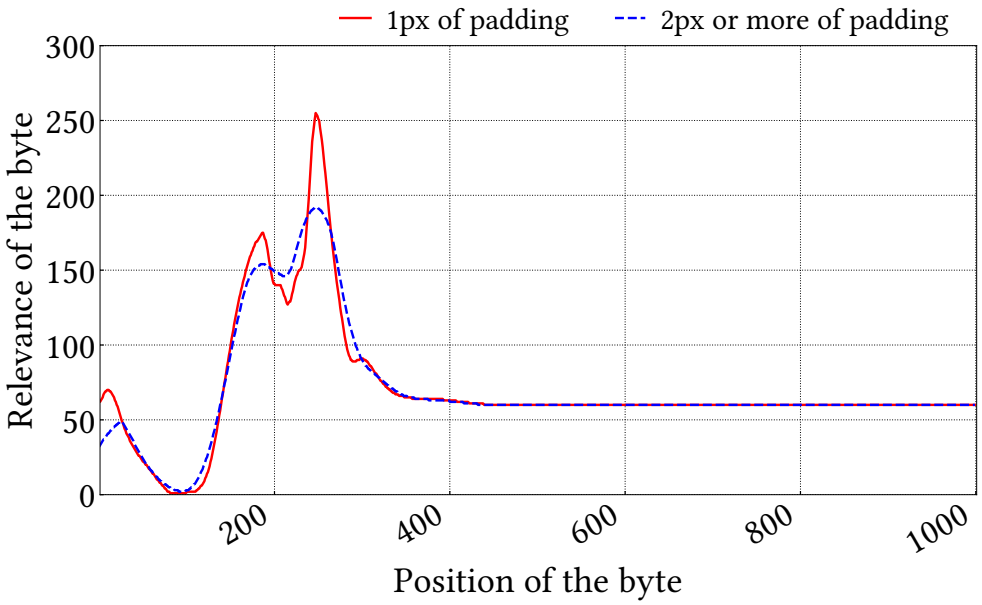


Figure 12. Difference between the use of 1 pixel padding vs 2 pixels or more in GradCAM.

the packet length. This means that the model is mostly focusing on the last bytes of content of the packet. Due to the convolutional nature of the CNNs, they are powerful to detect borders (abrupt changes from high values to low values), so in particular, it is likely that the convolutional model is just extracting the feature that the packet is ending at around 300 bytes. Bear in mind that we have used zero padding to fill the packet images, so the CNN is able detect this abrupt change to a zero-valued region as the end of each packet.

Similarly, Figure 13c exposes the relevance and counterfactual relevance of class Audio, where same behavior at around 100 bytes. This is confirmed by Figure 13d, with a mode of the packet length at around 100 bytes.

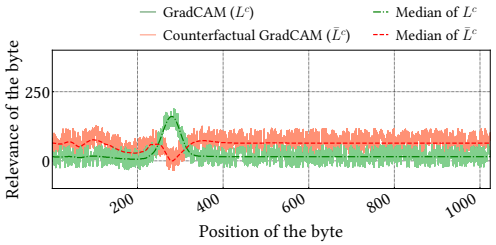
For class E-mail, the result is not exactly the same. In this case, the relevance L^c of Figure 13e is always above $\bar{L}^c$ except a small interval centered in 100 bytes. In this case, by looking at the histogram in Figure 13f, $\bar{L}^c$ highlights that values around 100 bytes of packet size are not likely for class E-mail.

Figure 13g shows the hardest to categorize behavior of all classes, class FTP. Due to the spread value of the packet sizes, models are showing values of L^c spread around all values with only the important maximum at 1024 bytes. In particular, this maximum is confirmed with the mode of the packet size in Figure 13h. This confirms the poor performance of this class observed in the confusion matrix (Figure 7), due to the overlapping of the packet length histograms among the classes.

