THE PENNSYLVANIA STATE UNIVERSITY
SCHREYER HONORS COLLEGE

DEPARTMENT OF INFORMATION SCIENCES AND TECHNOLOGY

Analysis on AI Methods for Detecting DNS Cache Poisoning Attack

BINCHEN FANG

A thesis
submitted in partial fulfillment
of the requirements
for a baccalaureate degree
in Cybersecurity Analytics and Operations
with honors in Cybersecurity Analytics and Operations

Reviewed and approved* by the following:

Peng Liu
Professor & Raymond Tronzo, M.D. Professor of Cybersecurity
Thesis Advisor

Michael Hills
Teaching Professor, Coordinator, B.S. in Security and Risk Analysis
Honors Advisor

* Electronic approvals are on file.

ABSTRACT

The Domain Name System (DNS) plays an important role in the proper functioning of the internet. It can translate human-readable domain names to machine-readable IP addresses. However, it is vulnerable to several types of cyberattacks, with the domain name system cache poisoning being the most harmful one. Traditional detection methods are often unable to identify the nuanced behaviors of cache poisoning, especially with evolving techniques adopted by malicious actors. By using artificial intelligence, filtering malicious domain name system responses becomes much more possible. This paper will present a comprehensive analysis and interpretation of the Artificial Intelligence-based method that is used to detect domain name system cache poisoning attacks. By using different explainable artificial intelligence(XAI) tools to analyze the AI model and understand how different tools explain the interpretation of the AI model differently.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to thank the following people for helping with this research project:

Dr. Peng Liu: I would like to express my deepest appreciation to Dr. Peng Liu for his support throughout the research.

Dr. Qingtian Zou: Special thanks to him for his help and support throughout the research.

The Pennsylvania State University Library and Staff: I want to thank the resources provided by the university library and staff members.

The Pennsylvania State University Graduate Team: I deeply appreciate the helpful assistance provided by the university graduate team members.

My family: I want to thank my family for their support and their understanding throughout this research.

Chapter 1 - Introduction

1.1 Overview

Cybersecurity is becoming more and more important in today's interconnected world, with the exponential growth of technology and a commensurate increase in internet connectivity. The significance of cybersecurity has emerged as a critical focal point for individuals, corporations, and nations. According to the Federal Bureau of Investigation Internet Crime Report[1], at least $10.3 billion was lost to cybercrime in 2022. Compared to 2021, it increased about 49%. With the increasing cost of defending security traditionally, artificial intelligence quickly becomes another choice that provides cost-effective and strong performances. With its capabilities to learn, adapt, and autonomously respond, it holds transformative potential in fortifying cybersecurity frameworks. It started to spread across more and more areas.

The "AI for Cybersecurity: A Handbook of Use Cases[1]" is an authoritative resource presented by the Penn State Cybersecurity Lab. In a world fraught with evolving cyber threats, this handbook demystifies the potent synergy of artificial intelligence with cybersecurity. Through meticulously curated use cases, the handbook bridges the gap between theoretical principles and practical applications of AI in the realm of cybersecurity. Each chapter delves into a distinct challenge, offering both an analytical perspective and actionable insights. Chapter 2 [2], [3] delves into the application of deep learning for reverse engineering tasks. Chapter 3 [4] zeroes in on the detection of Android malware. Chapter 4 [5], [6] focused on the identification of abnormal events within sequential data. Chapter 5 addresses the detection of DNS Cache Poisoning Attacks. In Chapter 6, the spotlight is on detecting malware in personal computers. Chapter 7 [7] embarks on a journey of comparing binary code similarity to uncover known

vulnerabilities. Finally, Chapter 8 [8], [9] showcases the prowess of AI in malware clustering. As a comprehensive guide, this handbook serves as an invaluable compass for those navigating the ever-converging domains of AI and cybersecurity.

As the use of machine learning (ML) models[10], [11] in cybersecurity continues to grow, understanding how ML models operate becomes crucial. When administrators understand the underlying reasons for alerts, they can more confidently take appropriate actions. Creating trust and transparency between the administrators and the model. When an alert is triggered, it's crucial to determine the origin of the threat. With the understanding of the ML model, root cause analysis can be performed to identify the exact vulnerabilities being exploited. With the analysis, administrators can also identify new attack strategies before they become widespread.

1.2 Explainable AI techniques and tools

Explainable AI (XAI) refers to techniques and methodologies to help ML models become more understandable and transparent decision-making models. AI models, especially deep learning, often act as "black boxes" with decision-making processes. This is very difficult for humans to interpret. XAI aims to bridge this gap, ensuring that AI systems are not only accurate but also comprehensible and accountable.

Explainable AI (XAI) tools serve as an essential bridge between complex ML models and human comprehension. Their main objective is to demystify the underlying processes of AI models, enabling users to grasp, trust, and manage AI-driven outputs. This transparency becomes

especially indispensable in sectors where the ramifications of AI decisions are monumental, and any misjudgment can entail significant repercussions.

Among the diverse array of XAI techniques, feature importance stands out as a foundational one. This technique revolves around gauging the weightage or importance of individual input variables in relation to the model's predictions. For instance, in a model predicting house prices, feature importance might indicate whether the number of bedrooms or proximity to a school influences the price more significantly. Tools like SHAP [12] (SHapley Additive exPlanations) and LIME [13] (Local Interpretable Model-agnostic Explanations) are at the forefront of this domain. SHAP derives its methodology from cooperative game theory, allocating an importance value to each feature based on its contribution to every possible prediction outcome. On the other hand, LIME builds simple interpretable models around individual predictions of any complex model to discern which features were pivotal for that specific decision.

Visualization tools within TensorFlow's library play a crucial role in the realm of XAI as well. These tools transform the multifaceted computations and architectures of deep learning models into intuitive visual representations. For instance, TensorBoard [14] allows users to visually monitor training metrics, observe model architectures, and probe into layer activations. This not only aids in understanding the model's behavior but is also invaluable for debugging and optimization purposes.

Lastly, the technique of rule extraction, exemplified by tools like Trustee, aims to distill the intricate logic of machine learning models into human-understandable rules. By leveraging simpler, interpretable models like decision trees, Trustee encapsulates the essence of a neural network's decision-making mechanism, translating it into a set of clear-cut rules and conditions.

This provides users with a straightforward roadmap of how inputs are processed and decisions are made.

In conclusion, as AI's role in decision-making processes continues to expand, the importance of tools and techniques that offer clarity to these decisions grows in tandem. XAI tools ensure that AI remains a tool for empowerment, elucidation, and enhancement rather than an inscrutable black box.

1.3 Research Questions

This research delves into an in-depth analysis of trained ML models, seeking to comprehend their interpretations using various XAI analysis tools. Examines the distinctions among these tools and elucidates their capabilities in explaining how each model identifies threats. The research questions are:

1. How do different tools explain the interpretation of the DNS attack detection model differently?
2. Which tool is the most suitable for analyzing DNS attack detection models?

1.4 Overview of DNS Spoofing Attack

A Domain Name System (DNS) Spoofing attack, DNS Cache Poisoning, or DNS Hijacking[15] is an attack where the attack tries to manipulate DNS records and redirect users toward a malicious IP. In the attack, malicious actors forge DNS responses, tricking DNS

resolvers into associating legitimate domain names with malicious IP addresses. A successful DNS spoofing attack will cause the user in that local area network to get redirected to the malicious website rather than the accrual domain they have typed in[16]. In combination with certificate spoofing or other attacks, it is very hard for individuals to identify real and fake websites. The DNS spoofing often proceeds with other cyber attacks like phishing, malware distribution, and data interception. This commonly results in users leaking sensitive data and causing financial loss. That is why preventing DNS spoofing is critical to reducing cybercrime.

Chapter 2 - Research Project Recreation

2.1 Overview of the research

The research that has been recreated is “AI for Cybersecurity: A Handbook of Use Cases” [1]. This handbook discusses the use of deep learning (DL) and reinforcement learning (RL) to resolve cybersecurity issues. The existing literature on this integration is considered to have limited practical utility for professionals enrolled in AI for the Cybersecurity area. The researchers present a handbook outlining various use cases to bridge this gap, such as utilizing AI for reverse engineering tasks and malware detection.

The handbook has seven different sections, each with a different task that uses a different AI model to resolve, which include Android Malware, Abnormal Events, PC Malware, etc. This recreation will focus on chapter 5 of the handbook. In Chapter 5, the topic focuses on using deep learning to detect DNS cache poisoning. With a well-documented paper[1] and GitHub page[17], everything that is required for the recreation is available to access

2.2 Research Recreation

In Figure 2-1, there is a very clear essay file structure provided by the handbook and GitHub page. This includes the Python script, the data sample used in the study, and a few pre-trained models. The README.md file also includes detailed instructions for using each Python script.

Figure **2-1**: File structure of handbook chapter 5.

After installation and setup of the basic environment with Anaconda, the following instructions/process was executed:

2.2.1: Run “chop-by-session.py”:

The script uses the “pyshark” library to read network capture data saved in “.pcapng” format. Then, packet data and use are handled by the “NumPy” library and reformated with NumPy array operations. The script has the following two functions listed below.

The “process-benign” function will read each packet provided by the “benign-dns_split.pcapng” files. It will extract bytes from index 14 to 25 and bytes from index 34 to 53 data from the packets if they have an IP source of 192.168.100.128 and a destination IP of 192.168.100.50. The data extracted will be appended to another list and saved as a “benign_bytes.npy” file.

The “process_malicious” function is very similar to the “process-benign” function. It will read each packet provided by the “malicious-dns_split.pcapng” files. It will extract bytes from index 14 to 25 and bytes from index 34 to 53 data from the packets and store them in a list. These 32 bytes of data are chosen to avoid privacy and noise. Also, these 32 bytes of data that were interested in are labeled in Table 2-1. When processing a packet that has an IP source of 192.168.100.18 and a destination IP of 192.168.100.50, it creates a new list and saves the following data on the new list. The data extracted will be appended to another list and saved as a “malicious_bytes.npy” file.

Layer	Fields	Size in bit	Explanation
IP	Version	4	For IPv4, this is always equal to 4.
	IHL	4	Internet header length.
	DSF	8	Differentiated service field, which includes differentiated services code point and explicit congestion notification.
	Len	16	The entire packet size.
	ID	16	An identification field that is primarily used for uniquely identifying the group of fragments of a single IP datagram.
	Flags	3	3 bits for controlling or identifying fragments.
	FragOff	13	13 bits for specifying the offset of a particular fragment relative to the beginning
	TTL	8	Time to live field, which limits a datagram’s lifetime.
	port	8	This field defines the protocol used in the data portion of the IP datagram.
	checksum	16	Header checksum for error-checking of the header.
UDP	src_port	16	Source port number.
	dst_port	16	Destination port number.
	hd_len	16	The length in bytes of the UDP header and UDP data.
	chksum	16	Checksum field for error-checking of the UDP header and UDP data.
DNS	TID	16	Transaction ID for synchronization between DNS servers/clients.
	flags	16	Control flags
	q	16	The number of entries in the question section.

	AnRR	16	The number of resource records in the answer section.
	AuRR	16	The number of resource records in the authoritative section.
	AdRR	16	The number of resource records in the additional section.

Table **2-1**: Fields of interest.

2.2.2: Run “process-to-datasets.py”:

This script performs preprocessing on network packet data provided by the last script to prepare for a machine-learning model. This script uses multi-processing to parallelize the computation. It can be modified by the --cpu-num for a number of CPU cores for parallel execution.

The main function will initialize a multiprocessing pool with the specified number of CPU cores, load two NumPy files saved from the last script, and define different window sizes and window steps. If the window size is 4, each window will capture four-packet data as one input dataset. If the window step is 1, each time, the window will be one packet down. For example, I have a data list of [0,1,2,3,4,5,6]. With a window size of 4 and a window step of 1, that dataset will be captured and formatted as [0,1,2,3], [1,2,3,4], [2,3,4,5], [3,4,5,6]. Then, for each window size and window step, call a

"thread_function." This function will remove duplicates, balance the dataset, split the data into training and test sets, and save them into a segmented NumPy file. During the process, the malicious data sample was labeled as [0, 1], and the normal data sample was labeled as [1, 0].

2.2.3: Run "train-classifiers.py":

his Python script is intended to create, train, and evaluate convolutional neural network (CNN) models. The library used in this training was TensorFlow. The script used different segments of the NumPy file to train different models. During the process, a text file of the evaluation of each model and the trained models will be saved in the "model" directory.

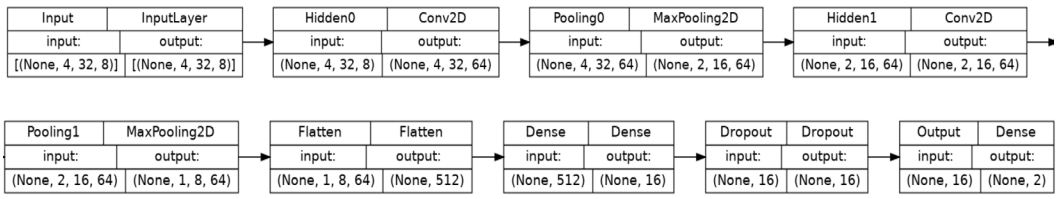

Figure **2-2**: CNN structure and layers for the first model.

Figure 2-2 shows the layer and design of the first model output. The model is designed with nine different layers order: Input Layer, Convolutional Layer, Max Pooling Layer, Convolutional Layer, Max Pooling Layer, Flatten Layer, Dense Layer, Dropout Layer, and Output Layer. The training process will have an epoch of 10 and a batch size of 50. Other parameters will be tuned for model training. This includes window sizes, window steps, hidden tensors, kernel sizes, and dropout rates. So, there is a total of 2250 combinations of tunable parameters that are tried. In total, about 9000 models will be trained. For each model, there is also a 4-fold cross-validation evaluation, and the result (F1 score, FP, FN, etc.) will be recorded in the log file.

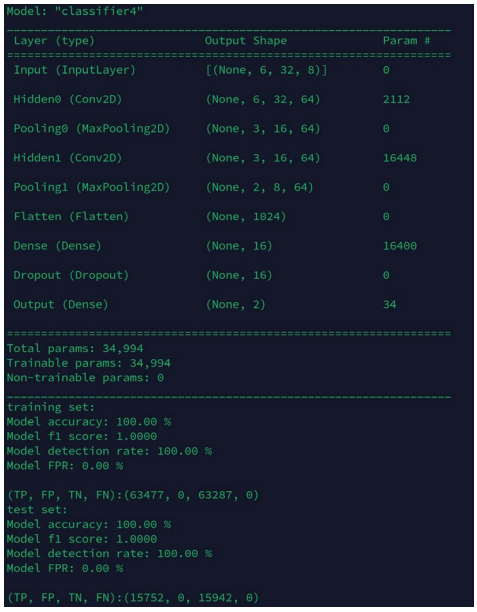

Figure **2-3**: Result of the model provided.

2.2.4: Run “load-model.py”:

This Python script is used for evaluating each trained model. It will load specified models (.h5) in the “model” directory, read needed data sets (.npy) in the “data” directory, and evaluate the specified model. The model selection can be done by adding arguments –n and –k when running the script. The result will be a summary of all four models trained on

different folds. Figure 2-3 shows an example output of the model that was provided by the GitHub Repository[17].

2.2.5: Run “run-time-optimization.py”

This Python script focuses on optimizing the specified models (.h5) in the “model” directory. The model selection can be done by adding arguments –n and –k when running the script. After optimization, the model will be output as a “.tflite” file. Comparing the models (.h5) and models (.tflite). The models (.tflite) have a better performance overall with reduced size and low latency.

2.2.6: Run “load-optimized-model.py”

This Python script is similar to the “load-model.py” file. It will load specified models (.tflite) in the “model” directory, read needed data sets (.npy) in the “data” directory, and evaluate the specified model. The model selection can be done by adding arguments –n and –k when running the script. Figure 2-4 shows an example output of the optimized model.

