applied sciences

MDPI

Article

No Pictures, Please: Using eXplainable Artificial Intelligence to Demystify CNNs for Encrypted Network Packet Classification

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

Department of Electronic and Communication Technologies, School of Engineering, Universidad Autónoma de Madrid, 28049 Madrid, Spain; daniel.perdices@uam.es (D.P.)
* Correspondence: jorge.lopez_vergara@uam.es; Tel.: +34-91-497-22-59

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 correct traffic policing and planning allow for 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 detects 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 and 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

check for updates

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**, *14*, 5466. https://doi.org/10.3390/app14135466

Academic Editors: Mirka Saarela, Lilia Georgieva and Vili Podgorelec

Received: 3 May 2024
Revised: 14 June 2024
Accepted: 19 June 2024
Published: 24 June 2024

Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

With the rise in 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 the 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 performed offline, i.e., after the network flow has already been completed, but rather online, i.e., while the first packets in 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, we saw the rudimentary use

of port-based classification [1] (with a basis on TCP and UDP well-known port numbers); later, we saw the use of Deep Packet Inspection (DPI) using pattern matching and detecting protocol anomalies [2]; and, most recently, algorithms based on statistical features, machine learning-based techniques such as Support Vector Machine (SVM) [3], and 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 reportedly 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 the capabilities of CNNs in depth 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 were 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 needed to make sure that the separation between packets in the train and validation sets prevented data leakage. That is, we ensured that we did not train and test with different packets in 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 conducting network packet processing, because packets in the same connection can cause data leakage if they are incorrectly split. Secondly, we wanted 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 models to the same standards of privacy, unbiasedness, and reliability as other standards in the industry. Once we obtained the right information, we built a reference CNN model for network traffic classification. From the aforementioned dataset, we filtered the headers of the packets in order to manipulate just the encrypted payload data and avoid using IP addresses and ports. After training, we analyzed the performance and the explainability. Regarding explainability, we needed to study how to extract data directly from a deep neural network. For this, we 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 focused on understanding important factors for model performance. First, we analyzed characteristics of the data that might be relevant. A good example of this is that data leakage leads to overfitting when packets in the same flow are in both the training and test set. We also analyzed the different parameters of the model that impact performance. Finally, the decision was analyzed using eXplainable Artificial Intelligence (XAI) techniques, so that we could try to understand how the features were built and what they focused on.

The rest of this paper is structured as follows. Next, Section 2 shows the related work, in order to contextualize our research. Then, Section 3 explains how the available datasets were 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. Section 6 applies this technique to real data, so we can understand what the neural network generalizes from the encrypted traffic it trains 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-nineteen-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 in interpreting deep learning models prevent their use in real scenarios as the need for accountability, responsibility, and transparency in 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, CNNs, 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 in real scenarios due to a lack of generalization and a misjudgment of the true capabilities of the model. Additionally, these techniques are sometimes applied to non-encrypted packets, which are not representative of the current 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.

The authors of [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 studies have 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 GPUs (Graphics Processing Units) 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 the authors of [17] noticed how flow characteristics might affect their network classifier. Specifically, their method utilizes flow information to contextualize the network traffic. By incorporating such information, they were 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.

Table 1. Review of recent publications on traffic classification. 'Online' column identifies if the works process network traffic on the fly ('✓') or not ('×'). 'No Flow Data Leakage' column identifies if the handling of bias and data leakage is adequate ('✓') or not ('×'), or if it is unknown due to lack of information in the paper ('?').

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 and nonVPN	Heartbleed attacks	✓	?
[21]	2022	1D-CNN	MIRAGE COVID CCMA 2022	Mobile apps	×	✓
[22]	2022	1D-CNN, DBN	ISCX VPN and nonVPN	VPN vs. nonVPN and Services	✓	×
[23]	2022	FDCNN	ISCX VPN and nonVPN	VPN vs. nonVPN and Services	✓	×
[12]	2021	2D-CNN	ISCX nonVPN	Applications	✓	?
[17]	2021	2D-CNN	ISCX VPN and nonVPN	Both	×	?
[24]	2021	GADCN	USTC TFC2016	Applications	×	?
[10]	2020	1D-CNN+tSNE	Own	Services	×	?
[25]	2020	SAE & 1D-CNN	ISCX VPN and nonVPN	Applications	✓	?
[26]	2020	CNN, LSTM, SAE, HAN	WIDE, USTC	Applications	×	?
[27]	2019	1D-CNN	ISCX VPN and nonVPN, ISCX 2012 IDS	Services and Malware	×	?

Table 1. *Cont.*

Paper	Year	Architecture	Dataset	Output	Online	No Flow Data Leakage
[28]	2019	1D-CNN, 2D-CNN, 3D-CNN	ISCX VPN and 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 and nonVPN	VPN vs. nonVPN and Services	✓	?
[33]	2016	k-NN	Own (VPN)	Applications	×	?

This 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, this review assesses 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 examines 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 studied the authors' handling of bias and data leakage. We observed that several studies report highly accurate performances 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 the train–test split was conducted ('?') or give hints that data leakage might have happened ('×'). Additionally, in many cases, it is not clear how the packet-to-image conversion was performed, with no information provided on whether headers were eliminated or if images were 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.

In the articles we reviewed, a wide variety of open datasets were 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 were observed [10,26,29,30]. The aforementioned datasets contain network capture 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 was 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 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 of the studied articles, such as [17–19], utilize convolutional neural networks to extract information from statistics generated with various packets in 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 method for classification, but it also has several disadvantages. The first challenge is the difficulty of classifying the first packets in 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 for the 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 perform 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 preprocessing. 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 preprocessing was handled and the processes, decisions, and developments that were 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 into 152 capture files. It is grouped in 42 labels, depending on the application. For our study, we 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 make this decision because there is no need to look at the Over-The-Top (OTT) service provider to apply fair QoS policies.