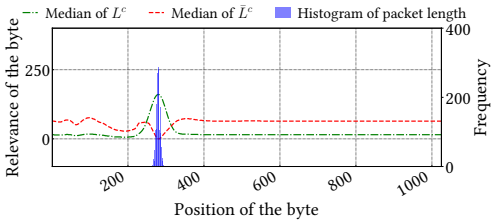
Class Video in Figure 13i exhibits a multi-modal relevance behavior with modes around 50, 125 and 1024 bytes. For the rest of the values, L^c is equal to $\hat{L}^c$. In the histogram of Figure 13j, we confirmed yet another time the hypothesis of the packet size being the most relevant attribute in encrypted traffic classification.

These results allow us to conclude that CNNs are overcomplicated models to classify high-entropy payloads that provide only the information of the size of the payload. With this in mind, our next step is to provide a simpler model that uses only the packet size to check whether similar performance can be achieved with a higher packet classification rate in packets/s.

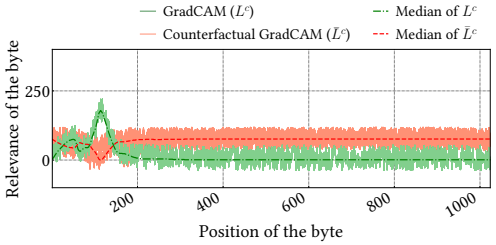
For this sake, we built a decision tree that uses only packet size for encrypted packet classification. The performance obtained across all metrics is around 80%, comparable to the 85% obtained before. This evidence supports our hypothesis that packet size is the most



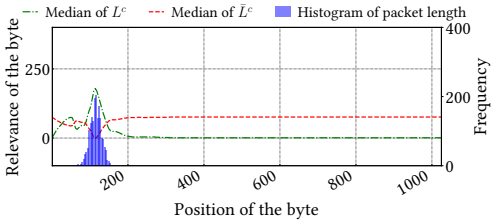
(a) Relevances for class Chat



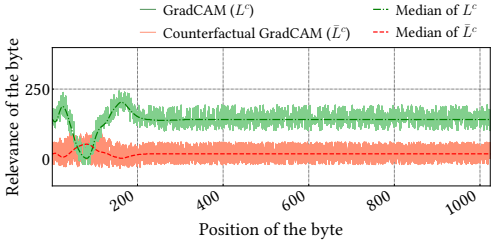
(b) Packet length histogram for class Chat



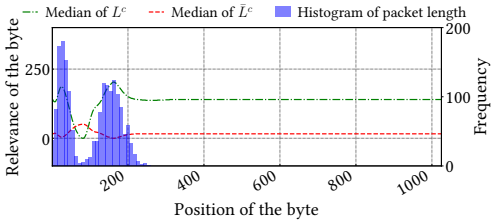
(c) Relevances for class Audio



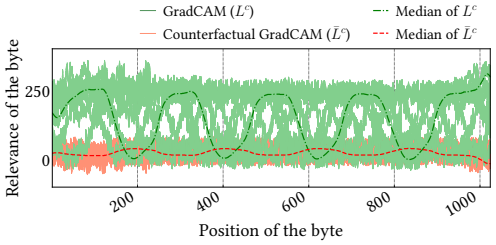
(d) Packet length histogram for class Audio



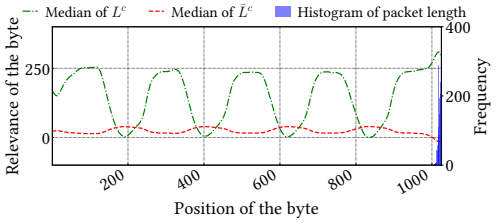
(e) Relevances for class E-mail



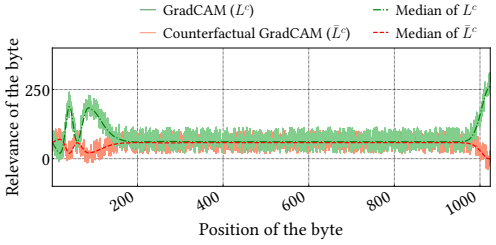
(f) Packet length histogram for class E-mail