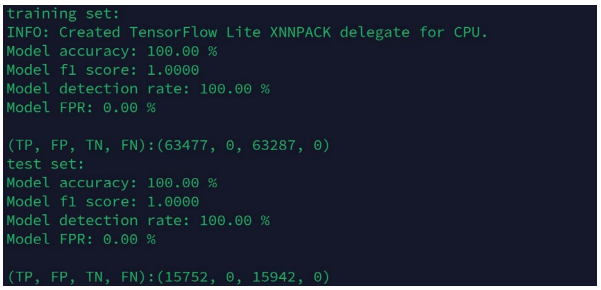

Figure **2-4**: Result of the optimized model.

2.3 Conclusion

In retracing the steps of the original thesis on “Deep Learning Model on DNS Cache Poisoning Attack,” this recreation has served not only as a validation process but also as an avenue to deeply understand the model training process. Through this exercise, the profound capabilities of deep learning in detecting and mitigating DNS Cache Poisoning attacks were reaffirmed. This recreation emphasizes the pivotal role of deep learning in enhancing our digital defense mechanisms against such sophisticated threats. It can serve as a pivotal reference point for striving to integrate advanced artificial intelligence techniques to enhance digital security.

However, of the 9000 models trained, most of the trained have an F1 score of 0.99, which is not a very good sign for a model. High accuracy doesn’t always indicate a well-performing model, especially with the limited amount of data set and the source of the data set. The goal of this research is to study different XAI tools. Learn how each of them behaves and their differences. Improving the model’s training process is not a part of this specific research.

Chapter 3 - Model Structure Analysis

3.1 Layer Explanation

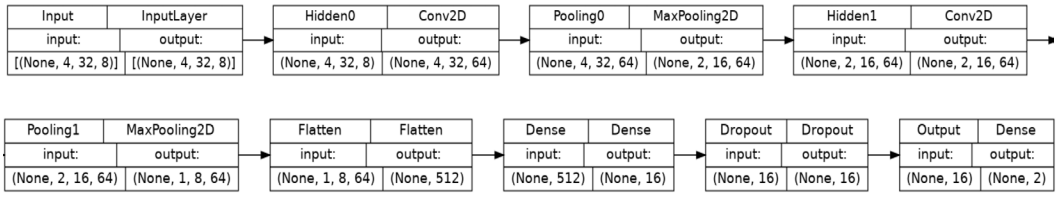

Figure 3-1: CNN structure and layers.

The model was trained using TensorFlow[18]. The first XAI tool used is TensorFlow's library. There are a total of nine different layers in the model. In this analysis, the model was trained using the code provided in Chapter 5 of the AI for Cybersecurity handbook[1]. The model analysis in this section will have a windows size of 4, windows steps of 1, epoch of 10, batch Size of 50, hidden tensors of 64, 64, 16, kernel sizes of 2x2,2x2, and a dropout rate of 5%.

As delineated in Chapter 2 of the thesis, the model boasts a structure composed of nine distinct layers. A visual breakdown of these layers, including their respective inputs and outputs, can be viewed. For a detailed understanding, an analysis using 24 data samples was conducted for each layer. The corresponding code utilized for this layer analysis can be found in Appendix A.

First layer:

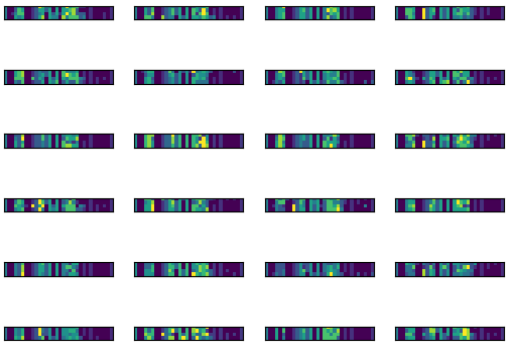

Figure 3-2: Visualize the input data set

The initial layer serves as the input layer. The dataset, represented by X_train, possesses a four-dimensional shape with dimensions (193072, 4, 32, 8). Figure 3-2 visually displays the first 24 data entries derived from the X_train-000.npy dataset. Within this figure, every depicted image spans a size of 4x32, where each individual box is a summary derived from 8 distinct data values.

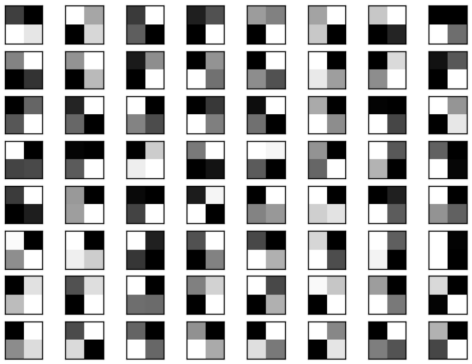

Figure 3-3: 64 filter used in the first convolutional layer

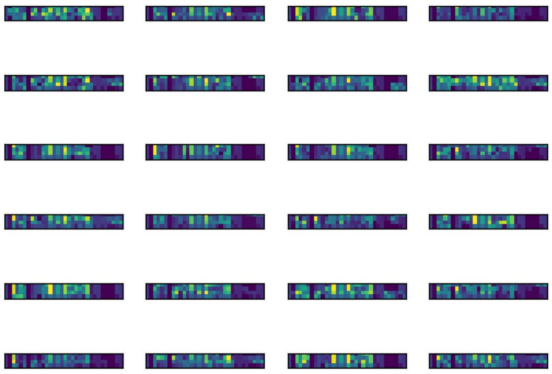

Figure 3-4: The output of the second layer

The Second Layer:

The subsequent layer is a convolutional layer, which stands as a cornerstone in the architecture of Convolutional Neural Networks. Within this layer, the model employs diverse filters to find patterns in the dataset. As depicted in Figure 3-3, the layer utilizes 64 filters, each

size 2x2, as its convolutional kernels. Figure 3-4 provides a visual representation of the resultant output. Consequently, each square in the figure captures a summary of 64 data values, each generated using one of the 64 specified kernels.

The Third Layer:

The third layer is a max pooling layer, primarily designed to reduce the spatial dimensions of the data. By doing so, the network becomes more resilient to minor variations, ensuring it accentuates the most salient features. With a pooling dimension of 2x2, the data undergoes a transformation, resizing from 4x32x64 to a more compact 2x16x64. The visual outcome from this transformation is illustrated in Figure 3-5.

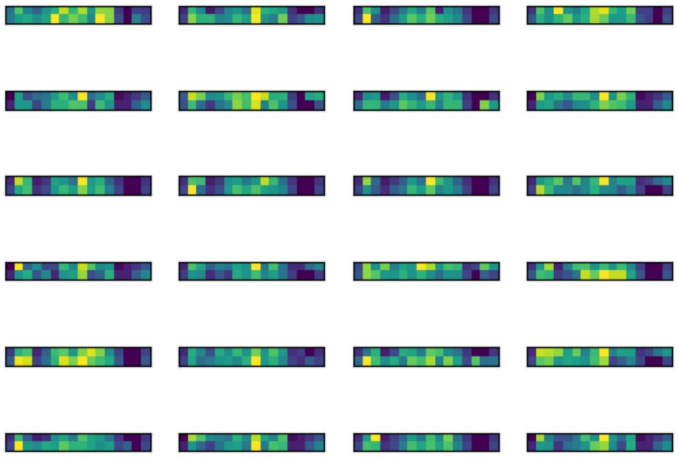

Figure 3-5: The output of the third layer.

The Fourth Layer:

The fourth layer introduces yet another convolutional phase. Within this layer, 64 distinct filters, as showcased in Figure 3-6, are employed as kernels. This added convolutional layer delves deeper into pattern identification. A visualization of the results from this layer can be observed in Figure 3-7.

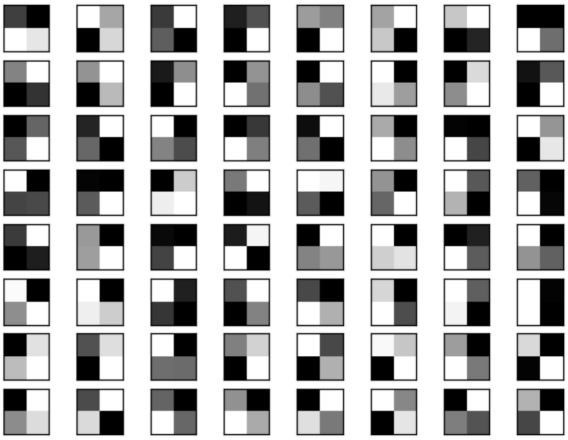

Figure 3-6: 64 filter what was used in the second convolutional layer.

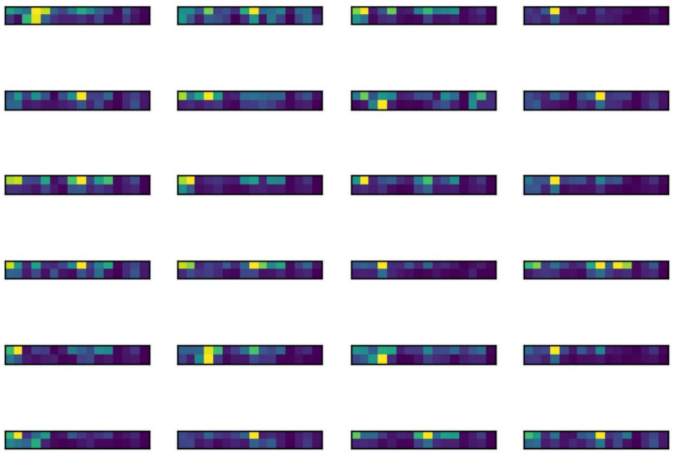

Figure 3-7: The output of the fourth layer

The Fifth layer:

The fourth layer represents an additional max pooling stage. Mirroring the specifications of the previous max pooling layer, this one also utilizes a pooling size of 2x2. Subsequent to this layer's processing, the data output is reshaped to 1x8x64, a transformation vividly illustrated in Figure 3-8.

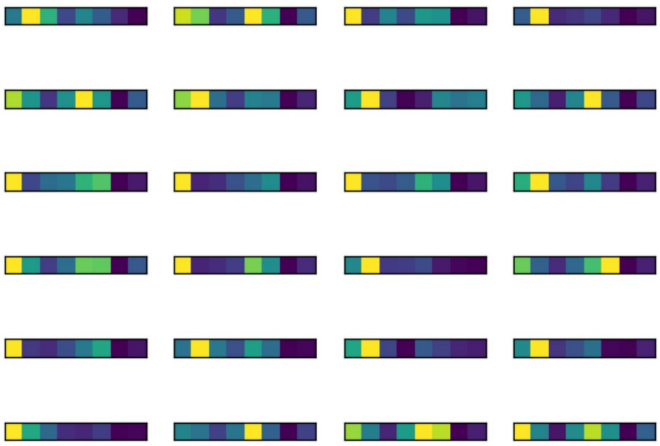

Figure 3-8: The output of the fifth layer.

The sixth layer:

The sixth layer functions as a flattening stage. Within this layer, the data is expanded into a one-dimensional tensor, ensuring seamless integration between the convolutional and fully connected layers. As illustrated in Figure 3-9, each data input is extended into a linear array consisting of 512 vectors.

Figure 3-9: The output of the sixth layer

The seventh layer:

This layer is a fully connected layer, signifying that each neuron from the preceding layer is intricately linked to every neuron in the subsequent hidden layer. Often referred to as the “black box” of the model, this layer plays a pivotal role in its operations. The outcomes generated by this fully connected layer can be observed in Figure 3-10.

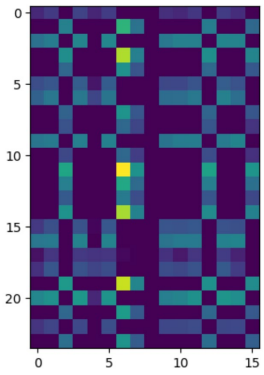

Figure 3-10**:** The output of the sixth layer

In Figure 3-10, out of the 24 inputs showcased, several display similar node value patterns. It’s evident that the appearance of malicious DNS responses differs significantly from the standard DNS responses. When studying the output of the fully connected layer, it is very clear that nodes at positions 0,1,3,4,5,9,10,11,13,14 contribute more towards identifying normal data samples. Conversely, nodes at positions 2,6,7,12,15 are more aligned with recognizing malicious data samples.

The eighth layer:

This is a dropout layer. It uses a strategy specifically designed to combat overfitting in neural networks. Doing so aids in ensuring that ML models focus on genuine patterns rather than incidental noise in the training data. Primarily influential during the training phase, this layer retains all neurons during testing, meaning no dropout is enacted. For this specific model, the dropout rate is set at 5%.

The Ninth Layer:

This represents the model's final layer. For every given input, it produces an array of two integers. An output of [0, 1] indicates that the data sample contains a malicious DNS data sample, while an output of [1, 0] suggests normal data samples.

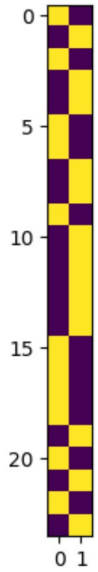

Figure 3-11: The Final output of the model.

3.2 Conclusion

The process of layer-by-layer analysis has illuminated the underlying architecture. As data flows through this system, akin to water coursing through a series of interconnected dams, each layer comes alive, showcasing its specific function. This real-time dissection presents how every layer plays a distinct role. When these layers work together, the model can predict and perform a comprehensive examination of the input data.

From a hands-on, practical standpoint, diving deep into each layer of the model is the process of uncovering the “black box.” This layer-based analysis turns the spotlight onto the inner workings of the model, thereby beginning to dispel the enigmatic “black box.” By delving into the minutiae of decision-making pathways, a clearer vision of the model’s logic will appear. This newfound clarity not only demystifies the process but also significantly bolsters transparency. And, as with any tool or system, when users can discern its workings and rationale, trust is naturally fostered and deepened.

Chapter 4 - Trustee

4.1 Into Trustee

When attempting to interpret and evaluate a trained machine learning model, different analytical tools can yield varied insights. Among all the tools available, decision trees stand out for their simplicity and efficacy. They operate by continually dividing the dataset into smaller subsets based on the most impactful attributes, forming a hierarchical structure of decisions. This makes unraveling the decision-making process of intricate models straightforward with a decision tree.

Trustee is a package that implements the trustee framework[19], [20]. This framework is purposefully crafted to extract decision tree explanations from conventional “black box” ML models. These trees offer a lucid and visual depiction, converting intricate machine learning choices into clear, structured logic, thereby spotlighting the principal decisions taken by the model.

4.2 Method analysis using Trustee

The Trustee tool is meticulous regarding the data and model it accepts as input. As a result, the data must be reshaped to a 1-dimensional array, and an extra layer of reshaping the 1-dimensional data back to its original shape should be added to the model to counteract this reshaping process. Additionally, the Trustee exclusively accepts models with outputs in 1-dimensional array format. Yet, our model delivers a 2-dimensional output, representing the percentages for both DNS spoofing attacks and regular DNS traffic. Thus, the Trustee's main package requires some adjustments, as detailed in Appendix B.

Upon making these modifications, the model can be dissected using the procedure provided in Appendix C. In this segment, we'll select four distinct classifiers, each with their unique set of hyperparameters. The decision tree will have a learning process of 4 rounds, each consisting of 50 iterations.

4.2.1 Classifier-1

The hyperparameters and configuration of this model are outlined in Table 4-1. With the use of Trustee, a decision tree is illustrated in Figure 4-2, while a pruned version of the same tree can be found in Figure 4-1.