As a multiclass classification problem, always choosing the same class in a balanced-class 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 step we will perform 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 performed to prevent the neural network from training with packets that can be classified by well-known ports in a real application or with DPI techniques. Then, we extract from each traffic capture the individual packets and convert them to

binary files. In this step, we 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, destination port, transport protocol). This does not mean that statistics are extracted, simply that the packets in the same flow are 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 in the same application (see, for instance, Figure 1). Thus, the objective is to train and test with packets in different flows in order to avoid overfitting and data leakage. Next, these individual packets are converted into images so that they can be used to train the neural network. Each byte of payload information is converted to a pixel, and then the pixels are arranged according to the desired dimension.

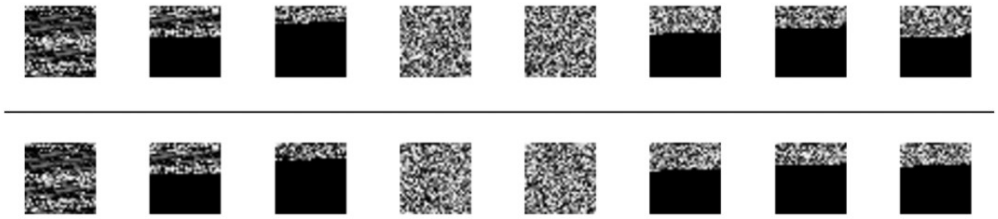


Figure 1. Payload images of the packets in two different Chat flows.

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 can use different QoS policies to improve the experience in the network. With this process, we achieved the fact that, from each packet, an application label was extracted individually, without the need to load memories of 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 can 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 2, 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 performed using Tshark, available in the GitHub repository link attached to this paper (https://github.com/elbisbe/XAI_DL_NetworkPacketClassification accessed on 14 March 2023). 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 overfitting. 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 are filtered, the payload binary data have 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 helps to normalize the dataset and can 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 is to fill in the rest of the packet with zeros (zero padding); the second is to fill it in with random values (random padding) [12]. We used the first alternative, as the intention of the model is to only alter the packets that need 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) were taken out of the dataset, as they would be pitch-black images. This packet-to-image process can be performed for grayscale one-dimensional (1D) or one-dimensional (2D) structures [38], although in other studies, so-called three-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 the performance, both in terms of the accuracy of the model and in terms of the throughput of the system. It is expected that a neural network can take advantage of image acceleration hardware architectures to process the 2D and 3D images in parallel. 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 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 the 1D-CNN achieves the highest precision and F1-score, which is consistent with previous findings. Interestingly, the 3D-CNN closely matches the performance of the 1D-CNN and even surpasses that of the 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.

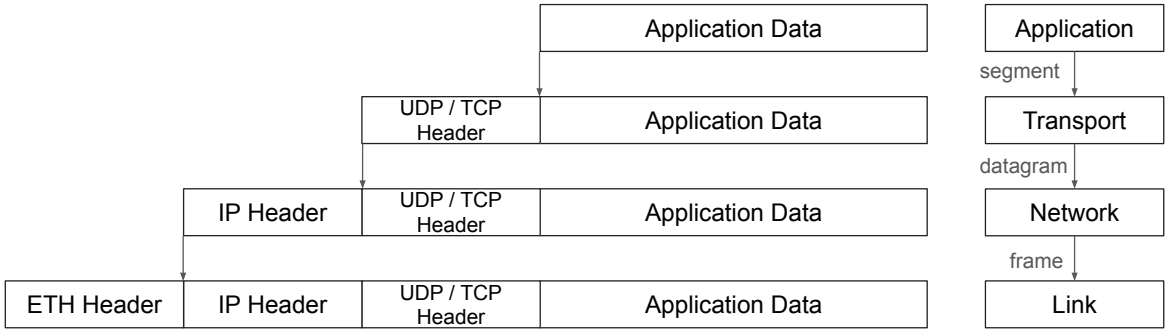


Figure 2. Protocol headers.

Regarding the size of the packet, it could be considered that a larger image size will result in better accuracy results, due to the increase in information about each packet that is provided to the neural network. In our preliminary results, we observed that increasing the image size from 28 $\times$ 28 pixels to 32 $\times$ 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 $\times$ 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 was used, i.e., square images of 32 × 32 pixels, so that we did not exceed the typical MTU of 1500 bytes. We empirically chose 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 in 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 3, a folder structure with three 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 images did not show substantial improvements. These images are stored in the 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 limits how many packets are used per traffic flow, so that there are no large differences in the number of packets in each flow. Similarly, the work in [29] developed traffic classifiers with high accuracy percentages that used only the first 20 packets in each flow. However, this may introduce bias in the training, since connection closure will not be observed in all flows.

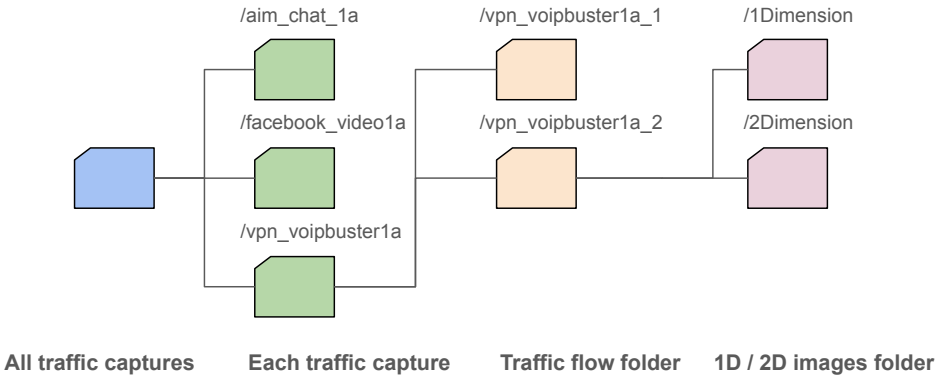


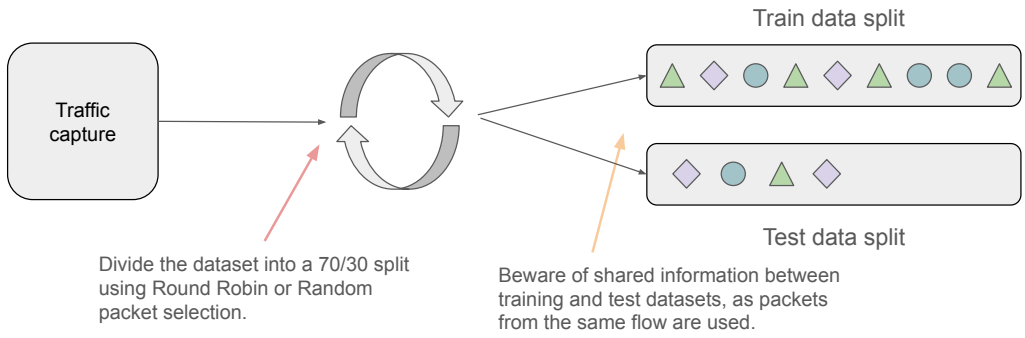
Figure 3. Proposed folder structure for traffic classification.