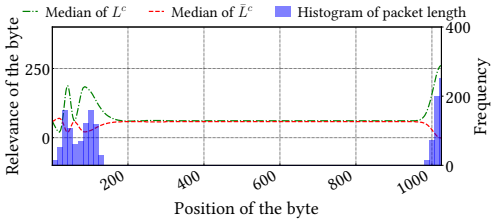
(g) Relevances for class FTP



(h) Packet length histogram for class FTP



(i) Relevances for class Video



(j) Packet length histogram for class Video

Figure 13. Relevance and counterfactual relevance compared to packet length per class

important attribute and that encryption poses quite a challenge that CNNs are not yet able to solve effectively. Figure 14 shows the trained decision tree, showing the simplicity and minimal number of parameters of the provided model. Moreover, this solution can run much faster than a CNN, which is necessary in this scope to classify the network packets in real time.

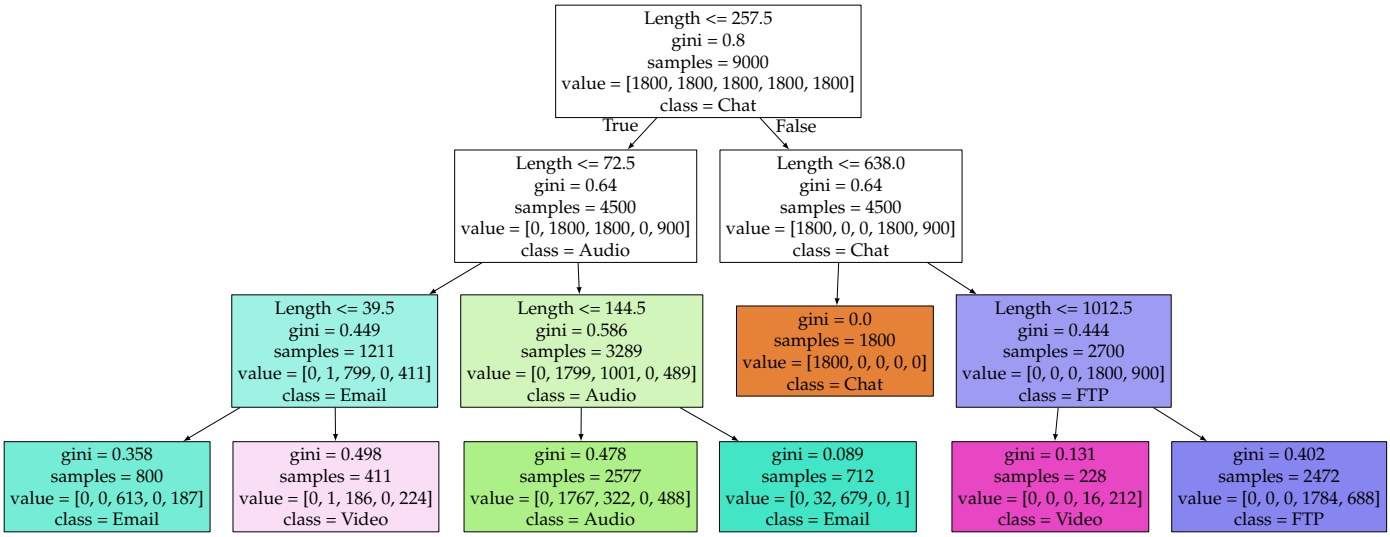


Figure 14. Decision tree for encrypted packet classification

7. Conclusion

In summary, we have analyzed the key aspects from packet classification CNN-based methodologies, with a focus on their wise choices, to replicate them, and mistakes, to avoid them. With such information, we built a model that achieved state-of-the-art performance for the most used dataset, ISCX VPN vs nonVPN [33], and analyze the model predicted traffic class using GradCAM, a XAI technique for convolutional feature extraction. The conclusion of the analysis is that, after removing all kind of biases that identify the flow or the servers, the model just looks at the packet size to make a decision. Consequently, we are able to match performance by just using a decision tree on the packet size. Despite having analyzed CNNs for the most popular dataset in encrypted network packet classification, this methodology can be applied to other datasets with different categories. Next, we detail the most important contributions of this work.