Classifier Name	Classifier-3-0-0.h5
Deep Learning Algorithm	CNN
Epoch	10
Batch Size	50
Window Sizes	4
Windows Steps	1
Hidden Tensors	64, 64, 16
Kernel Sizes	2x2,2x2
Dropout Rate	5%

Table 4-1: Hyer parameter for classifier 1

In the Trustee-generated decision tree, the initial file denotes the feature name. A comprehensive list of these feature names and their related bit is available in Appendix D. The values within the tree represent the count of standard and malicious samples. For instance, the root node displays the value [14039,14342], indicating 14,039 regular data samples and 14,342 malicious ones.

Reviewing the tree in Figures 4-1 and 4-2, the decision tree encompasses 127 nodes, whereas its pruned counterpart has only 27. The root node reveals the model's primary decision point: it checks if the fourth TTL bit in the fourth packet is less than or equal to 0.5. This implies that if the fourth TTL bit is set to 1, the sample is deemed malicious. Otherwise, the evaluation moves to the subsequent branch. The color codes—red for regular samples and blue for malicious ones—highlight the significance of decisions based on attributes like DNS flags, TTL, IP total length, and UDP length. Given the four packets in each input window, both the first and last packets play pivotal roles, as evidenced by their proximity to the tree's root.

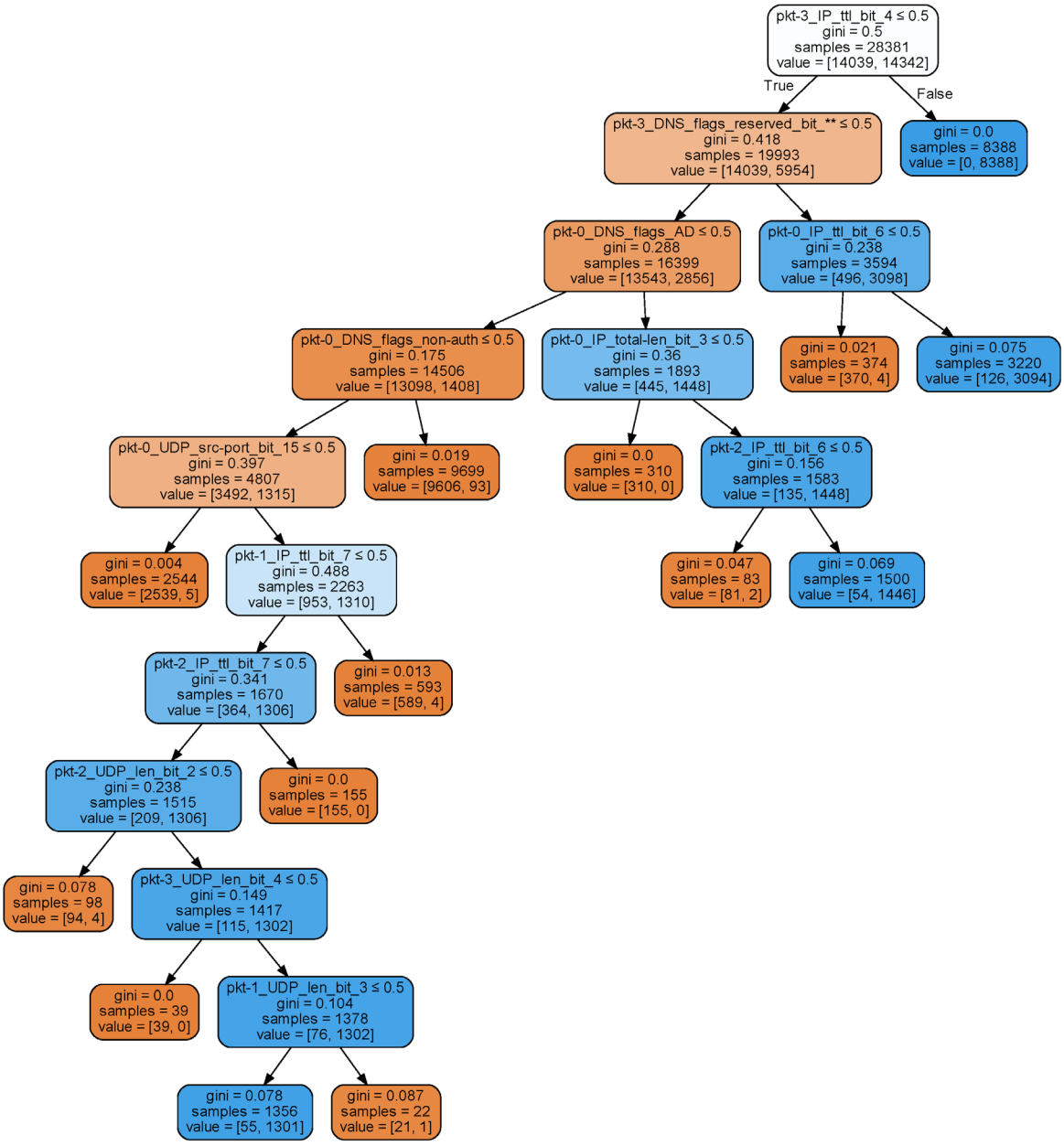

Figure 4-1: Pruned decision tree for classifier 1

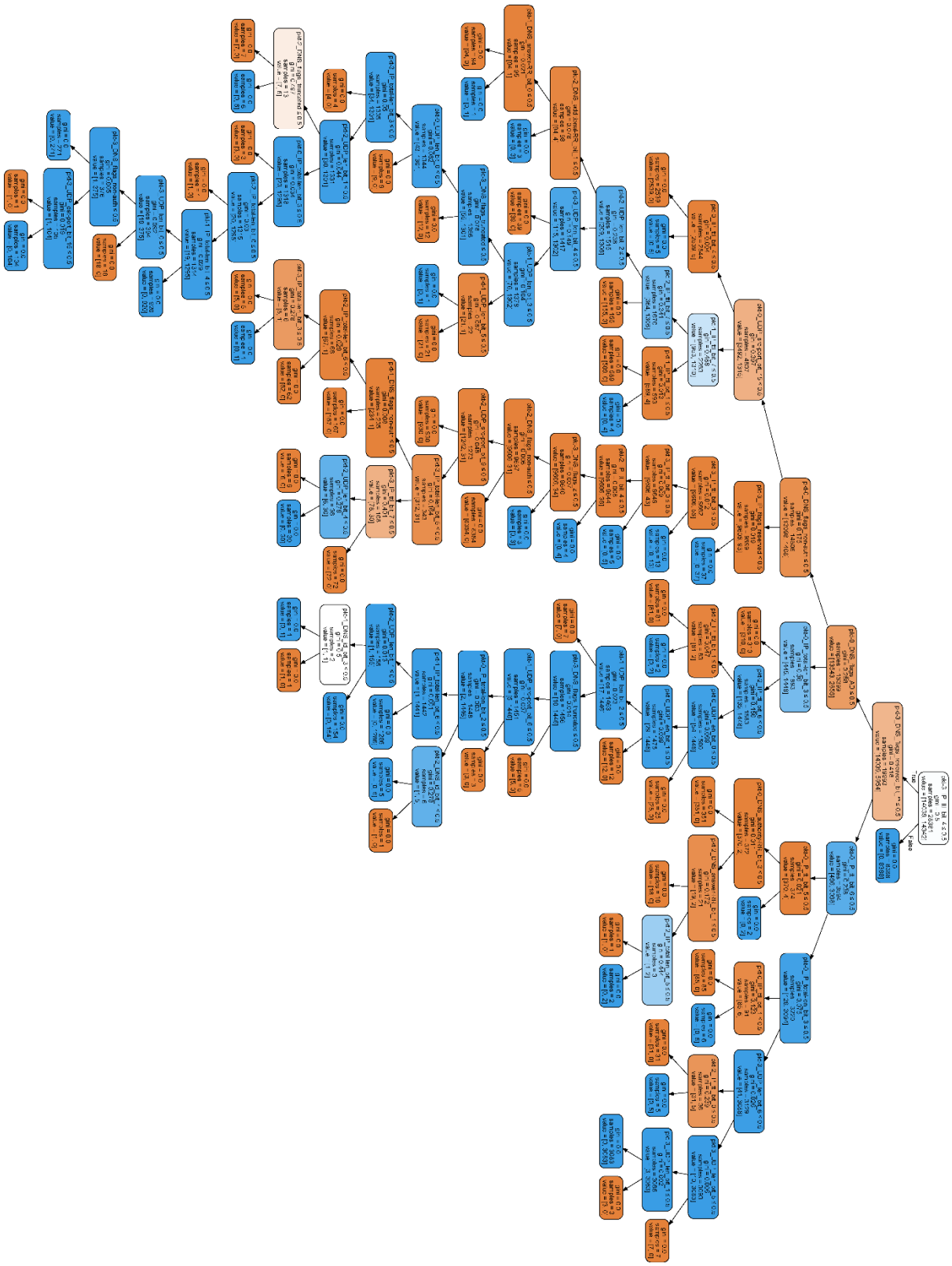

Figure 4-2: Decision tree for classifier 1

4.2.2 Classifier-2

The model's hyperparameters and configurations are detailed in Table 4-2. Leveraging the Trustee tool, we constructed a decision tree, as visualized in Figure 4-4. Concurrently, a pruned decision tree was formulated, showcased in Figure 4-3.

Classifier Name	Classifier-3-15-0.h5
Deep Learning Algorithm	CNN
Epoch	10
Batch Size	50
Window Sizes	4
Windows Steps	1
Hidden Tensors	64,32,16
Kernel Sizes	2x2,2x2
Dropout Rate	5%

Table 4-2: Hyer parameter for classifier 2

This classifier stands out with a different hidden tensor size, setting it apart from the first classifier. Such a difference would typically result in a markedly different internal structure or "black box." Intriguingly, the decision trees from both classifiers converge on the same primary decision at their roots, focusing on the fourth TTL bit in the fourth packet. The comprehensive decision tree boasts a size of 129, while its pruned counterpart is more concise, with just 25 nodes.

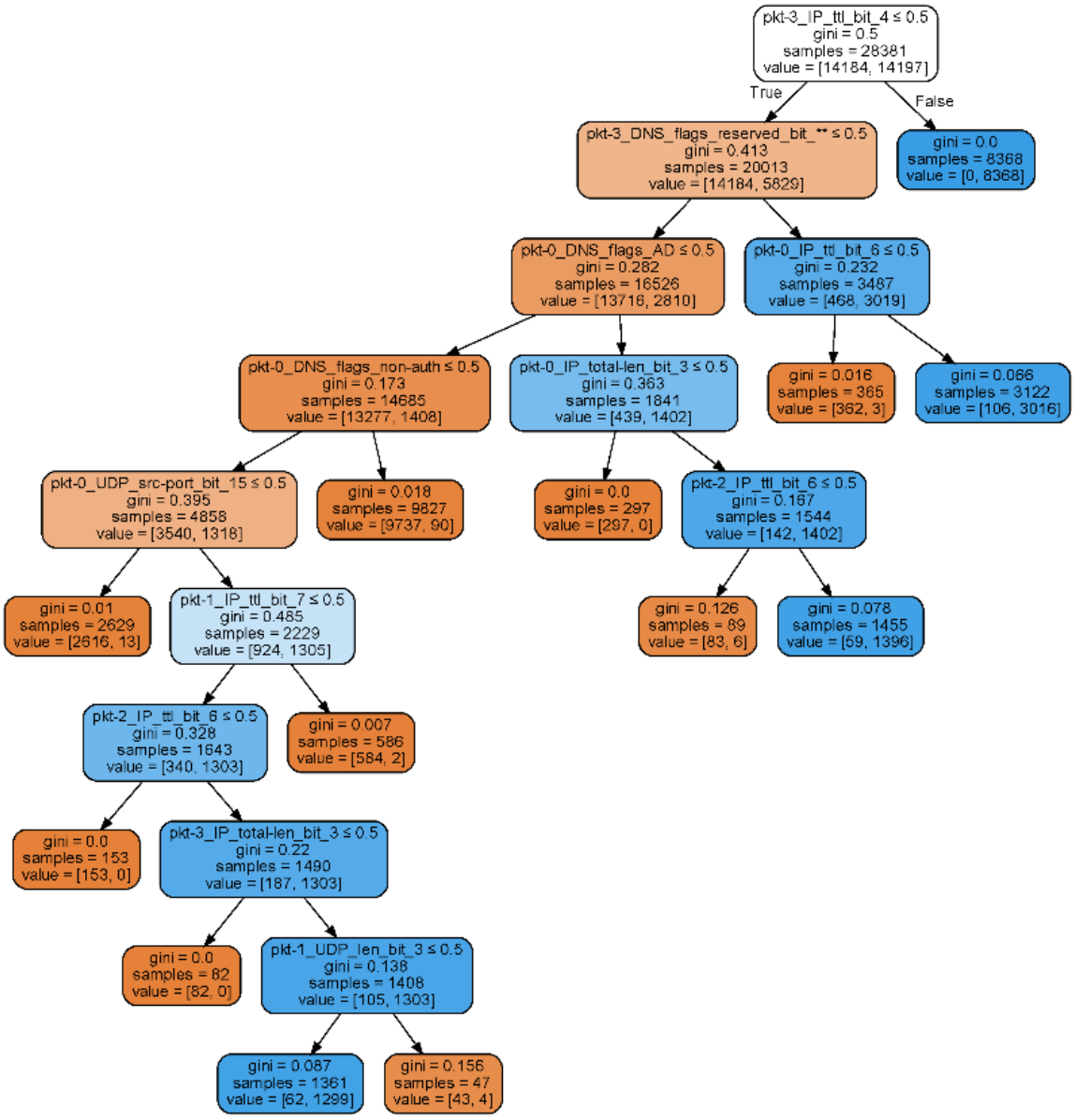

Figure 4-3: Pruned decision tree for classifier 2

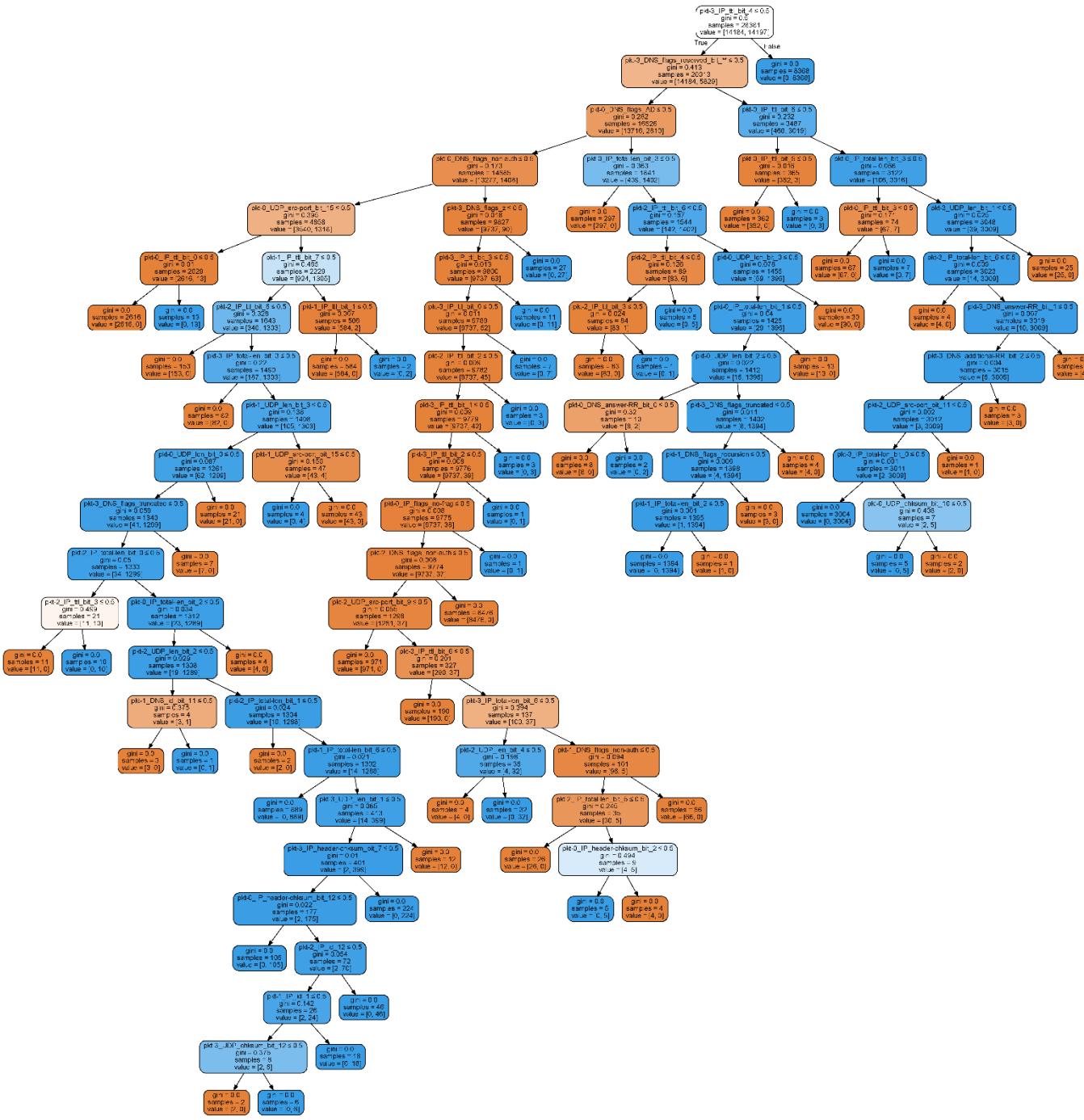

Figure 4-4: Decision tree for classifier 2

4.2.3 Classifier-3

The hyperparameters and configurations of this model are delineated in Table 4-3. With the Trustee tool, a decision tree is generated in Figure 4-6. Simultaneously, a refined, pruned version of the decision tree was crafted and can be viewed in Figure 4-5.