Figure 4a 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 4b 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 refer to this phenomenon as 'flow dependence'.

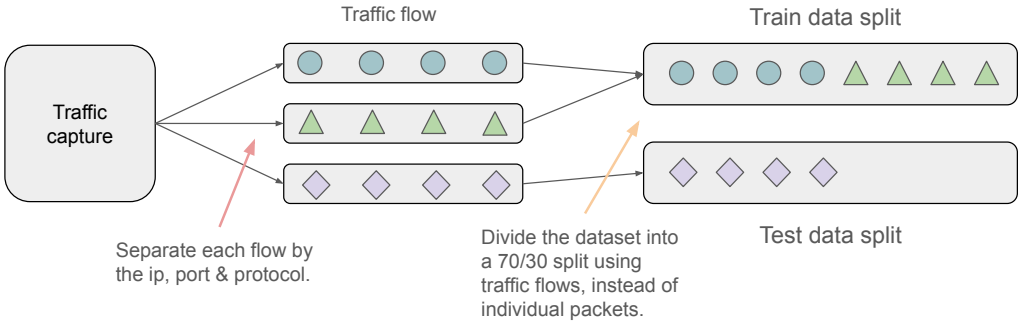
With these considerations, we can perform a desired data-leakage-free train–test split. Suppose we want to divide the dataset between 70% training files and 30% test files. In a first approach, the first 70% of the flows in the trace are assigned to the training dataset, and the last 30% are assigned 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 overfitting would be introduced. An alternative is to perform a round-robin system between the flows.

In this way, the first seven flows would be for training, the next three for testing, then seven again for training, etc. This method corrects the possible overfitting introduced due to the order of the flows. The last proposal consists of randomly choosing 70% of the flows in the training set and 30% in the test set. This method is very difficult to debug and check for errors. On the other hand, there are no problems with the order of flows and their relation and overfitting.

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 preprocessing 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 the neural network is 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 4. Comparison between both methods for dataset split.

4. Model Definition

After reviewing all the publications in Section 2, we came up with a general schema of a convolutional architecture, depicted in Figure 5. It is divided into 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 three or five, 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 in 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 two or three on each axis by summarizing the information of squared patches of pixels into a single pixel that represents the maximum of the patch. One can consider 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 5) until the image is small enough to have captured few but interesting features.

2. **Feature analysis:** Once we extract 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 performing this, we obtain 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.

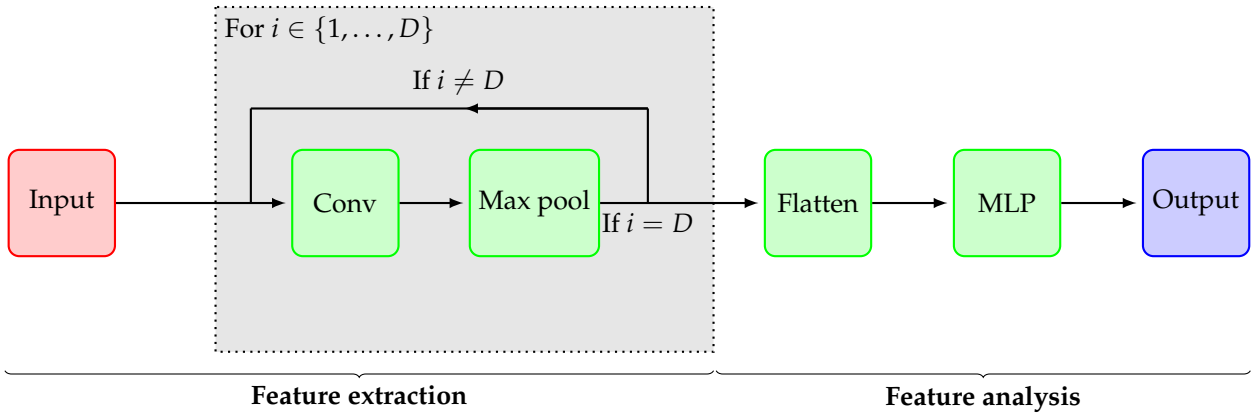


Figure 5. General CNN architecture with D convolutional stages.

With this architecture, we 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 mostly benefit the model we were trying to optimize. In particular, 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 complete feature extraction. All activation functions, except the one at the output, are ReLU, given its good performance and well-known characteristics to avoid any vanishing gradient issues. Since the size of the image is divided by two in each convolutional stage, we multiply 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 $\times$ 32 pixels and the output of the feature extraction is 64 images of sizes 6 $\times$ 6 (or 8 $\times$ 8 if we had considered a convolution layer with border padding).

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

Table 2. *Cont.*

Common Parameters	
2D model	
Max-Pool Size	2×2
Number of Conv2D stages	4
Number of fully connected stages	4
Number of LSTM stages (when used)	2

5. Gradient Analysis

As we have seen, proper separation in flows is needed in order to avoid overfitting or data leakage and reach 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 obtain an explanation of the behavior of the models. For this purpose, we applied GradCAM, known as gradient-weighted class activation mapping. This XAI algorithm was presented initially by [39] and used for image-based classification, captioning, and a 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 structured output, or with multi-modal inputs. GradCAM is a generalization of CAM for CNN architectures [41].

The main concept behind GradCAM is deeper representation on a CNN capture higher-level visual construct; 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 6.