1. *Poor experimental protocols result in an unrealistic performance evaluation*: As it has been shown, the bias introduced in CNN training and validation datasets has not been sufficiently taken into consideration before. This can be demonstrated by separating the traffic flows in the dataset preparation process. The neural network will learn based on data closer to what can be expected from a real traffic network, however the real accuracy will be limited compared to the theoretical results of other authors. We propose a better experimental protocol by ensuring that biases are eliminated from the dataset and avoiding data leakage from the training set to the test set.
2. *XAI techniques are of great help in the assessment of the model*: Using the XAI techniques discussed above, it has been shown that caution must be had when preprocessing data, as the model might find not be able to properly identify the classes if two flows that behave differently (client and server flows) are aggregated into one category. Moreover, the use of the heatmap shows that the correlations found in the trace by the model are purely statistical and not based on concrete evidence or certain values of metadata, so more strict codification that further diffuses the statistical traits of the payload will affect the accuracy of the model significantly. The use of heatmaps helps to understand the viability of the model in practical scenarios, where the point

of interest of the model can be extracted and discussed in order to determine if the training exercise was successful or not.

3. *CNNs do not detect significant features in encrypted payloads*: Based on the presented results, CNN do not seem to be the right tool to classify encrypted network packets, as they are just focusing on the packet length. Just adding random length padding to the packets of encrypted traffic would be enough to make such classification unfeasible, at the cost of more payload per packet. This could be a good solution for users' privacy, but an inadequate one for the network operators and systems, which would need to find other solutions to classify the network traffic and handle the extra computing load and bandwidth. On the other hand, before packets would include random padding, this conclusion provides a valid means to heuristically classify the traffic at high speed, just by looking at the packet lengths.

Regarding limitations, in this work we have addressed only CNNs with a technique, GradCAM, specific for CNNs. Using a generic CNN allowed us to focus on the convolutional features in network encrypted packets. While GradCAM allowed us to extract deeper information about CNNs, it also limited our study to convolutional features. Nowadays, other techniques—e.g. Transformers—are emerging, and their rapid development poses again the challenge of understanding what they are learning. This opens the topic of extracting and understanding more complex features to improve the differentiation between traffic patterns.

For the reproducibility of our results, we have published the code and methods in GitHub[1]. Furthermore, the same repository includes tools to preprocess and split datasets to build an appropriate experimental protocol with no biases.

Author Contributions: Conceptualization, J.E.L.d.V.; methodology, J.E.L.d.V.; software, E.L.B, V.M.G.; validation, E.L.B, V.M.G. and D.P.B.; formal analysis, D.P.B.; investigation, E.L.B, V.M.G.; resources, J.E.L.d.V.; data curation, E.L.B, V.M.G.; writing—original draft preparation, E.L.B, V.M.G., D.P.B and J.E.L.d.V.; writing—review and editing, E.L.B, V.M.G., D.P.B and J.E.L.d.V.; visualization, E.L.B, V.M.G. and D.P.B; supervision, J.E.L.d.V.; project administration, J.E.L.d.V.; funding acquisition, J.E.L.d.V. All authors have read and agreed to the published version of the manuscript.

Funding: This research has been partially funded by the Spanish State Research Agency under the project AgileMon (AEI PID2019-104451RB-C21) and by the Spanish Ministry of Science, Innovation and Universities under the program for the training of university lecturers (Grant number: FPU19/05678).