Classifier Name	Classifier-3-50-0.h5
Deep Learning Algorithm	CNN
Epoch	10
Batch Size	50
Window Sizes	4
Windows Steps	1
Hidden Tensors	32, 32, 8
Kernel Sizes	3x3,3x3
Dropout Rate	5%

Table 4-3: Hyer parameter for classifier 3

With the patterns observed in the previous two classifiers, the root node's decision remains hinged on the fourth TTL bit of the fourth network packet. The decision nodes at the first, second, and third levels exhibit striking similarities. Within the decision tree, a majority of the nodes hinge their decisions on specific attributes: TTL, DNS Flags, IP length, and UDP length. Recognizing this consistency underscores a key observation: subtle alterations in kernel size and hidden tensor configurations appear to have minimal bearing on the overarching structure and logic of the decision tree.

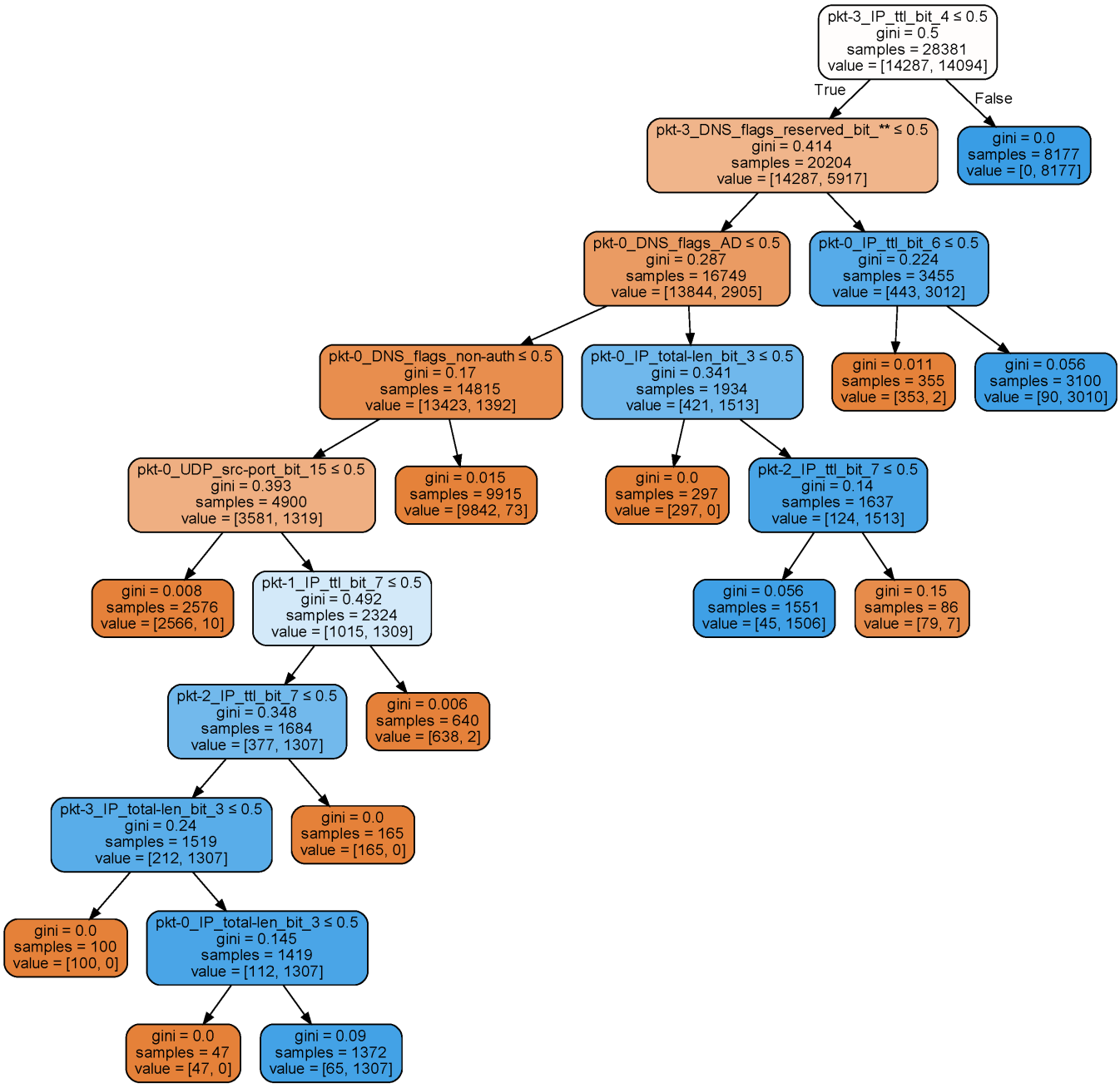

Figure 4-5: Pruned decision tree for classifier 3

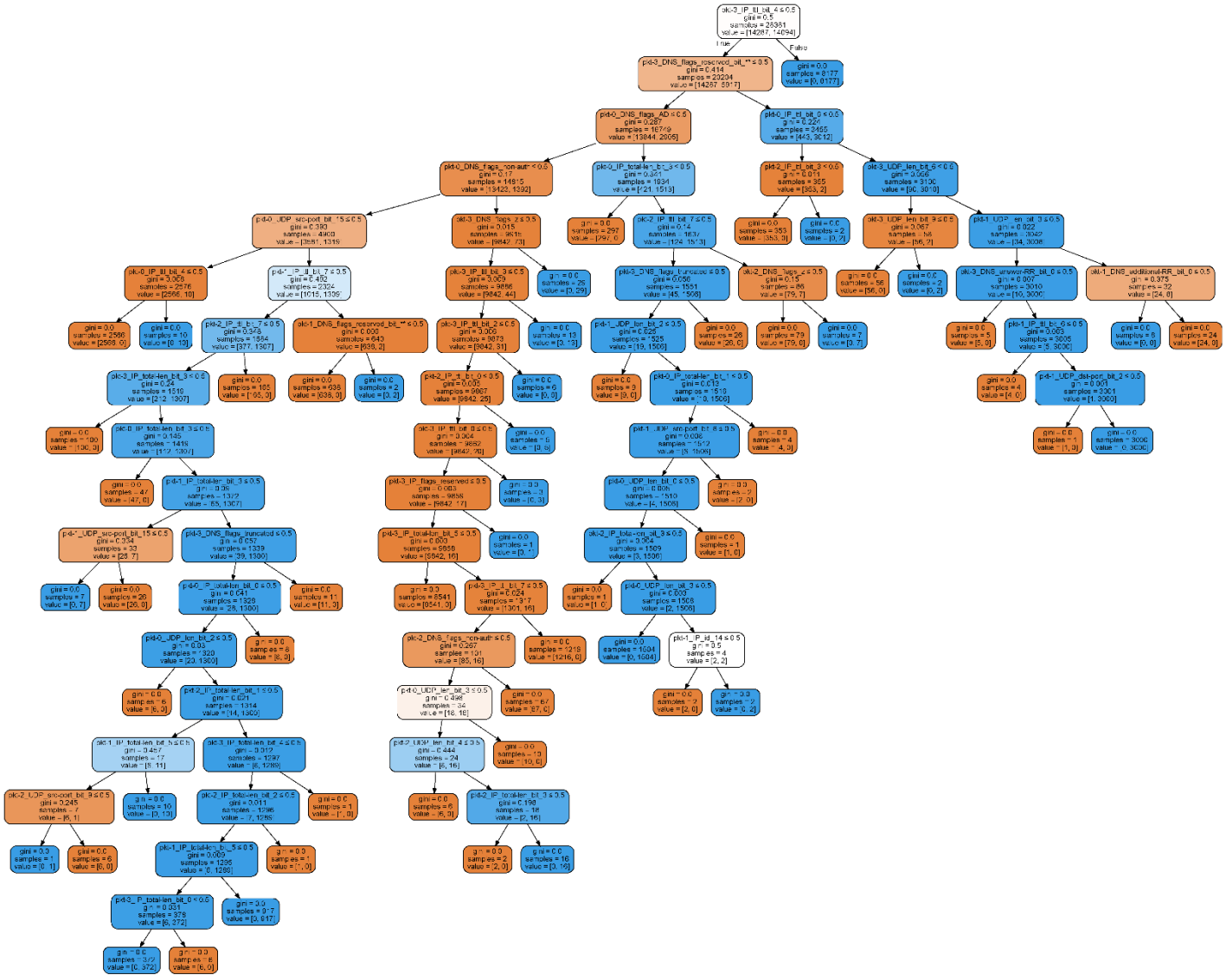

Figure 4-6: Decision tree for classifier 3

4.2.4 Classifier-4

Tables 4-4 show the hyperparameters and configurations for this model. The corresponding decision tree is shown in Figures 4-8 by using the Trustee tool. Concurrently, a pruned version of this tree has been presented in Figure 4-7.

Classifier Name	Classifier-3-74-0.h5
Deep Learning Algorithm	CNN
Epoch	10
Batch Size	50
Window Sizes	4
Windows Steps	1
Hidden Tensors	32, 16, 8
Kernel Sizes	4x4,4x4
Dropout Rate	25%

Table 4-4: Hyer parameter for classifier 4

Despite significant adjustments to the hyperparameters in this fourth classifier, the decision tree exhibits only nuanced variations. However, the root's decision criterion shifted from focusing on the 4th bit to the 5th bit. It seems to be a minor change.