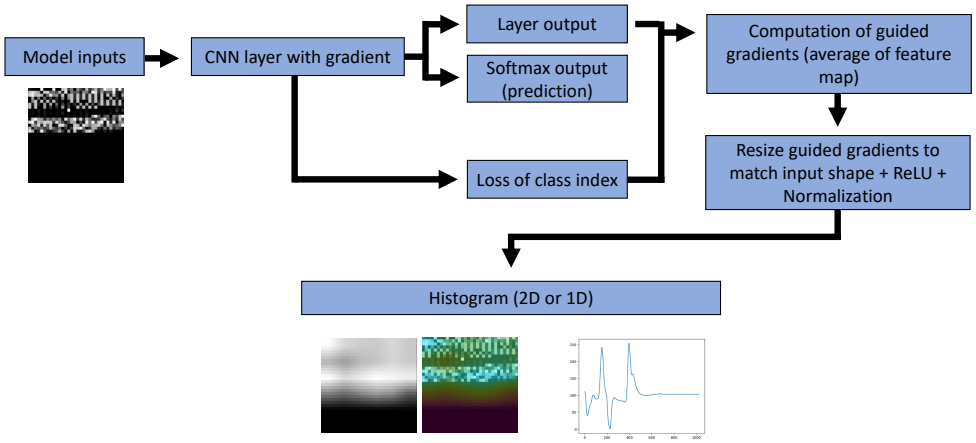


Figure 6. Histogram computation diagram in GradCAM.

To obtain the class activation map or heatmap, we first need to find the CNN layer in our model that we want to extract the gradient from. In Figure 5, this is the convolutional layer when $i = D$. As a result, we obtain (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 retained information of the gradients of y with respect to A computed with backpropagation, i.e., $\frac{\partial y}{\partial A}$. These gradients were used together with A to compute a heatmap for each class. Finally, the heatmaps were 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 performed 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 they are seeing to determine that a possible packet does not belong to a category. Merging both results, we obtain the two most important features the model considers for the prediction, positive and negative, as shown in Figure 7.

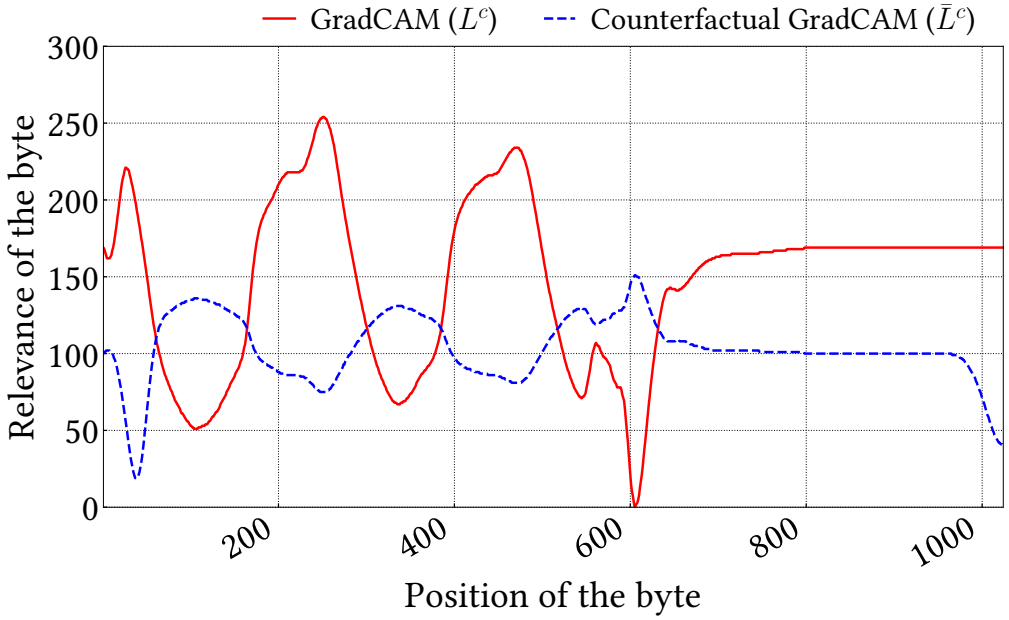


Figure 7. Relevance values given by GradCAM-1D when using both standard and counterfactual analysis.

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 using guided gradients, i.e., for the i-th element of the tensor

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

By using the first, we only consider the gradients in every part of A^k, while in the second approach we only take into consideration elements whose value is positive, i.e., they would

trigger the ReLU activation, and whose gradient is also positive, i.e., contributes 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, that 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 performed 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:

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

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 hint not to choose class c.

Another consideration that must be made is that on multiclass 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 widely 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 the fact that RNNs and 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 the explanations are for the decisions of the models.

6. Results

In this section, we present the results of our experiments grouped into 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 significantly alter the performance. Third, the same process is repeated for the model and its parameters. Last, we apply XAI to understand the decisions of the model.

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 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 sets 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 in the same flow can only belong to either the training or the test set.

Consequently, in Table 3 we present 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 covered in the next subsection, are due to the flow dependence. Furthermore, Figure 8 displays the confusion matrix for the 1D model after some data preprocessing. This shows that the error is mostly focused on the Audio class, and is confused with E-mail and Video classes.