Data Availability Statement: The data presented in this study were derived from the following resources available in the public domain: ISCX VPN-nonVPN dataset [33]. The code to process these data is available at GitHub [1].

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

[1] https://github.com/elbisbe/XAI_DL_NetworkPacketClassification

1D	1-dimensional
2D	2-dimensional
3D	3-dimensional
ACK	Acknowledgement
AI	Artificial Intelligence
CNN	Convolutional Neural Network
DL	Deep Learning
DNS	Domain Name System
DPI	Deep Packet Inspection
FPGA	Field Programmable Gate Array
FTP	File Transfer Protocol
GPU	Graphics Processing Unit
GradCAM	Gradient-weighted Class Activation Mapping
IP	Internet Protocol
ISP	Internet Service Provider
MDPI	Multidisciplinary Digital Publishing Institute
ML	Machine Learning
MTU	Maximum Transmission Unit
OTT	Over-The-Top
QoS	Quality of Service
ReLU	Rectified Linear Unit
RGB	Red, Green, Blue
RNN	Recurrent Neural Network
SVM	Support Vector Machine
SYN	Synchronize
TCP	Transport Control Protocol
TLS	Transport Layer Security
UDP	User Datagram Protocol
VPN	Virtual Private Network
XAI	eXplainable Artificial Intelligence

References

1. Zeidanloo, H.R.; Manaf, A.B.A. *Botnet Detection by Monitoring Similar Communication Patterns*; Vol. abs/1004.1232, LAP LAMBERT Academic Publishing, 2010.
2. Bremler-Barr, A.; Harchol, Y.; Hay, D.; Koral, Y. Deep packet inspection as a service. In Proceedings of the Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies, 2014, pp. 271–282.
3. Yuan, R.; Li, Z.; Guan, X.; Xu, L. An SVM-based machine learning method for accurate internet traffic classification. *Information Systems Frontiers* **2010**, pp. 149–156.
4. Rezaei, S.; Liu, X. Deep learning for encrypted traffic classification: An overview. *IEEE Communications Magazine* **2019**, *57*, 76–81.
5. Xin, R.; Zhang, J.; Shao, Y. Complex network classification with convolutional neural network. *Tsinghua Science and Technology* **2020**, *25*, 447–457. https://doi.org/10.26599/TST.2019.9010055.
6. tao Ren, J.; ling Ou, X.; Zhang, Y.; cheng Hu, D. Research on network-level traffic pattern recognition. In Proceedings of the Proceedings. The IEEE 5th International Conference on Intelligent Transportation Systems, 2002, pp. 500–504. https://doi.org/10.1109/ITSC.2002.1041268.
7. Roshan, K.; Zafar, A. Utilizing XAI Technique to Improve Autoencoder based Model for Computer Network Anomaly Detection with Shapley Additive Explanation(SHAP). *International journal of Computer Networks & Communications* **2021**, *13*, 109–128. https://doi.org/10.5121/ijcnc.2021.13607.
8. Zhang, T.; Qiu, H.; Mellia, M.; Li, Y.; Li, H.; Xu, K. Interpreting AI for Networking: Where We Are and Where We Are Going. *IEEE Communications Magazine* **2022**, *60*, 25–31. https://doi.org/10.1109/MCOM.001.2100736.
9. Aceto, G.; Ciuonzo, D.; Montieri, A.; Pescapé, A. Toward effective mobile encrypted traffic classification through deep learning. *Neurocomputing* **2020**, *409*, 306–315.
10. Beliard, C.; Finamore, A.; Rossi, D. Opening the Deep Pandora Box: Explainable Traffic Classification. In Proceedings of the IEEE INFOCOM 2020 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), 2020, pp. 1292–1293. https://doi.org/10.1109/INFOCOMWKSHPS50562.2020.9162704.
11. Meng, Z.; Wang, M.; Bai, J.; Xu, M.; Mao, H.; Hu, H. Interpreting deep learning-based networking systems. In Proceedings of the Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication, 2020, pp. 154–171.
12. Ismailaj, K.; Camelo, M.; Latré, S. When Deep Learning May Not Be The Right Tool For Traffic Classification. In Proceedings of the 2021 IFIP/IEEE International Symposium on Integrated Network Management (IM), 2021, pp. 884–889.