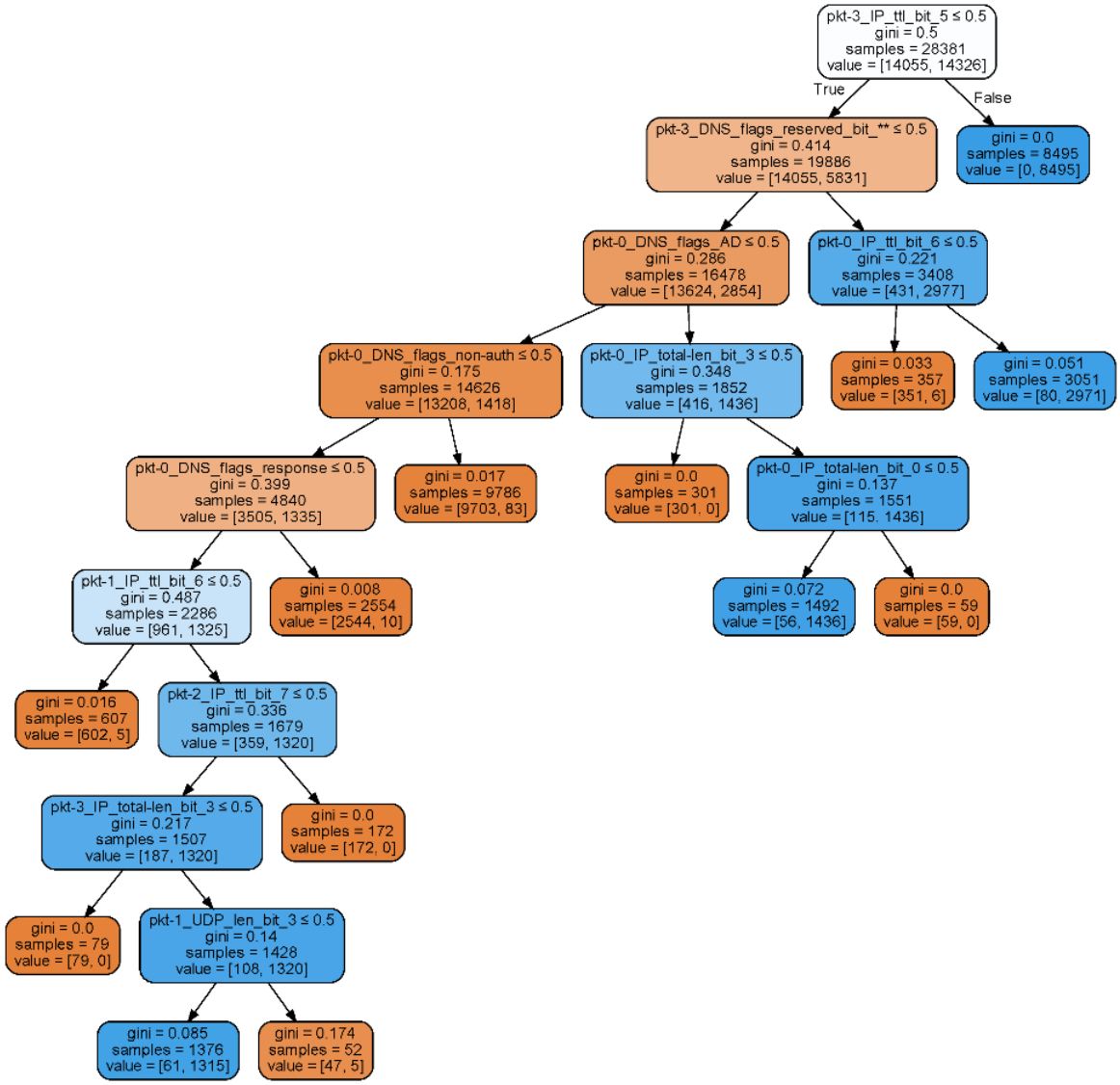

Figure 4-7: Pruned decision tree for classifier 4

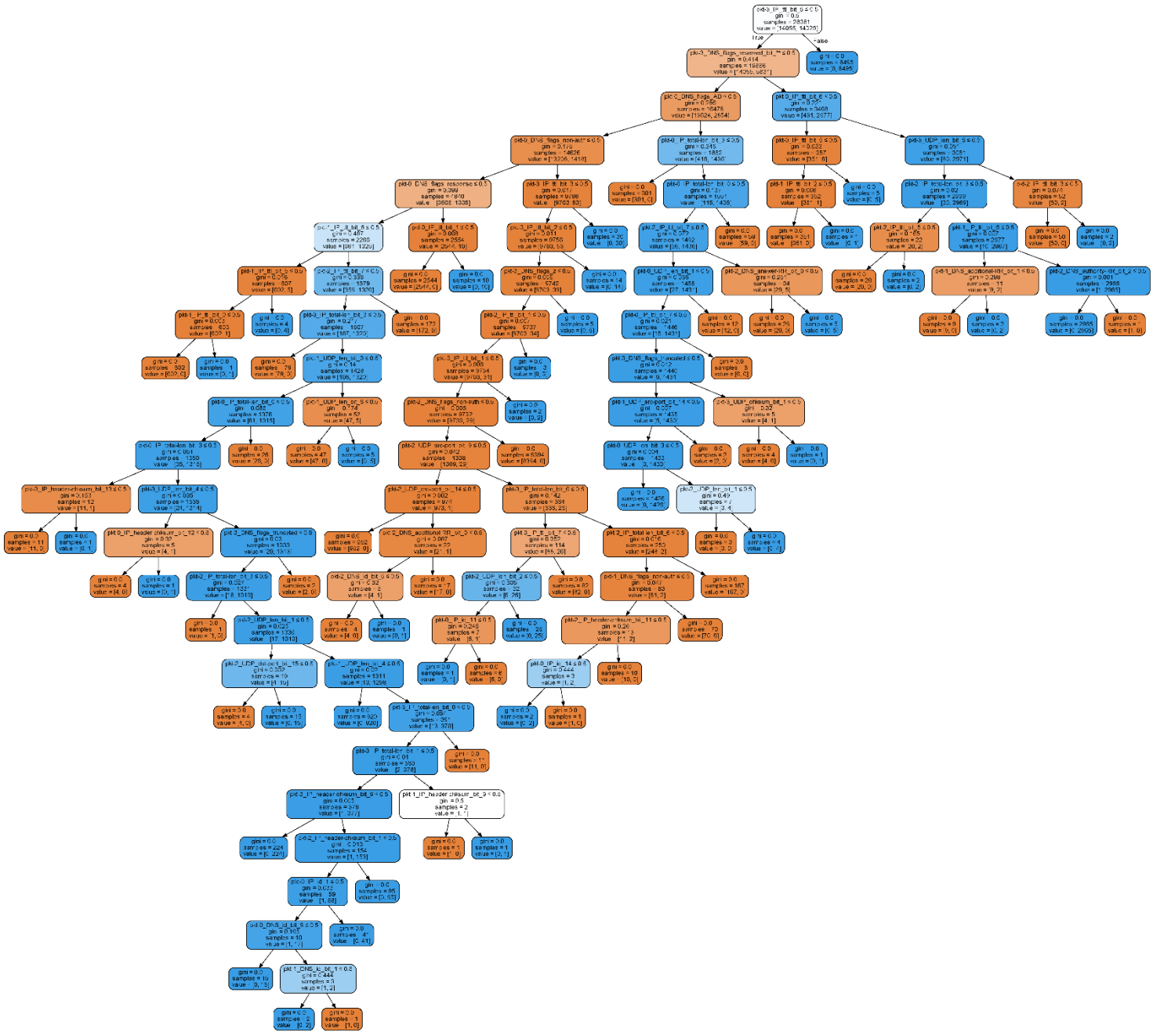

Figure 4-8: Decision tree for classifier 4

4.3 Conclusion

Utilizing the Trustee tool, four decision trees were analyzed from classifiers with distinct hyperparameters. It shows that different hyperparameters don't have a major effect on the decision tree. Across these four trees, the root nodes consistently base their decisions on the TTL (Time To Live) byte from network packets.

Time To Live (TTL) is an integral component of an IP packet's header, acting as a safeguard against packets endlessly circulating within a network. Typical default values for TTL are 64, 128, or 255. As a packet journeys through routers or other networking devices, its TTL value diminishes by one. Once the TTL reaches zero, the packet is discarded, triggering a "Time Exceeded" message relayed back to the original sender. Conventionally, network packets exchanged between the same sender and receiver should maintain consistent TTL values upon reaching their destination. Evidently, the classifier has recognized and harnessed this pattern. With this singular decision criterion, it has adeptly isolated a significant proportion of malicious samples within the dataset.

However, this consistency raises pertinent questions. There's a looming concern about potential biases in the training data. Given that the foundational dataset was amassed in a controlled lab setting [21], one wonders if this environment may have influenced TTL outcomes. Should this model be deployed in real-world scenarios, could a TTL-dependent decision yield excessive false positives? It's worth noting that during the data collection phase, protocol fuzzing was employed, which could introduce realistic noise, simulating true-world data dynamics. But the situation of TTL still exists. However, when crafting packets, TTL and TCP flags can be set differently[22] to avoid this issue.

In subsequent tiers of the decision tree, attributes such as DNS flags, IP length, and UDP length are factored into the decision-making process. By the time the tree reaches its third tier, the differences in decision criteria become more nuanced. This granularity enables the tree to precisely classify a limited subset of samples, but it also leads to the expansion of the tree with added layers and intricacies.

When using Trustee, it is evident that Trustee offers significant advantages in demystifying complex neural networks. By transforming traditionally opaque models into more interpretable decision trees, Trustee provides a view of the underlying decision-making processes, fostering a deeper understanding and trust in the outcomes. Its ability to create both comprehensive and pruned decision trees allows for a tailored approach to data analysis, capturing either the entire tree or the most influential decision points. However, the tool is not without its challenges. The need for data and model restructuring introduces an additional layer of complexity, and its stringent requirements regarding input and output formats can be a limiting factor, especially for models that produce multi-dimensional results. In essence, while Trustee is a powerful asset for enhancing transparency in deep learning models, users must be cognizant of its limitations and be prepared to make necessary adjustments to use its capabilities fully.

Chapter 5 - SHAP

5.1 Intro SHAP

SHAP refers to SHapley Additive exPlanations. It is a Python method that is used to explain the output of any machine learning model. It leverages game theory principles to allocate contribution values for each feature, helping users understand the impact of features on model predictions.[12] Drawing inspiration from cooperative game theory, SHAP utilizes the concept of Shapley values, which traditionally assign a fair distribution of rewards among players based on their individual contributions. Translated into the realm of machine learning, each "player" represents a feature, the "reward" signifies the model's prediction, and the "contribution" quantifies the extent to which each feature influences the prediction relative to the average prediction. When exploring classifiers, SHAP emerges as a powerful tool, enhancing model interpretability. It achieves this by generating intuitive visualizations that spotlight the intricacies of a model's decision-making process, shedding light on the significance and interplay of individual features. Many have used SHAP [23], [24] as tools for analyzing cybersecurity threats.

5.2 Method analysis using SHAP

In contrast to the stringent input requirements of Trustee, SHAP exhibits greater flexibility in handling data inputs. Nonetheless, the nature of the input can influence the variety of visualizations SHAP can produce. For our analysis of each classifier, we will employ two explainers: one that retains the original input dimensions and another that works with a flattened

input structure, necessitating data reshaping. Furthermore, SHAP offers a diverse set of explainer variants. Given our focus on a CNN classifier, the Deep Explainer[25] emerges as the preferred choice. This explainer is specifically crafted for deep neural networks, ingeniously merging the DeepLIFT algorithm with SHAP values to deliver insightful interpretations of deep learning models. Through the utilization of background samples, the Deep SHAP method quantifies the significance of each feature within a neural network. This quantification aids in clarifying the disparity between a particular prediction and the average of all predictions. The specifics of our classifier, including its hyperparameters, are enumerated in Table 5-1. The code used is provided in Appendix E.

Classifier Name	Classifier-3-0-0.h5
Deep Learning Algorithm	CNN
Epoch	10
Batch Size	50
Window Sizes	4
Windows Steps	1
Hidden Tensors	64, 64, 16
Kernel Sizes	2x2,2x2
Dropout Rate	5%

Table 5-1: Hyer parameter for classifier 1

5.2.1 SHAP Value

After deploying the Deep Explainer on the classifier, an array of SHAP values is produced. These values represent the contribution of each feature to a specific prediction relative to the average prediction. In essence, SHAP values illuminate the influence of features on the model's decision-making. In Table 5-2, the top ten most impactful features are ranked in descending order of their importance in summary. This ranking elucidates which attributes are pivotal in swaying the classifier's predictions. Through this table, SHAP succinctly encapsulates and highlights the features that predominantly shape the model's outcomes.

	Feature Name	SHAP Value
838	pkt-3_IP_ttl_bit_1	0.022268
832	pkt-3_IP_ttl_bit_7	0.021788
187	pkt-0_DNS_flags_non-auth	0.019801
320	pkt-1_IP_ttl_bit_7	0.019260
839	pkt-3_IP_ttl_bit_0	0.018181
64	pkt-0_IP_ttl_bit_7	0.015694
576	pkt-2_IP_ttl_bit_7	0.014655
582	pkt-2_IP_ttl_bit_1	0.013904
837	pkt-1_DNS_flags_non-auth	0.011587
443	pkt-3_IP_ttl_bit_2	0.010621

Table 5-2: List of important features

5.2.2 Summary plot

SHAP values offer a detailed interpretation of a machine learning model's output for a particular input, breaking it down by the influence of each feature. This means every input will exhibit unique SHAP values for its features. In Figure 5-1[26], each blue dot signifies the SHAP value of a specific input sample. This visual representation simplifies the process of understanding the behavior and influence of each feature. For instance, the feature "pkt-2_IP_total_len_0" displays a cone-like shape on both ends, with a dense concentration of data points around 0. Such a pattern suggests that this feature has a high likelihood of exerting only a marginal influence, or perhaps none at all, on the output. Conversely, the feature "pkt-3_IP_ttl_bit_1" spreads its data points prominently on both extremes, indicating that this feature often plays a pivotal role in altering the output direction.

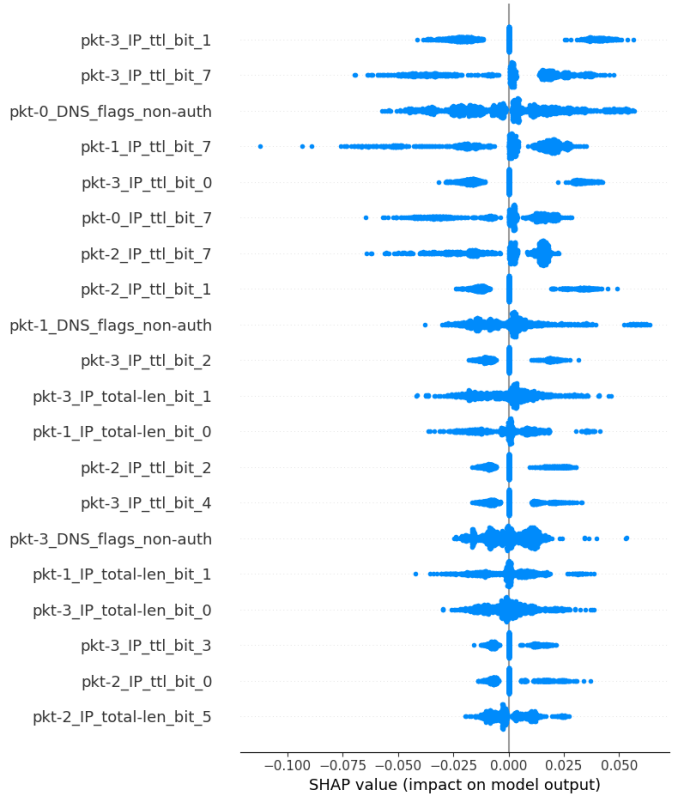

Figure 5-1: Summary plot

5.2.3 Decision Plot

Figure 5-2 presents a decision plot[27] for the initial 100 inputs. Within this visual, each line represents an individual input. As these lines ascend from the base to the peak, the respective input is influenced by features, causing a shift either left or right based on the specific feature values it possesses. The model incorporates a total of 1024 features, but the plot only illustrates 20 of these for clarity. Therefore, at the starting point, each line is positioned at a distinct value. This initial position accounts for the influence of the remaining 1004 features not explicitly depicted. A shift toward blue (0.0) indicates that the data sample is typical, while a drift toward red (1.0) signifies that the sample is malicious.

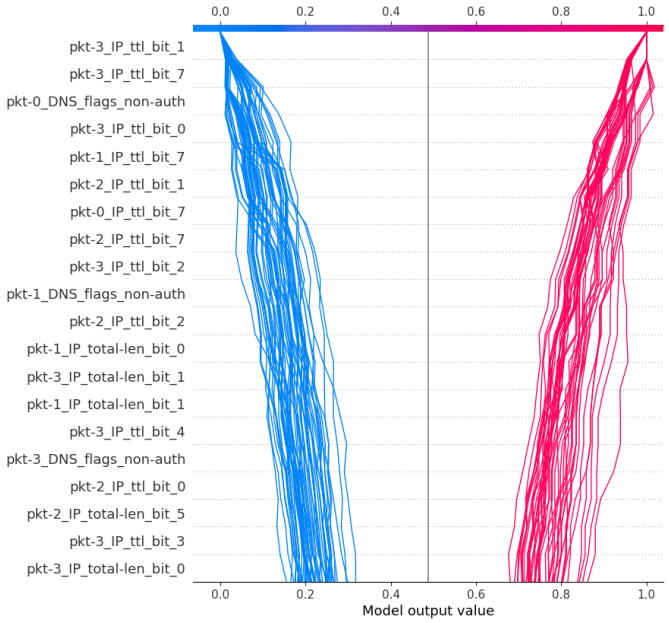

Figure 5-2: Decision plot

For a more detailed examination, Figure 5-3 displays a plot centered on a singular input. This figure elucidates the value of each feature,

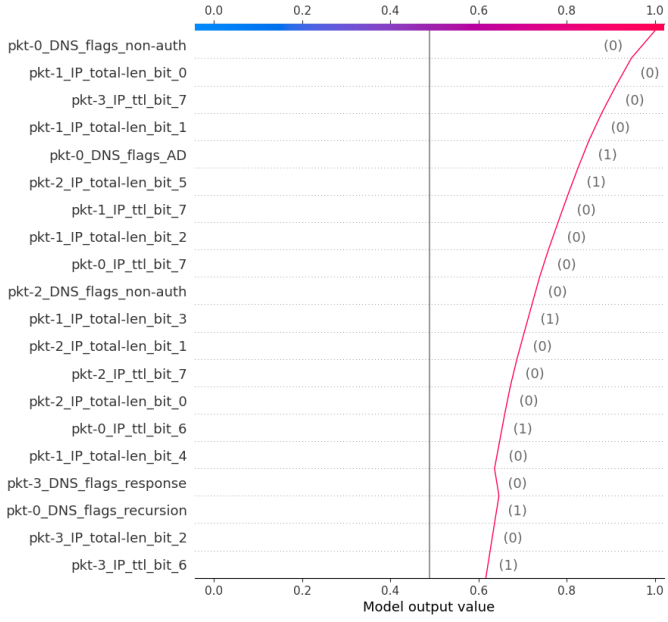

Figure 5-3: Decision plot with one input

which is denoted on the side of the line. The plot reveals that the vast majority of the features push the outcome rightward, suggesting that the input contains a DNS poisoning attack.

5.2.4 Waterfall Plot

The waterfall plot[28], illustrated in Figure 5-4, is a unique visualization tool from SHAP. It is designed to elucidate the sequential contribution of individual features towards the final prediction for a single instance. Unlike conventional plots, it presents a step-by-step decomposition of how each feature moves the prediction from a base value to the model's final output for that particular input.