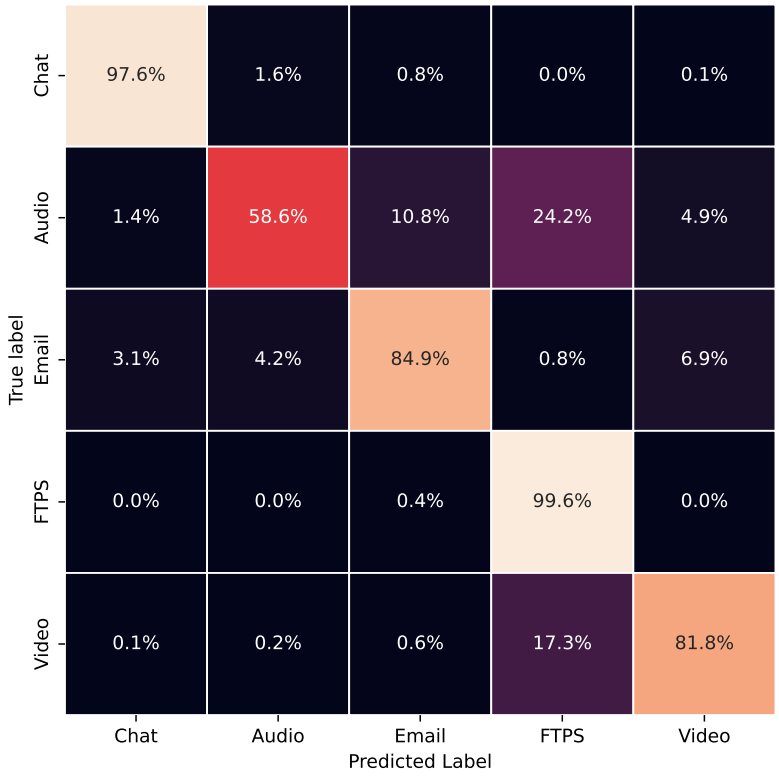


Figure 8. Final confusion matrix model using 1D input images and separated client–server flow direction.

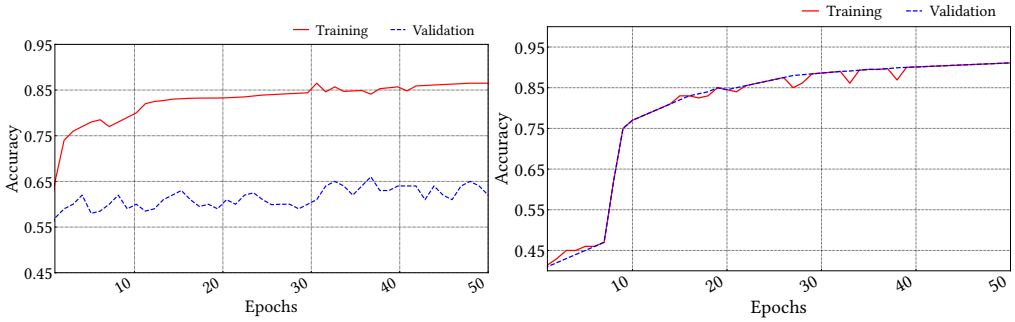
Table 3. Global metrics of the proposed models.

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

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 Figure 9. Using a validation set of random packets, we managed to observe that both the training and validation curves overlap, showing an accuracy above 90%. On a first view, this would mean that the model is not overfitting, and the performance is optimal.



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

Figure 9. Training and validation of the base model with dependent and independent flows.

On the other hand, when training and validating the system with independent flows, the performance changed drastically. While the accuracy in training was about 90%, independent flows in the validation set showed that the real number was closer to 65%. That is, the model is not learning to generalize the specific per-class characteristics of the packets, but is 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.

Second, we analyzed the difference depending on the direction of the flow. Figure 10 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 the client and server in this case, uplink relevance is focused on the first bytes of the packets, where requests should be the most relevant part, while downlink relevance is more relevant 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 studied the impact of padding, that is, filling with extra data when the packet does not reach the desired size of 1024. This can be performed 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 to add data to maintain the entropy as they 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 can intuitively help identify traffic classes, since we expect that larger packets should belong to data

transfers and smaller packets should belong to other delay-sensitive services. Therefore, we only considered zero padding, as shown in Figure 1; the bottom of the 2D images is filled with black pixels.

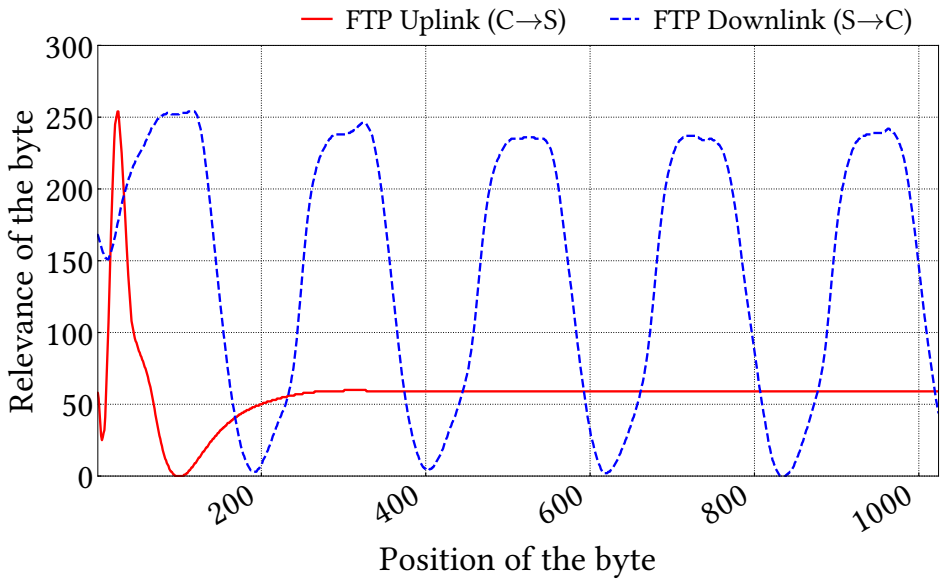


Figure 10. GradCAM histogram of FTP client vs. server.

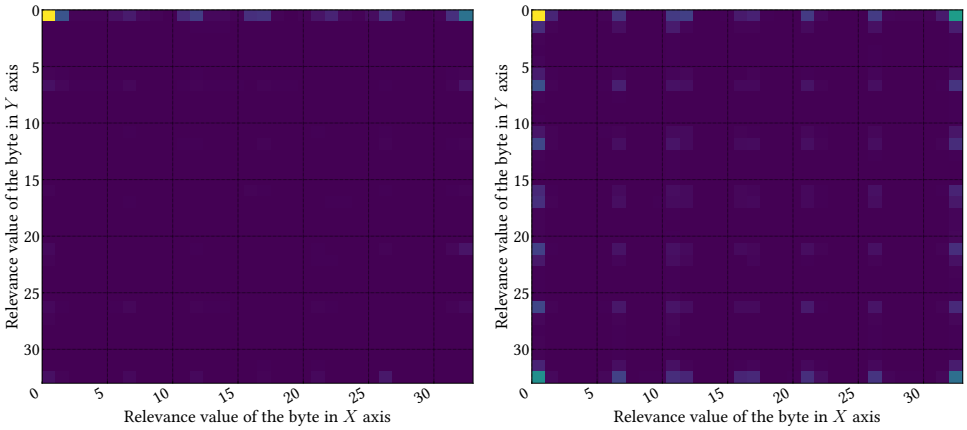
6.3. Understanding the Model

With all the previous observations about the data, we are now able to understand better what the model might be doing. At this point, we studied what are the difference is 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 by adding an extra relation to 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 happens 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] 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 conducted 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 show that packet size is a relevant metric when

analyzing the performance of classifiers made with CNNs [12]. Upon testing different widths, we observed a clear softening 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 gave a clear outline identified for that packet's position in the flow.