13. Caforio, F.P.; Andresini, G.; Vessio, G.; Appice, A.; Malerba, D. Leveraging Grad-CAM to Improve the Accuracy of Network Intrusion Detection Systems. In Proceedings of the Discovery Science; Soares, C.; Torgo, L., Eds., Cham, 2021; pp. 385–400.
14. Nascita, A.; Montieri, A.; Aceto, G.; Ciuonzo, D.; Persico, V.; Pescapé, A. XAI Meets Mobile Traffic Classification: Understanding and Improving Multimodal Deep Learning Architectures. *IEEE Transactions on Network and Service Management* **2021**, *18*, 4225–4246. https://doi.org/10.1109/TNSM.2021.3098157.
15. Xie, G.; Li, Q.; Jiang, Y.; Dai, T.; Shen, G.; Li, R.; Sinnott, R.; Xia, S. Sam: Self-attention based deep learning method for online traffic classification. In Proceedings of the Proceedings of the Workshop on Network Meets AI & ML, 2020, pp. 14–20.
16. Morales Gómez, V. Sistema de clasificación de paquetes a alta tasa utilizando redes neuronales convolucionales y FPGAs (High-rate packet classification system using convolutional neural networks and FPGAS). Master thesis, Máster Universitario en Ingeniería de Telecomunicación. Universidad Autónoma de Madrid, 2021.
17. Shapira, T.; Shavitt, Y. FlowPic: A Generic Representation for Encrypted Traffic Classification and Applications Identification. *IEEE Transactions on Network and Service Management* **2021**, *18*, 1218–1232. https://doi.org/10.1109/TNSM.2021.3071441.
18. Bovenzi, G.; Nascita, A.; Yang, L.; Finamore, A.; Aceto, G.; Ciuonzo, D.; Pescapé, A.; Rossi, D. Benchmarking Class Incremental Learning in Deep Learning Traffic Classification. *IEEE Transactions on Network and Service Management* **2024**, *21*, 51–69. https://doi.org/10.1109/TNSM.2023.3287430.
19. Banihashemi, S.B.; Akhtarkavan, E. Encrypted Network Traffic Classification Using Deep Learning Method. In Proceedings of the 2022 8th International Conference on Web Research (ICWR), 2022, pp. 1–8. https://doi.org/10.1109/ICWR54782.2022.9786247.
20. Jacobs, A.S.; Beltiukov, R.; Willinger, W.; Ferreira, R.A.; Gupta, A.; Granville, L.Z. AI/ML for Network Security: The Emperor Has No Clothes. In Proceedings of the Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, New York, NY, USA, 2022; CCS '22, p. 1537–1551. https://doi.org/10.1145/3548606.3560609.
21. Guarino, I.; Aceto, G.; Ciuonzo, D.; Montieri, A.; Persico, V.; Pescapè, A. Contextual counters and multimodal Deep Learning for activity-level traffic classification of mobile communication apps during COVID-19 pandemic. *Computer Networks* **2022**, *219*, 109452. https://doi.org/10.1016/j.comnet.2022.109452.
22. Izadi, S.; Ahmadi, M.; Rajabzadeh, A. Network Traffic Classification Using Deep Learning Networks and Bayesian Data Fusion. *Journal of Network and Systems Management* **2022**, *30*, 25. https://doi.org/10.1007/s10922-021-09639-z.
23. Appiah, B.; Sackey, A.K.; Kwabena, O.A.; Kanpogninge, A.J.A.; Buah, P.A. Fusion Dilated CNN for Encrypted Web Traffic Classification. *International Journal of Network Security* **2022**, *24*, 733–740.
24. Dong, S.; Xia, Y.; Peng, T. Traffic identification model based on generative adversarial deep convolutional network. *Annals of Telecommunications* **2022**, *77*, 573–587. https://doi.org/10.1007/s12243-021-00876-6.
25. Lotfollahi, M.; Jafari Siavoshani, M.; Shirali Hossein Zade, R.; Saberian, M. Deep packet: a novel approach for encrypted traffic classification using deep learning. *Soft Computing* **2020**, *24*, 1999–2012. https://doi.org/10.1007/s00500-019-04030-2.
26. Lee, K.H.; Lee, S.H.; Kim, H.C. Traffic Classification Using Deep Learning: Being Highly Accurate is Not Enough. In Proceedings of the Proceedings of the SIGCOMM '20 Poster and Demo Sessions, New York, NY, USA, 2021; SIGCOMM '20, p. 1–2. https://doi.org/10.1145/3405837.3411369.
27. Zeng, Y.; Gu, H.; Wei, W.; Guo, Y. $Deep - Full - Range$: A Deep Learning Based Network Encrypted Traffic Classification and Intrusion Detection Framework. *IEEE Access* **2019**, *7*, 45182–45190. https://doi.org/10.1109/ACCESS.2019.2908225.
28. Zhang, J.; Li, F.; Wu, H.; Ye, F. Autonomous Model Update Scheme for Deep Learning Based Network Traffic Classifiers. In Proceedings of the 2019 IEEE Global Communications Conference (GLOBECOM), 2019, pp. 1–6. https://doi.org/10.1109/GLOBECOM38437.2019.9014036.
29. Lopez-Martin, M.; Carro, B.; Sanchez-Esguevillas, A.; Lloret, J. Network Traffic Classifier With Convolutional and Recurrent Neural Networks for Internet of Things. *IEEE Access* **2017**, *5*, 18042–18050. https://doi.org/10.1109/ACCESS.2017.2747560.
30. Dong, S.; Li, R. Traffic identification method based on multiple probabilistic neural network model. *Neural Computing and Applications* **2019**, *31*, 473–487. https://doi.org/10.1007/s00521-017-3081-x.
31. Wang, W.; Zhu, M.; Zeng, X.; Ye, X.; Sheng, Y. Malware traffic classification using convolutional neural network for representation learning. In Proceedings of the 2017 International Conference on Information Networking (ICOIN), 2017, pp. 712–717. https://doi.org/10.1109/ICOIN.2017.7899588.
32. Wang, W.; Zhu, M.; Wang, J.; Zeng, X.; Yang, Z. End-to-end encrypted traffic classification with one-dimensional convolution neural networks. In Proceedings of the 2017 IEEE International Conference on Intelligence and Security Informatics (ISI), 2017, pp. 43–48. https://doi.org/10.1109/ISI.2017.8004872.
33. Draper-Gil., G.; Lashkari., A.H.; Mamun., M.S.I.; A. Ghorbani., A. Characterization of Encrypted and VPN Traffic using Time-related Features. In Proceedings of the Proceedings of the 2nd International Conference on Information Systems Security and Privacy - ICISSP,. INSTICC, SciTePress, 2016, pp. 407–414. https://doi.org/10.5220/0005740704070414.
34. Shiravi, A.; Shiravi, H.; Tavallaee, M.; Ghorbani, A.A. Toward developing a systematic approach to generate benchmark datasets for intrusion detection. *Computers & Security* **2012**, *31*, 357–374. https://doi.org/10.1016/j.cose.2011.12.012.
35. 3GPP. Policy and charging control architecture, 2021. TS 23.203, rel.17.
36. Hwang, R.H.; Peng, M.C.; Nguyen, V.L.; Chang, Y.L. An LSTM-based deep learning approach for classifying malicious traffic at the packet level. *Applied Sciences* **2019**, *9*, 3414.
37. Bermudez, I.N.; Mellia, M.; Munafo, M.M.; Keralapura, R.; Nucci, A. Dns to the rescue: Discerning content and services in a tangled web. In Proceedings of the Proceedings of the 2012 Internet Measurement Conference, 2012, pp. 413–426.