Figure 5-4: Waterfall Plot

When closely examining the plot in Figure 5-4, one can discern a starting point at 0.512, which represents the aforementioned base value. As the plot cascades upwards, each feature's impact—either increasing or decreasing the prediction—is sequentially added. This progressive, additive nature facilitates an intuitive understanding of feature influence, providing clear visual cues about which features push the model's prediction up and which pull it down. Compared to the decision plot in Figure 5-3, the waterfall plot accentuates the magnitude and

direction of each feature's influence. In essence, the waterfall plot's strength lies in its ability to offer a holistic yet granular view of feature contributions.

5.2.5 Force Plot

The force plot[29], provided by SHAP, serves as a visualization tool to discern the influence of individual features on a classifier's prediction for a specific instance. As seen in Figure 5-5, this graphical representation focuses on the top 50 features out of a total of 1024, based on their prominence in summary values. Even with this fractional subset, the model manages to produce predictions that are generally on track, though understandably not as nuanced as when all features are considered.

Figure 5-5 comprises two distinct plots: the first one characterizing a malicious input and the second delineating a benign or normal input. Central to each plot is the base value, in this case, 0.512, which is essentially the average prediction over all instances as computed by SHAP. The point of intersection between the blue and red bars indicates the model's predicted outcome for that specific instance.

As we navigate the plot, it's evident that features play varying roles in nudging the

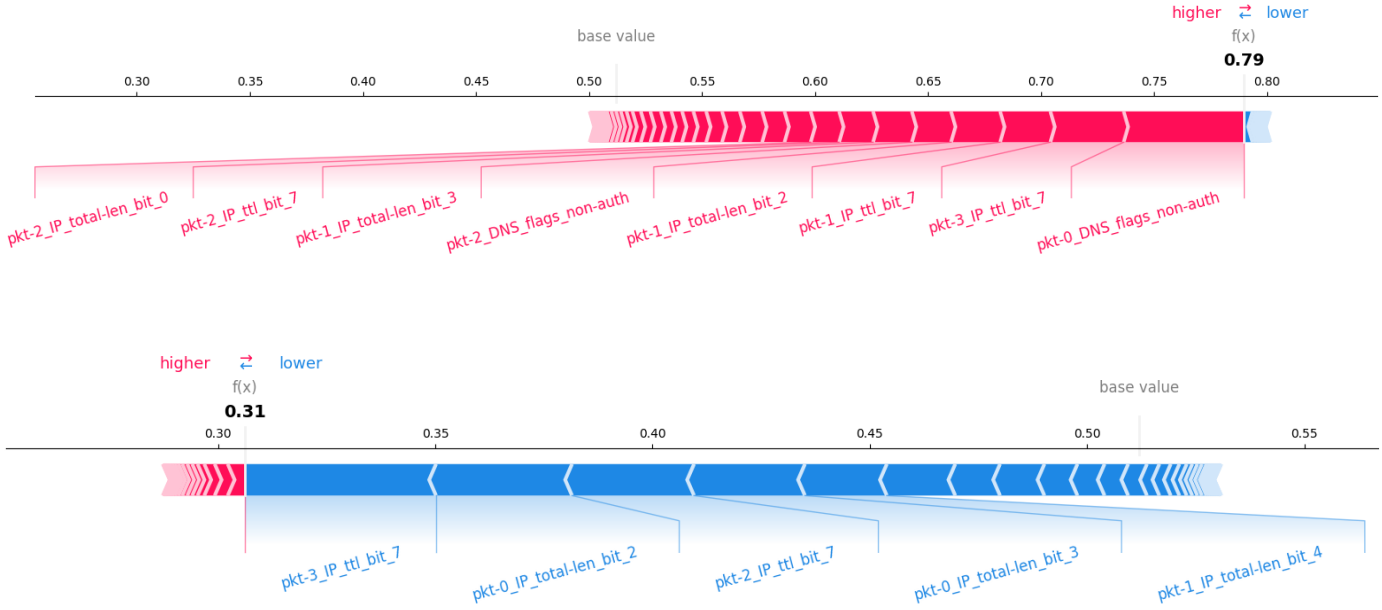

Figure 5-5: Force Plot

prediction. Their effect can be visualized as forces pushing the prediction to the left (toward a benign classification) or to the right (toward a malicious classification). The magnitude of each feature's influence is mirrored in its size; larger bars denote features with a more pronounced contribution to the prediction.

For instance, in the first plot, there's a clear dominance of red features exerting their force towards the right, driving the prediction away from the base value to reach an outcome of 0.79, indicative of a malicious classification. This visual representation allows users to intuitively understand and trace back the decision-making process of the model for specific inputs, highlighting the balance of feature influences that culminate in the final prediction.

5.3 Conclusion

SHAP offers a mathematical approach based on the principles of game theory. It boasts an extensive suite of explainers and visualization utilities, emphasizing its prowess in elucidating individual predictions—a standout advantage. The generation of SHAP values for every input not only enhances model transparency but also endows it with local interpretability. This allows for discerning the rationale behind the model's predictions for specific cases, moving beyond a mere global comprehension of feature significance.

Within the classifier, a cadre of salient features embedded in the network packet has emerged. For instance, the "DNS_flag_non_auth" discerns if a DNS packet originates from a non-authoritative source. Deviations here could be indicative of potential spoofing or even network misconfigurations. The "IP_total_len" sheds light on the entire stretch of the IP packet, acting as a barometer for the data's nature and its volume. Any discrepancy in this length might suggest the packet has been tampered with. "IP_TTL" serves as a beacon for the packet's lifespan, delineating the count of permissible hops before it's deemed obsolete. Anomalies here could be a telltale sign of routing anomalies or a rogue sender. Additionally, the "DNS_flags_response" indicates if a DNS packet stands as a retort to an antecedent plea, while "DNS_flags_recursion" signals the necessity for a recursive resolution in the DNS query. Differences in these flag patterns might show that network packets have been modified, like DNS tunneling or other malevolent DNS exploits.

SHAP's strengths lie in its versatility across different model types, the depth of insights provided by its diverse visualization tools, and its capacity to detail individual predictions, facilitating a granular understanding of model behavior. However, SHAP's computational complexity can be a limitation, especially for large datasets or complex models, potentially

leading to longer processing times and challenges in scalability. This interplay of strengths and weaknesses underscores the need to carefully consider SHAP's applicability based on the specific requirements and constraints of a given task.

Chapter 6 - Conclusion

6.1 Question 1

This research study is on how different analysis tools interpret and explain DNS attack detection models. Aiming to find differences and what they were good at. Each of the tools is used to analyze the model and explain it. However, each of them provides a different way of analyzing the model.

TensorFlow's own tool provides seamless integration with TensorFlow models. It offers real-time visualization and is exceptionally efficient for large-scale neural networks. Using a layer-by-layer view of the data being processed in the model can quickly help understand the process and structure of the model. However, there is a key limitation. That is, the TensorFlow can only be used on TensorFlow models.

Trustee excels on black-box models. Many deep learning models remain hidden in their decision-making process. Trustee delves deep to provide a structured overview. Its primary strength is based on providing a comprehensive understanding of a model's decision-making process by breaking down the model's decisions into a simple decision tree. Trustee allows researchers and practitioners to understand not just what the model is doing but potentially why it's doing it.

SHAP takes a more mathematical approach to model interpretation. It offers consistent and fair attributions, making them highly reliable. Which means it will take a lot of resources to calculate those values. Also, they can be applied to any model, with the rich library of different explainers and visualization tools. It will never face a model that it can't analyze.

6.2 Question 2

To effectively select the optimal XAI tool for DNS attack detection models, it's crucial to first understand the data input format of DNS network packets.

The structure and representation of network packet data play pivotal roles in shaping the performance and clarity of explanations from machine learning models. Consider the scenario where network packets are confined to a length of 32 bytes. This leads to the inception of multiple challenges. Primarily, employing a binary representation, although introducing uniformity, considerably amplifies the feature space. A single byte translates to 8 bits; hence, a packet of 32 bytes corresponds to 256 unique binary features. Introducing a window size of 4, this escalates rapidly to a staggering 1024 binary features. Such a surge in dimensionality can strain computational resources and may hinder the model's ability to discern pertinent patterns among the plethora of features. For tools like SHAP, this poses a unique challenge. With an overwhelming number of features, SHAP may not be able to produce plots with all features. Features exceeding 20 tend to be aggregated, diluting the clarity of visual insights. Also, the heightened calculation demands require more computational resources. However, the SHAP value can still be used for analysis.

Another structure that can affect is the choice of window size. A solitary network packet encapsulates a linear data sequence. The architecture of CNNs and their kernel sizes is critically tethered to the input data's window size. Essentially, the window size delineates the count of network packets amalgamated as a singular input for the ML model. For instance, a mere window size of 2 permits only a 2x2 hidden tensor and a singular max pooling layer. However, an excessively large window size risks devising a model that's globally accurate but lacks

precision. Different window sizes lead to a 4-dimensional input structure. Yet, tools like Trustee inherently support only 2-dimensional input arrays, necessitating adaptations in the model.

Within the intricate domain of DNS, Trustee has been designed to prioritize features, ensuring that the output is both relevant and easily interpretable for domain experts. Through a rigorous selection process, it ensures that its outputs prominently highlight key features, right down to their individual bit significance. This emphasis on clarity and domain-specificity enables experts to swiftly grasp the implications of the results.

Conversely, while SHAP is undeniably a powerful tool with robust methodology, it may not seem immediately understandable to DNS specialists. Its outputs, particularly the SHAP values, can be intricate and might not align with the expertise of those deeply rooted in the DNS field. For a comprehensive understanding and to extract meaningful insights, a deeper dive into SHAP's intricacies is required.

To illustrate the differences between Trustee and SHAP, let's consider DNS flags, a significant feature highlighted in Chapters 4 and 5. DNS flags serve as indicators within DNS messages, relaying specific statuses and guiding aspects of the communication process. For instance, they can signify if a query is recursive or if a response is authoritative. When employing Trustee, a domain expert would encounter a clear representation of how each flag, or a combination thereof, impacts the overall behavior of the DNS system. If the recursion flags play a crucial role in the model's decision-making process, Trustee would emphasize their influence and prioritize them in its decision tree, offering immediate clarity to the expert.

Conversely, SHAP's output often presents a detailed numerical breakdown, with SHAP values illustrating the influence of each feature on each input of the model's prediction. While this information is undoubtedly valuable, it demands a DNS specialist to decipher. This in-depth

granularity may challenge those seeking direct DNS-centric interpretations. Thus, SHAP, despite its comprehensive perspective, requires users to delve deeper into its methodology to transform its outputs into meaningful DNS insights.

In conclusion, the answer for finding the best XAI tool analyzing DNS attack detection models, a hybrid approach should be considered. For each XAI tool: TensorFlow with its instantaneous processing visualization; Trustee with its global prediction; SHAP with its individual prediction; all bring its own set of strengths and idiosyncrasies to the table. Recognizing the limitations of each tool, a collaborative approach might be the key. By synergizing the capabilities of various tools, one could potentially craft a holistic analytical framework that not only accentuates the strengths of each tool but also adeptly counterbalances their individual shortcomings, paving the way for more insightful and robust cybersecurity model evaluations.

References

[1] "AI for Cybersecurity: A Handbook of Use Cases," AI for Cybersecurity: A Handbook of Use Cases. Accessed: Oct. 01, 2023. [Online]. Available: https://psucybersecuritylab.github.io/

[2] "palmtreemodel/PalmTree: Offical Implementation for PalmTree." Accessed: Nov. 01, 2023. [Online]. Available: https://github.com/palmtreemodel/PalmTree

[3] "shensq04/EKLAVYA." Accessed: Nov. 01, 2023. [Online]. Available: https://github.com/shensq04/EKLAVYA

[4] "APIChecker." Accessed: Nov. 01, 2023. [Online]. Available: https://apichecker.github.io/

[5] "Loghub." LOGPAI, Nov. 01, 2023. Accessed: Nov. 01, 2023. [Online]. Available: https://github.com/logpai/loghub

[6] "The UNSW-NB15 Dataset | UNSW Research." Accessed: Nov. 01, 2023. [Online]. Available: https://research.unsw.edu.au/projects/unsw-nb15-dataset

[7] "xiaojunxu/dnn-binary-code-similarity." Accessed: Nov. 01, 2023. [Online]. Available: https://github.com/xiaojunxu/dnn-binary-code-similarity/tree/master

[8] "go2starr/lshhdc at d9dda96c3f297b54f9f09df06c4db6440144addf," GitHub. Accessed: Nov. 01, 2023. [Online]. Available: https://github.com/go2starr/lshhdc

[9] "panda-re/panda at 24681b0dfa80f780d1de4be9bd8b47a286331f55," GitHub. Accessed: Nov. 01, 2023. [Online]. Available: https://github.com/panda-re/panda

[10] X. Yuan, P. He, Q. Zhu, and X. Li, "Adversarial Examples: Attacks and Defenses for Deep Learning," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 30, no. 9, pp. 2805–2824, Sep. 2019, doi: 10.1109/TNNLS.2018.2886017.

[11] "PCCN: Parallel Cross Convolutional Neural Network for Abnormal Network Traffic Flows Detection in Multi-Class Imbalanced Network Traffic Flows | IEEE Journals & Magazine | IEEE Xplore." Accessed: Oct. 30, 2023. [Online]. Available: https://ieeexplore.ieee.org/document/8787567

[12] "Welcome to the SHAP documentation — SHAP latest documentation." Accessed: Oct. 02, 2023. [Online]. Available: https://shap.readthedocs.io/en/latest/

[13] "LIME: Local Interpretable Model-Agnostic Explanations," C3 AI. Accessed: Oct. 31, 2023. [Online]. Available: https://c3.ai/glossary/data-science/lime-local-interpretable-model-agnostic-explanations/

[14] "TensorBoard," TensorFlow. Accessed: Oct. 31, 2023. [Online]. Available: https://www.tensorflow.org/tensorboard

[15] "What is DNS cache poisoning? | DNS spoofing | Cloudflare." Accessed: Oct. 29, 2023. [Online]. Available: https://www.cloudflare.com/learning/dns/dns-cache-poisoning/

[16] M. Janbeglou, M. Zamani, and S. Ibrahim, "Redirecting outgoing DNS requests toward a fake DNS server in a LAN," in *2010 IEEE International Conference on Software Engineering and Service Sciences*, Jul. 2010, pp. 29–32. doi: 10.1109/ICSESS.2010.5552339.