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

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

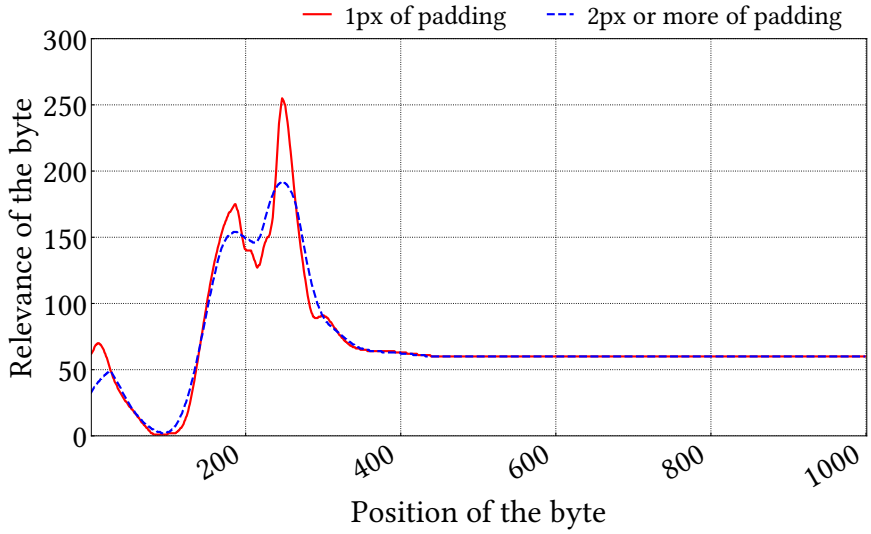
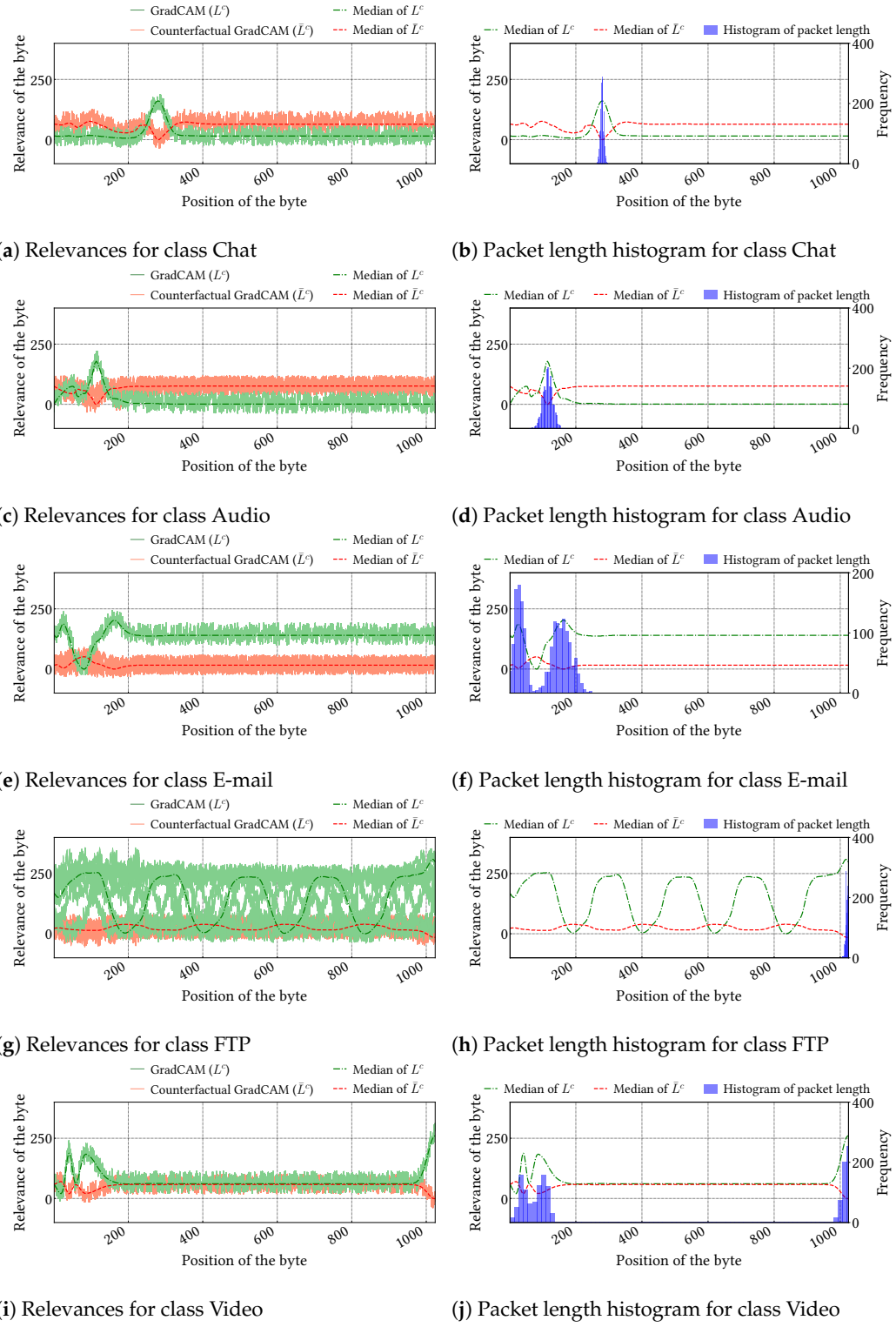


Figure 12. Difference between the use of 1-pixel padding vs. 2 pixels or more in GradCAM.