38. Wang, P.; Chen, X.; Ye, F.; Sun, Z. A survey of techniques for mobile service encrypted traffic classification using deep learning. *IEEE Access* **2019**, *7*, 54024–54033.
39. Selvaraju, R.R.; Cogswell, M.; Das, A.; Vedantam, R.; Parikh, D.; Batra, D. Grad-cam: Visual explanations from deep networks via gradient-based localization. In Proceedings of the Proceedings of the IEEE international conference on computer vision, 2017, pp. 618–626.
40. Wu, Z.; Shen, C.; Van Den Hengel, A. Wider or deeper: Revisiting the resnet model for visual recognition. *Pattern Recognition* **2019**, *90*, 119–133.
41. Rosebrock, A. Grad-CAM: Visualize class activation maps with Keras, TensorFlow, and Deep Learning, 2020.
42. Selvaraju, R.; Das, A.; et al. Grad-CAM: Gradient-weighted Class Activation Mapping. http://gradcam.cloudcv.org/.
43. Madhukar, B. Using Grad-CAM to Visually Verify the Performance of CNN Model. https://analyticsindiamag.com/using-grad-cam-to-visually-verify-the-performance-of-cnn-model/, 2020.
44. Singirikonda, M. How Padding helps in CNN ? https://mahithas.medium.com/how-padding-helps-in-cnn-2b87957e1b, 2020.