[17] "PSUCyberSecurityLab/AIforCybersecurity." Accessed: Oct. 02, 2023. [Online]. Available: https://github.com/PSUCyberSecurityLab/AIforCybersecurity

[18] "Introduction to TensorFlow." Accessed: Oct. 30, 2023. [Online]. Available: https://www.tensorflow.org/learn?hl=en
[19] "trustee/examples at master · TrusteeML/trustee," GitHub. Accessed: Oct. 02, 2023. [Online]. Available: https://github.com/TrusteeML/trustee/tree/master/examples
[20] "Trustee 1.1.4 documentation." Accessed: Oct. 02, 2023. [Online]. Available: https://trusteeml.github.io/
[21] Q. Zou, A. Singhal, X. Sun, and P. Liu, "Deep Learning for Detecting Network Attacks: An End to End approach," *NIST*, vol. 12840, Jul. 2021, Accessed: Oct. 22, 2023. [Online]. Available: https://www.nist.gov/publications/deep-learning-detecting-network-attacks-end-end-approach
[22] E. Anthi, L. Williams, A. Javed, and P. Burnap, "Hardening machine learning denial of service (DoS) defences against adversarial attacks in IoT smart home networks," *Comput. Secur.*, vol. 108, p. 102352, Sep. 2021, doi: 10.1016/j.cose.2021.102352.
[23] R. Alenezi and S. A. Ludwig, "Explainability of Cybersecurity Threats Data Using SHAP," in *2021 IEEE Symposium Series on Computational Intelligence (SSCI)*, Dec. 2021, pp. 01–10. doi: 10.1109/SSCI50451.2021.9659888.
[24] "Sensors | Free Full-Text | Classification and Explanation for Intrusion Detection System Based on Ensemble Trees and SHAP Method." Accessed: Oct. 30, 2023. [Online]. Available: https://www.mdpi.com/1424-8220/22/3/1154
[25] "shap.DeepExplainer — SHAP latest documentation." Accessed: Oct. 30, 2023. [Online]. Available: https://shap-lrjball.readthedocs.io/en/latest/generated/shap.DeepExplainer.html
[26] "shap.summary_plot — SHAP latest documentation." Accessed: Oct. 30, 2023. [Online]. Available: https://shap-lrjball.readthedocs.io/en/latest/generated/shap.summary_plot.html
[27] "shap.decision_plot — SHAP latest documentation." Accessed: Oct. 30, 2023. [Online]. Available: https://shap-lrjball.readthedocs.io/en/latest/generated/shap.decision_plot.html
[28] "shap.waterfall_plot — SHAP latest documentation." Accessed: Oct. 30, 2023. [Online]. Available: https://shap-lrjball.readthedocs.io/en/latest/generated/shap.waterfall_plot.html
[29] "shap.force_plot — SHAP latest documentation." Accessed: Oct. 30, 2023. [Online]. Available: https://shap-lrjball.readthedocs.io/en/latest/generated/shap.force_plot.html

Appendix A

Code for Layer Analysis

```
# %% [markdown]

# ### 1 Import


# %%

import os

import numpy as np

import tensorflow as tf

import matplotlib.pyplot as plt

import matplotlib.image as mpimg

from tensorflow_addons.metrics import F1Score


# %% [markdown]

# ### 2 Setup


# %% [markdown]

# #### 2.1 Model


# %%

#import
```

```
model = tf.keras.models.load_model(os.path.join("model","classifier-3-0-
0.h5"),custom_objects={"metric":F1Score(num_classes=2)})
X_train=np.load(os.path.join("data","X_train-000.npy"))
print()
X_test=np.load(os.path.join("data","X_test-000.npy"))
X_train.shape, X_test.shape


# %%
model.summary()


# %%
fig, axs = plt.subplots(6, 4)

for i in range(24):
    # plot filter channel in grayscale
    axs[i // 4, i % 4].imshow(np.sum(X_train[i], axis=2) / 8)
    axs[i // 4, i % 4].set_xticks([])
    axs[i // 4, i % 4].set_yticks([])

plt.show()
'''Each square is a bite 256 bit and 32 Byte light means more  4*32*8'''


# %%
```

```
        model_layer_1 = tf.keras.Model(inputs=model.inputs,
outputs=model.layers[1].output)


        # %%
        # summarize filter shapes
        for layer in model.layers:
            # check for convolutional layer
            if 'Hidden' not in layer.name:
                continue
            # get filter weights
            filters, biases = layer.get_weights()


            # normalize filter values to 0-1 so we can visualize them
            f_min, f_max = filters.min(), filters.max()
            filters = (filters - f_min) / (f_max - f_min)
            # plot first few filters
            n_filters, ix = 8, 1
            for i in range(n_filters):
                # get the filter
                f = filters[:, :, :, i]
                # plot each channel separately
                for j in range(8):
                    # specify subplot and turn of axis
```

```
            ax = plt.subplot(n_filters, 8, ix)
            ax.set_xticks([])
            ax.set_yticks([])
            # plot filter channel in grayscale
            plt.imshow(f[:, :, j], cmap='gray')
            ix += 1
            # show the figure
        plt.show()


    # %%

    model_layer_1 = tf.keras.Model(inputs=model.inputs,
outputs=model.layers[1].output)

    fig, axs = plt.subplots(6, 4)

    fm = model_layer_1.predict(tf.convert_to_tensor(X_train[:24]))

    for i in range(24):
        # plot filter channel in grayscale
        axs[i // 4, i % 4].imshow(np.sum(fm[i], axis=2) / 64)
        axs[i // 4, i % 4].set_xticks([])
```

```
        axs[i // 4, i % 4].set_yticks([])

    plt.show()



    # %%
    model_layer_2 = tf.keras.Model(inputs=model.layers[1].output,
outputs=model.layers[2].output)

    fig, axs = plt.subplots(6, 4)

    fm_2 = model_layer_2.predict(tf.convert_to_tensor(fm))

    for i in range(24):
        # plot filter channel in grayscale
        axs[i // 4, i % 4].imshow(np.sum(fm_2[i], axis=2) / 64)
        axs[i // 4, i % 4].set_xticks([])
        axs[i // 4, i % 4].set_yticks([])

    plt.show()

    # %%
```

```
        model_layer_3 = tf.keras.Model(inputs=model.layers[2].output,
outputs=model.layers[3].output)

        fig, axs = plt.subplots(6, 4)

        fm_3 = model_layer_3.predict(tf.convert_to_tensor(fm_2))

        for i in range(24):
            # plot filter channel in grayscale
            axs[i // 4, i % 4].imshow(np.sum(fm_3[i], axis=2) / 64)
            axs[i // 4, i % 4].set_xticks([])
            axs[i // 4, i % 4].set_yticks([])

        plt.show()

        # %%
        model_layer_4 = tf.keras.Model(inputs=model.layers[3].output,
outputs=model.layers[4].output)

        fig, axs = plt.subplots(6, 4)

        fm_4 = model_layer_4.predict(tf.convert_to_tensor(fm_3))
```

```
        for i in range(24):
            # plot filter channel in grayscale
            axs[i // 4, i % 4].imshow(np.sum(fm_4[i], axis=2) / 64)
            axs[i // 4, i % 4].set_xticks([])
            axs[i // 4, i % 4].set_yticks([])

        plt.show()

        # %%
        fm_5[0].shape

        # %%
        model_layer_5 = tf.keras.Model(inputs=model.layers[4].output,
outputs=model.layers[5].output)
        fm_5 = model_layer_5.predict(tf.convert_to_tensor(fm_4))
        plt.imshow(fm_5)
        plt.show()
        #24 行 叠起来



        # %%
        model_layer_6 = tf.keras.Model(inputs=model.layers[5].output,
outputs=model.layers[6].output)
```

```
        fm_6 = model_layer_6.predict(tf.convert_to_tensor(fm_5))
        plt.imshow(fm_6)
        plt.show()
        512

        # %%
        model_layer_7 = tf.keras.Model(inputs=model.layers[6].output,
outputs=model.layers[7].output)
        fm_7 = model_layer_7.predict(tf.convert_to_tensor(fm_6))
        plt.imshow(fm_7)
        plt.show()

        # %%
        model_layer_8 = tf.keras.Model(inputs=model.layers[7].output,
outputs=model.layers[8].output)
        fm_8 = model_layer_8.predict(tf.convert_to_tensor(fm_7))
        plt.imshow(fm_8)
        plt.show()

        # %%
```

Appendix B

main.py Code Modified for Trustee Analysis

```
"""
Trustee
==================================
The core module of the Trustee project
"""
import abc
import functools
import numpy as np
import pandas as pd

from copy import deepcopy



from sklearn.metrics import f1_score, r2_score
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor

from trustee.utils.tree import get_dt_info, top_k_prune
```

```
    def _check_if_trained(func):
        """
        Checks whether the Trustee is already fitted and self._best_student exists

        Parameters
        ----------
        func
            Function to apply decorator to.
        """

        @functools.wraps(func)
        def wrapper(self, *args, **kwargs):
            if len(self._top_students) == 0:
                raise ValueError("No student models have been trained yet. Please fit()
Trustee explainer first.")
            return func(self, *args, **kwargs)

        return wrapper


    class Trustee(abc.ABC):
        """
```

```
Base implementation the Trust-oriented Decision Tree Extraction (Trustee)
algorithm to train student model based on observations from an Expert model.
"""

def __init__(self, expert, student_class, logger=None):
    """
    Trustee constructor

    Parameters
    ----------
    expert
        The ML blackbox model to analyze.
    student_class
        Class of student to train based on blackbox model predictions
    logger (optional)
        A logger object

    """
    self.log = logger.log if logger else print
    self.expert = expert
    self.student_class = student_class

    self._students_by_iter = []
```

```
    self._top_students = []
    self._stable_students = []

    self._X_train = []
    self._X_test = []
    self._y_train = []
    self._y_test = []

    self._best_student = None
    self._features = None
    self._nodes = None
    self._branches = None

@abc.abstractmethod
def _score(self, y_true, y_pred):
    """Score function for student models"""

def fit(
    self,
    X,
    y,
    top_k=10,
    max_leaf_nodes=None,
```

```
    max_depth=None,
    ccp_alpha=0.0,
    train_size=0.7,
    num_iter=50,
    num_stability_iter=5,
    num_samples=2000,
    samples_size=None,
    use_features=None,
    predict_method_name="predict",
    optimization="fidelity",  # for comparative purposes only
    aggregate=True,  # for comparative purposes only
    verbose=False,
):
    """
    Trains Decision Tree Regressor to imitate Expert model.

    Parameters
    ----------
    X
    y
    max_leaf_nodes
    max_depth
    ccp_alpha
```

```
            train_size
            num_iter
            num_samples
            samples_size
            use_features
            predict_method_name
            optimization
            aggregate
            verbose
            """
            if verbose:
                self.log(f"Initializing training dataset using {self.expert} as expert model")

            if len(X) != len(y):
                raise ValueError("Features (X) and target (y) values should have the same
length.")

            # convert data to np array to facilitate processing
            X = pd.DataFrame(X)
            y = pd.Series(y)

            # split input array to train DTs and evaluate agreement
```

```
            self._X_train, self._X_test, self._y_train, self._y_test = train_test_split(X, y,
train_size=train_size)

            features = self._X_train
            targets = pd.Series(getattr(self.expert,
predict_method_name)(self._X_train).argmax(axis=1))

            if hasattr(targets, "shape") and len(targets.shape) >= 2:
                targets = targets.ravel()

            student = self.student_class(
                random_state=0, max_leaf_nodes=max_leaf_nodes,
max_depth=max_depth, ccp_alpha=ccp_alpha
            )

            if verbose:
                self.log(f"Expert model score: {self._score(self._y_train, targets)}")
                self.log(f"Initializing Trustee outer-loop with {num_stability_iter}
iterations")

            # Trustee outer-loop
            for i in range(num_stability_iter):
                self._students_by_iter.append([])
```

```
                if verbose:
                    self.log("#" * 10, f"Outer-loop Iteration {i}/{num_stability_iter}", "#" *
10)
                    self.log(f"Initializing Trustee inner-loop with {num_stability_iter}
iterations")

                # Trustee inner-loop
                for j in range(num_iter):
                    if verbose:
                        self.log("#" * 10, f"Inner-loop Iteration {j}/{num_iter}", "#" * 10)

                    dataset_size = len(features)
                    size = int(int(len(self._X_train)) * samples_size) if samples_size else
num_samples
                    # Step 1: Sample predictions from training dataset
                    if verbose:
                        self.log(
                            f"Sampling {size} points from training dataset with
({len(features)}, {len(targets)}) entries"
                        )

                    samples_idxs = np.random.choice(dataset_size, size=size,
replace=False)
```

```
                X_iter, y_iter = features.iloc[samples_idxs], targets.iloc[samples_idxs]
                X_iter_train, X_iter_test, y_iter_train, y_iter_test = train_test_split(
                    X_iter, y_iter, train_size=train_size
                )

                X_train_student = X_iter_train
                X_test_student = X_iter_test
                if use_features is not None:
                    X_train_student = X_iter_train.iloc[:, use_features]
                    X_test_student = X_iter_test.iloc[:, use_features]

                # Step 2: Traing DecisionTreeRegressor with sampled data
                student.fit(X_train_student.values, y_iter_train.values)
                student_pred = student.predict(X_test_student.values)

                if verbose:
                    self.log(
                        f"Student model {i}-{j} trained with depth {student.get_depth()}
and {student.get_n_leaves()} leaves:"
                    )
                    self.log(f"Student model score: {self._score(y_iter_test,
student_pred)}")
```

```
                # Step 3: Use expert model predictions to aggregate original dataset
                expert_pred = pd.Series(getattr(self.expert,
predict_method_name)(X_iter_test).argmax(axis=1))
                if hasattr(expert_pred, "shape") and len(expert_pred.shape) >= 2:
                    expert_pred = expert_pred.ravel()

                if aggregate:
                    features = pd.concat([features, X_iter_test])
                    targets = pd.concat([targets, expert_pred])

                if optimization == "accuracy":
                    # Step 4: Calculate reward based on Decistion Tree Classifier
accuracy
                    reward = self._score(y_iter_test, student_pred)
                else:
                    # Step 4: Calculate reward based on Decistion Tree Classifier fidelity
to the Expert model
                    reward = self._score(expert_pred, student_pred)

                if verbose:
                    self.log(f"Student model {i}-{j} fidelity: {reward}")

                # Save student to list of iterations dt
```

```
                self._students_by_iter[i].append((deepcopy(student), reward))

            # Save student with highest fidelity to list of top students by iteration
            self._top_students.append(max(self._students_by_iter[i], key=lambda item: item[1]))

        # Get best overall student based on mean agreement
        self._best_student = self.explain(top_k=top_k)[0]

    @_check_if_trained
    def explain(self, top_k=10):
        """
        Returns explainable model that best imitates Expert model, based on calculated rewards.
        """
        # Return dt with highest mean agreement when pruned (with no thrshold)
        stable = self.get_stable(top_k=top_k, threshold=0, sort=False)
        return max(stable, key=lambda item: item[2])

    @_check_if_trained
    def get_stable(self, top_k=10, threshold=0.9, sort=True):
        """
```