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 on the left-hand side of Figure 13. All of them but Figure 13g display similar behavior: they are smooth functions with some random noise. In order to model this, we provide a median function to act as the centroid of the groups that, by nature, should be less noisy. The 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.



(**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.

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 at 300 bytes, where L^c increases and surpasses $\bar{L}^c$. As shown in Figure 1, 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 not random, but the mode of the packet length. This means that the model is mostly focusing on the last bytes of content in the packet. Due to the convolutional nature of the CNNs, they are powerful

when detecting 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 used zero padding to fill the packet images, so the CNN was able to detect this abrupt change to a zero-valued region at the end of each packet.

Similarly, Figure 13c exposes the relevance and counterfactual relevance of class Audio, where the same behavior occurred 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 for a small interval centered at 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, the models showing values of L^c spread around all values, with an important maximum only 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 8), 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 confirm 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 only provide information on the size of the payload. With this in mind, our next step is to provide a simpler model that only uses the packet size to check whether a similar performance can be achieved with a higher packet classification rate in packets/s.

For this reason, we built a decision tree that only uses 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 important attribute and that encryption poses quite a challenge in that CNNs are not yet able to solve encryption 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.

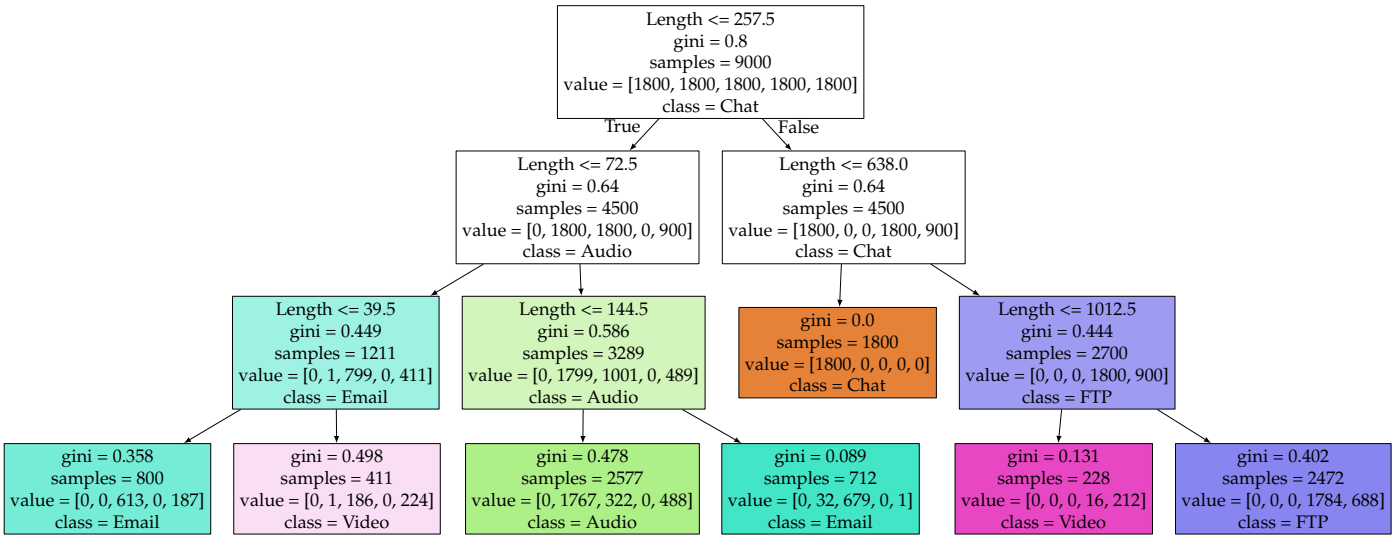


Figure 14. Decision tree for encrypted packet classification.

7. Conclusions

In summary, we have analyzed the key aspects of 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 analyzed the model's predicted traffic class using GradCAM, an 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 only looks at the packet size to make a decision. Consequently, we were able to match the 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 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 a 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, CNNs 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 lengths of padding to the packets in encrypted traffic would be enough to make such classifications unfeasible, at the cost of more payload per packet. This could be a good solution for users' privacy, but it is 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 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 only addressed 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 again poses 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 (https://github.com/elbisbe/XAI_DL_NetworkPacketClassification accessed on 14 March 2023). 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. and V.M.-G.; validation, E.L.-B., V.M.-G. and D.P.; formal analysis, D.P.; investigation, E.L.-B. and V.M.-G.; resources, J.E.L.d.V.; data curation, E.L.-B. and V.M.-G.; writing—original draft preparation, E.L.-B., V.M.-G., D.P. and J.E.L.d.V.; writing—review and editing, E.L.-B., V.M.-G., D.P. and J.E.L.d.V.; visualization, E.L.-B., V.M.-G. and D.P.; 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 was 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 (https://github.com/elbisbe/XAI_DL_NetworkPacketClassification) accessed on 14 March 2023.

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

Abbreviations

The following abbreviations are used in this manuscript:

1D	one-dimensional
2D	two-dimensional
3D	three-dimensional
ACK	Acknowledgment
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. *Int. J. Comput. Sci. Inf. Secur.* **2010**, *7*, 36–45.
2. Bremler-Barr, A.; Harchol, Y.; Hay, D.; Koral, Y. Deep packet inspection as a service. In Proceedings of the 10th ACM International on Conference on Emerging Networking Experiments and Technologies, Sydney, Australia, 2–5 December 2014; pp. 271–282.
3. Yuan, R.; Li, Z.; Guan, X.; Xu, L. An SVM-based machine learning method for accurate internet traffic classification. *Inf. Syst. Front.* **2010**, *12*, 149–156. [CrossRef]
4. Rezaei, S.; Liu, X. Deep learning for encrypted traffic classification: An overview. *IEEE Commun. Mag.* **2019**, *57*, 76–81. [CrossRef]