Short Biography of Authors

Ernesto Luis Bisbé is currently a data engineer at Telefónica (Spain). He received his B.Sc. and M.Sc. degrees in Telecommunication Engineering from Universidad Autónoma de Madrid (Spain) in 2020 and 2022, respectively. His research topics are network traffic measurement and network and service monitoring.

Víctor Morales Gómez is currently an observability engineer at Future Space (Spain). Previously, he worked at Naudit HPCN (Spain). He received his B.Sc. and M.Sc. degrees in Telecommunication Engineering from Universidad Autónoma de Madrid (Spain) in 2019 and 2021, respectively. His research topics are network traffic measurement and network and service monitoring.

Daniel Perdices is assistant professor at Universidad Autónoma de Madrid (Spain) since 2023. He previously held an FPU research scholarship by the Spanish Ministry of Science, Innovation and Universities. Previously, he was an R&D engineer at Naudit HPCN. He received the B.Sc. (Hons) degrees in Mathematics and in Computer Science (2018), the M.Sc. in Mathematics (2019), the M.Sc. in Information and Communications Technologies (2020), and Ph.D. in Computer Science and Telecommunication Engineering (2023), all at Universidad Autónoma de Madrid (Spain). He was a visiting scholar in 2022 for three months at SmartData@PoliTO, Politecnico di Torino (Italy). He researches on statistics, mathematical modeling, machine learning, network traffic analysis, and SDN.

Jorge E. López de Vergara is associate professor at Universidad Autónoma de Madrid (Spain) since 2007. He was accredited as full professor in 2023. He was founding partner until 2023 of Naudit HPCN, a spin-off company established in 2009, devoted to high-performance traffic monitoring and analysis. He received his M.Sc. and Ph.D. degrees in Telecommunication Engineering from Universidad Politécnica de Madrid (Spain) in 1998 and 2003, respectively, where he also held an FPU-MEC research scholarship. During his Ph.D., he stayed for 6 months in 2000 at HP Labs in Bristol. He studies network and service management and monitoring, and has coauthored more than 100 scientific papers on topics related to this field.