5. Xin, R.; Zhang, J.; Shao, Y. Complex network classification with convolutional neural network. *Tsinghua Sci. Technol.* **2020**, *25*, 447–457. [CrossRef]
6. Ren, J.; Ou, X.; Zhang, Y.; Hu, D. Research on network-level traffic pattern recognition. In Proceedings of the IEEE 5th International Conference on Intelligent Transportation Systems, Singapore, 6 September 2002; pp. 500–504. [CrossRef]
7. Roshan, K.; Zafar, A. Utilizing XAI Technique to Improve Autoencoder based Model for Computer Network Anomaly Detection with Shapley Additive Explanation (SHAP). *Int. J. Comput. Netw. Commun.* **2021**, *13*, 109–128. [CrossRef]
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 Commun. Mag.* **2022**, *60*, 25–31. [CrossRef]
9. Aceto, G.; Ciuonzo, D.; Montieri, A.; Pescapé, A. Toward effective mobile encrypted traffic classification through deep learning. *Neurocomputing* **2020**, *409*, 306–315. [CrossRef]
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), Toronto, ON, Canada, 6–9 July 2020; pp. 1292–1293. [CrossRef]
11. Meng, Z.; Wang, M.; Bai, J.; Xu, M.; Mao, H.; Hu, H. Interpreting deep learning-based networking systems. In Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication, Virtual Event, 10–14 August 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), Bordeaux, France, 17–21 May 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.; Springer: Cham, Switzerland, 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 Trans. Netw. Serv. Manag.* **2021**, *18*, 4225–4246. [CrossRef]
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 Workshop on Network Meets AI & ML, Virtual Event, 10–14 August 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's Thesis, Máster Universitario en Ingeniería de Telecomunicación, Universidad Autónoma de Madrid, Madrid, Spain, 2021.
17. Shapira, T.; Shavitt, Y. FlowPic: A Generic Representation for Encrypted Traffic Classification and Applications Identification. *IEEE Trans. Netw. Serv. Manag.* **2021**, *18*, 1218–1232. [CrossRef]
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 Trans. Netw. Serv. Manag.* **2024**, *21*, 51–69. [CrossRef]
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), Tehran, Iran, 11–12 May 2022; pp. 1–8. [CrossRef]
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 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS '22, Los Angeles, CA, USA, 7–11 November 2022; pp. 1537–1551. [CrossRef]
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. *Comput. Netw.* **2022**, *219*, 109452. [CrossRef] [PubMed]
22. Izadi, S.; Ahmadi, M.; Rajabzadeh, A. Network Traffic Classification Using Deep Learning Networks and Bayesian Data Fusion. *J. Netw. Syst. Manag.* **2022**, *30*, 25. [CrossRef]
23. Appiah, B.; Sackey, A.K.; Kwabena, O.A.; Kanpogninge, A.J.A.; Buah, P.A. Fusion Dilated CNN for Encrypted Web Traffic Classification. *Int. J. Netw. Secur.* **2022**, *24*, 733–740.
24. Dong, S.; Xia, Y.; Peng, T. Traffic identification model based on generative adversarial deep convolutional network. *Ann. Telecommun.* **2022**, *77*, 573–587. [CrossRef]
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 Comput.* **2020**, *24*, 1999–2012. [CrossRef]
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 SIGCOMM'20 Poster and Demo Sessions, Virtual Event, 10–14 August 2020; pp. 1–2. [CrossRef]
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. [CrossRef]
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), Waikoloa, HI, USA, 9–13 December 2019; pp. 1–6. [CrossRef]
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. [CrossRef]
30. Dong, S.; Li, R. Traffic identification method based on multiple probabilistic neural network model. *Neural Comput. Appl.* **2019**, *31*, 473–487. [CrossRef]

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), Da Nang, Vietnam, 11–13 January 2017; pp. 712–717. [CrossRef]
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), Beijing, China, 22–24 July 2017; pp. 43–48. [CrossRef]
33. Draper-Gil, G.; Lashkari, A.H.; Mamun, M.S.I.; Ghorbani, A.A. Characterization of Encrypted and VPN Traffic Using Time-related Features. In Proceedings of the 2nd International Conference on Information Systems Security and Privacy—ICISSP, INSTICC, SciTePress, Rome, Italy, 19–21 February 2016; pp. 407–414. [CrossRef]
34. Shiravi, A.; Shiravi, H.; Tavallaee, M.; Ghorbani, A.A. Toward developing a systematic approach to generate benchmark datasets for intrusion detection. *Comput. Secur.* **2012**, *31*, 357–374. [CrossRef]
35. 3GPP. Policy and Charging Control Architecture, 2021. TS 23.203, Rel.17. Available online: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=810 (accessed on 1 May 2024).
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. *Appl. Sci.* **2019**, *9*, 3414. [CrossRef]
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 2012 Internet Measurement Conference, Boston, MA, USA, 14–16 November 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. [CrossRef]
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 IEEE International Conference on Computer Vision, Venice, Italy, 22–29 October 2017; pp. 618–626.
40. Wu, Z.; Shen, C.; Van Den Hengel, A. Wider or deeper: Revisiting the resnet model for visual recognition. *Pattern Recognit.* **2019**, *90*, 119–133. [CrossRef]
41. Rosebrock, A. Grad-CAM: Visualize Class Activation Maps with Keras, TensorFlow, and Deep Learning, 2020. Available online: https://pyimagesearch.com/2020/03/09/grad-cam-visualize-class-activation-maps-with-keras-tensorflow-and-deep-learning/ (accessed on 2 May 2024).
42. Selvaraju, R.R.; Das, A.; Vedantam, R.; Cogswell, M.; Parikh, D.; Batra, D. Grad-CAM: Gradient-Weighted Class Activation Mapping. Available online: http://gradcam.cloudcv.org/ (accessed on 2 May 2024).
43. Madhukar, B. Using Grad-CAM to Visually Verify the Performance of CNN Model. 2020. Available online: https://analyticsindiamag.com/using-grad-cam-to-visually-verify-the-performance-of-cnn-model/ (accessed on 2 May 2024).
44. Singirikonda, M. How Padding Helps in CNN? 2020. Available online: https://mahithas.medium.com/how-padding-helps-in-cnn-2b87957e1b (accessed on 2 May 2024).

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.