Filters out explanations from Trustee sability analysis with less than threshold agreement.

```
        Parameters
        ----------
        top_k = 10
        threshold = 0.9
        sort = True
        """
        if len(self._stable_students) == 0:
            agreement = []
            # Calculate pair-wise agreement of all top students generated during inner loop
            for i, _ in enumerate(self._top_students):
                agreement.append([])
                # Apply top-k prunning before calculating agreement
                base_tree = top_k_prune(self._top_students[i][0], top_k=top_k)
                for j, _ in enumerate(self._top_students):
                    # Apply top-k prunning before calculating agreement
                    iter_tree = top_k_prune(self._top_students[j][0], top_k=top_k)

                    iter_y_pred = iter_tree.predict(self._X_test.values)
                    base_y_pred = base_tree.predict(self._X_test.values)
```

```
                agreement[i].append(self._score(iter_y_pred, base_y_pred))

            # Save complete dt, top-k prune dt, mean agreement and fidelity
            self._stable_students.append(
                (
                    self._top_students[i][0],
                    base_tree,
                    np.mean(agreement[i]),
                    self._top_students[i][1],
                )
            )

        stable = self._stable_students
        if threshold > 0:
            stable = filter(lambda item: item[2] >= threshold, stable)

        if sort:
            return sorted(stable, key=lambda item: item[2], reverse=True)

        return stable

    @_check_if_trained
```

```
        def get_all_students(self):
            """
            Returns list of all (student, reward) obtained during the inner-loop process.
            """
            return self._students_by_iter

        @_check_if_trained
        def get_top_students(self):
            """
            Returns list of top (students, reward) obtained during the outer-loop process.
            """
            return self._top_students

        @_check_if_trained
        def get_n_features(self):
            """
            Returns number of features used in the top student model.
            """
            if not self._features:
                self._features, self._nodes, self._branches =
get_dt_info(self._best_student)

            return len(self._features.keys())
```

```
        @_check_if_trained
        def get_n_classes(self):
            """
            Returns number of classes used in the top student model.
            """
            return self._best_student.tree_.n_classes[0]

        @_check_if_trained
        def get_samples_sum(self):
            """
            Returns the sum of all samples in all non-leaf _nodes in best student model.
            """
            left = self._best_student.tree_.children_left
            right = self._best_student.tree_.children_right
            samples = self._best_student.tree_.n_node_samples

            return np.sum([n_samples if left[node] != right[node] else 0 for node,
n_samples in enumerate(samples)])

        @_check_if_trained
        def get_top_branches(self, top_k=10):
            """
```

```
            Returns list of top _branches of the best student.
            """
            if not self._branches:
                self._features, self._nodes, self._branches =
get_dt_info(self._best_student)

            return sorted(self._branches, key=lambda p: p["samples"],
reverse=True)[:top_k]

        @_check_if_trained
        def get_top_features(self, top_k=10):
            """
            Returns list of top _features of the best student.
            """
            if not self._features:
                self._features, self._nodes, self._branches =
get_dt_info(self._best_student)

            return sorted(self._features.items(), key=lambda p: p[1]["samples"],
reverse=True)[:top_k]

        @_check_if_trained
        def get_top_nodes(self, top_k=10):
```

```
            """
            Returns list of top _nodes of the best student.
            """
            if not self._nodes:
                self._features, self._nodes, self._branches =
get_dt_info(self._best_student)

            return sorted(
                self._nodes, key=lambda p: p["samples"] * abs(p["gini_split"][0] -
p["gini_split"][1]), reverse=True
            )[:top_k]

        @_check_if_trained
        def get_samples_by_level(self):
            """
            Returns list of samples by level of the best student.
            """
            if not self._nodes:
                self._features, self._nodes, self._branches =
get_dt_info(self._best_student)

            samples_by_level = list(np.zeros(self._best_student.get_depth() + 1))
```

```
            nodes_by_level = list(np.zeros(self._best_student.get_depth() +
1).astype(int))
            for node in self._nodes:
                samples_by_level[node["level"]] += node["samples"]

            for node in self._branches:
                samples_by_level[node["level"]] += node["samples"]
                nodes_by_level[node["level"]] += 1

            return samples_by_level

        @_check_if_trained
        def get_leaves_by_level(self):
            """
            Returns list of leaves by level of the best student.
            """
            if not self._branches:
                self._features, self._nodes, self._branches =
get_dt_info(self._best_student)

            leaves_by_level = list(np.zeros(self._best_student.get_depth() +
1).astype(int))
            for node in self._branches:
```

```
                leaves_by_level[node["level"]] += 1

        return leaves_by_level

    @_check_if_trained
    def prune(self, top_k=10, max_impurity=0.10):
        """
        Prunes and returns the best student model explanation from the list of
_students_by_iter.

        Parameters
        ----------
        top_k
        max_impurity
        """
        return top_k_prune(self._best_student, top_k=top_k,
max_impurity=max_impurity)


class ClassificationTrustee(Trustee):
    """
    Implements the Trust-oriented Decision Tree Extraction (Trustee) algorithm to
train
```

```
        a student Decision Tree Classifier based on observations from an Expert
classification model.
        """

        def __init__(self, expert, logger=None):
            """
            Classification Trustee constructor

            Parameters
            ----------
            expert
                The ML blackbox model to analyze.
            student_class
                Class of student to train based on blackbox model predictions
            logger (optional)
                A logger object
            """
            super().__init__(expert, student_class=DecisionTreeClassifier,
logger=logger)

        def _score(self, y_true, y_pred, average="macro"):
            """
            F1-score function for classification student models
```

```
        Parameters
        ----------
        y_true
        y_pred
        """
        return f1_score(y_true, y_pred, average=average)


class RegressionTrustee(Trustee):
    """
    Implements the Trust-oriented Decision Tree Extraction (Trustee) algorithm to train a
    student Decision Tree Regressor based on observations from an Expert regression model.
    """

    def __init__(self, expert, logger=None):
        """
        Regression Trustee constructor

        Parameters
        ----------
```

```
            expert
                The ML blackbox model to analyze.
            student_class
                Class of student to train based on blackbox model predictions
            logger (optional)
                A logger object
            """
            super().__init__(expert=expert, student_class=DecisionTreeRegressor,
logger=logger)


        def _score(self, y_true, y_pred):
            """
            R2-score function for regression student models

            Parameters
            ----------
            y_true
            y_pred
            """
            return r2_score(y_true, y_pred)
```

Appendix C

Code for Trustee Analysis

```
# %%

import os

import argparse

import numpy as np

import tensorflow as tf

from sklearn import tree

from tensorflow_addons.metrics import F1Score

from sklearn.metrics import classification_report

from trustee import ClassificationTrustee

from sklearn.metrics import r2_score

import graphviz

from sklearn import tree

import pickle

FLAGS=None


# %%

def perf_measure(y_actual, y_pred):

    TP = 0

    FP = 0

    TN = 0
```

```
    FN = 0
    for i in range(len(y_pred)):
        if y_actual[i][1] == 1 and y_pred[i][1] >= 0.5:
            TP += 1
        if y_pred[i][1] >= 0.5 and y_actual[i][1] == 0:
            FP += 1
        if y_actual[i][1] == 0 and y_pred[i][1] < 0.5:
            TN += 1
        if y_pred[i][1] <= 0.5 and y_actual[i][1] == 1:
            FN += 1

    return(TP, FP, TN, FN)



# %%
if __name__ == "__main__":
    parser=argparse.ArgumentParser()
    parser.add_argument(
        '--n',
        type=int,
        required=False,
        help='ID of training.'
    )
```

```
        parser.add_argument(
            '--k',
            type=int,
            required=False,
            help='ID of model. The models to load should be placed in the model
directory, named \"classifier-N-K-[0,1,2,3].h5\"'
        )
        FLAGS, unparsed =parser.parse_known_args()


        gpus = tf.config.list_physical_devices('GPU')
        if gpus:
            try:
                # Currently, memory growth needs to be the same across GPUs
                for gpu in gpus:
                    tf.config.experimental.set_memory_growth(gpu, True)
                logical_gpus = tf.config.experimental.list_logical_devices('GPU')
                print(len(gpus), "Physical GPUs,", len(logical_gpus), "Logical GPUs")
            except RuntimeError as e:
                # Memory growth must be set before GPUs have been initialized
                print(e)
        else:
            tf.config.threading.set_inter_op_parallelism_threads(8)
            tf.config.threading.set_intra_op_parallelism_threads(8)
```

```
    n_epoch = 10
    n_batch = 50
    FLAGS.k = 74
    FLAGS.n = 3
    ss=FLAGS.k//90

# %%
X_train=np.load(os.path.join("data","X_train-%.3d.npy"%ss),allow_pickle=True)
oldshape = X_train.shape
print("X_train Orignal:"+str(X_train.shape))
X_train = X_train.reshape(X_train.shape[0], -1)
print("X_train reshape:"+str(X_train.shape))
newshape = X_train.shape
X_test=np.load(os.path.join("data","X_test-%.3d.npy"%ss),allow_pickle=True)
print("X_test Orignal:"+str(X_test.shape))
X_test = X_test.reshape(X_test.shape[0], -1)
print("X_test reshape:"+str(X_test.shape))
Y_train=np.load(os.path.join("data","Y_train-%.3d.npy"%ss),allow_pickle=True)
print("Y_train Orignal:"+str(Y_train.shape))
Y_train = Y_train.reshape(Y_train.shape[0], -1)
print("Y_train reshape:"+str(Y_train.shape))
Y_test=np.load(os.path.join("data","Y_test-%.3d.npy"%ss),allow_pickle=True)
```

```
        print("Y_test Orignal:"+str(Y_test.shape))
        Y_test_flat = Y_test.reshape(Y_test.shape[0], -1)
        print("Y_test reshape:"+str(Y_test.shape))

        # %%
        predictions_train=[]
        predictions_test=[]
        for i in range(1):
            model=tf.keras.models.load_model(os.path.join("model","classifier-
"+str(FLAGS.n)+"-"+str(FLAGS.k)+"-
"+str(i)+".h5"),custom_objects={"metric":F1Score(num_classes=2)})
            '''Added'''



            new_input = tf.keras.layers.Input(shape=(newshape[1],))

            reshaped_input = tf.keras.layers.Reshape((oldshape[1], oldshape[2],
oldshape[3]))(new_input)

            # Get the output of your trained model
            output = model(reshaped_input)
```

```
    # Create a new model
    new_model = tf.keras.Model(inputs=new_input, outputs=output)




    '''
    '''
    predictions_train.append(new_model.predict(X_train))
    predictions_test.append(new_model.predict(X_test))
new_model.summary()

# %%
# performing predictions on the test dataset
y_pred = new_model.predict(X_test)



# %%
# Evaluate model accuracy
print("Model R2-score:")
print(r2_score(Y_test, y_pred))

# %%
```

```
field_names={
    0: "IP_ver_bit_3", 1: "IP_ver_bit_2", 2: "IP_ver_bit_1", 3: "IP_ver_bit_0",
    4: "IP_header-len_bit_3", 5: "IP_header-len_bit_2", 6: "IP_header-len_bit_1",
7: "IP_header-len_bit_0",
    8: "IP_diff-service_bit_7", 9: "IP_diff-service_bit_6", 10: "IP_diff-
service_bit_5", 11: "IP_diff-service_bit_4", 12: "IP_diff-service_bit_3", 13: "IP_diff-
service_bit_2", 14: "IP_diff-service_bit_1", 15: "IP_diff-service_bit_0",
    16: "IP_total-len_bit_15", 17: "IP_total-len_bit_14", 18: "IP_total-len_bit_13",
19: "IP_total-len_bit_12", 20: "IP_total-len_bit_11", 21: "IP_total-len_bit_10", 22:
"IP_total-len_bit_9", 23: "IP_total-len_bit_8", 24: "IP_total-len_bit_7", 25: "IP_total-
len_bit_6", 26: "IP_total-len_bit_5", 27: "IP_total-len_bit_4", 28: "IP_total-len_bit_3",
29: "IP_total-len_bit_2", 30: "IP_total-len_bit_1", 31: "IP_total-len_bit_0",
    32: "IP_id_bit_15", 33: "IP_id_14", 34: "IP_id_13", 35: "IP_id_12", 36:
"IP_id_11", 37: "IP_id_10", 38: "IP_id_9", 39: "IP_id_8", 40: "IP_id_7", 41: "IP_id_6",
42: "IP_id_5", 43: "IP_id_4", 44: "IP_id_3", 45: "IP_id_2", 46: "IP_id_1", 47: "IP_id_0",
    48: "IP_flags_reserved", 49: "IP_flags_no-frag", 50: "IP_flags_more-frag",
    51: "IP_frag-offset_bit_12", 52: "IP_frag-offset_bit_11", 53: "IP_frag-
offset_bit_10", 54: "IP_frag-offset_bit_9", 55: "IP_frag-offset_bit_8", 56: "IP_frag-
offset_bit_7", 57: "IP_frag-offset_bit_6", 58: "IP_frag-offset_bit_5", 59: "IP_frag-
offset_bit_4", 60: "IP_frag-offset_bit_3", 61: "IP_frag-offset_bit_2", 62: "IP_frag-
offset_bit_1", 63: "IP_frag-offset_bit_0",
    64: "IP_ttl_bit_7", 65: "IP_ttl_bit_6", 66: "IP_ttl_bit_5", 67: "IP_ttl_bit_4", 68:
"IP_ttl_bit_3", 69: "IP_ttl_bit_2", 70: "IP_ttl_bit_1", 71: "IP_ttl_bit_0",
```

72: "IP_protocol_bit_7", 73: "IP_protocol_bit_6", 74: "IP_protocol_bit_5", 75: "IP_protocol_bit_4", 76: "IP_protocol_bit_3", 77: "IP_protocol_bit_2", 78: "IP_protocol_bit_1", 79: "IP_protocol_bit_0",

80: "IP_header-chksum_bit_15", 81: "IP_header-chksum_bit_14", 82: "IP_header-chksum_bit_13", 83: "IP_header-chksum_bit_12", 84: "IP_header-chksum_bit_11", 85: "IP_header-chksum_bit_10", 86: "IP_header-chksum_bit_9", 87: "IP_header-chksum_bit_8", 88: "IP_header-chksum_bit_7", 89: "IP_header-chksum_bit_6", 90: "IP_header-chksum_bit_5", 91: "IP_header-chksum_bit_4", 92: "IP_header-chksum_bit_3", 93: "IP_header-chksum_bit_2", 94: "IP_header-chksum_bit_1", 95: "IP_header-chksum_bit_0",

96: "UDP_src-port_bit_15", 97: "UDP_src-port_bit_14", 98: "UDP_src-port_bit_13", 99: "UDP_src-port_bit_12", 100: "UDP_src-port_bit_11", 101: "UDP_src-port_bit_10", 102: "UDP_src-port_bit_9", 103: "UDP_src-port_bit_8", 104: "UDP_src-port_bit_7", 105: "UDP_src-port_bit_6", 106: "UDP_src-port_bit_5", 107: "UDP_src-port_bit_4", 108: "UDP_src-port_bit_3", 109: "UDP_src-port_bit_2", 110: "UDP_src-port_bit_1", 111: "UDP_src-port_bit_0",

112: "UDP_dst-port_bit_15", 113: "UDP_dst-port_bit_14", 114: "UDP_dst-port_bit_13", 115: "UDP_dst-port_bit_12", 116: "UDP_dst-port_bit_11", 117: "UDP_dst-port_bit_10", 118: "UDP_dst-port_bit_9", 119: "UDP_dst-port_bit_8", 120: "UDP_dst-port_bit_7", 121: "UDP_dst-port_bit_6", 122: "UDP_dst-port_bit_5", 123: "UDP_dst-port_bit_4", 124: "UDP_dst-port_bit_3", 125: "UDP_dst-port_bit_2", 126: "UDP_dst-port_bit_1", 127: "UDP_dst-port_bit_0",

128: "UDP_len_bit_15", 129: "UDP_len_bit_14", 130: "UDP_len_bit_13", 131: "UDP_len_bit_12", 132: "UDP_len_bit_11", 133: "UDP_len_bit_10", 134: "UDP_len_bit_9", 135: "UDP_len_bit_8", 136: "UDP_len_bit_7", 137: "UDP_len_bit_6", 138: "UDP_len_bit_5", 139: "UDP_len_bit_4", 140: "UDP_len_bit_3", 141: "UDP_len_bit_2", 142: "UDP_len_bit_1", 143: "UDP_len_bit_0",

144: "UDP_chksum_bit_15", 145: "UDP_chksum_bit_14", 146: "UDP_chksum_bit_13", 147: "UDP_chksum_bit_12", 148: "UDP_chksum_bit_11", 149: "UDP_chksum_bit_10", 150: "UDP_chksum_bit_9", 151: "UDP_chksum_bit_8", 152: "UDP_chksum_bit_7", 153: "UDP_chksum_bit_6", 154: "UDP_chksum_bit_5", 155: "UDP_chksum_bit_4", 156: "UDP_chksum_bit_3", 157: "UDP_chksum_bit_2", 158: "UDP_chksum_bit_1", 159: "UDP_chksum_bit_0",

160: "DNS_id_bit_15", 161: "DNS_id_bit_14", 162: "DNS_id_bit_13", 163: "DNS_id_bit_12", 164: "DNS_id_bit_11", 165: "DNS_id_bit_10", 166: "DNS_id_bit_9", 167: "DNS_id_bit_8", 168: "DNS_id_bit_7", 169: "DNS_id_bit_6", 170: "DNS_id_bit_5", 171: "DNS_id_bit_4", 172: "DNS_id_bit_3", 173: "DNS_id_bit_2", 174: "DNS_id_bit_1", 175: "DNS_id_bit_0",

176: "DNS_flags_response", 177: "DNS_flags_opcode_bit_3", 178: "DNS_flags_opcode_bit_2", 179: "DNS_flags_opcode_bit_1", 180: "DNS_flags_opcode_bit_0", 181: "DNS_flags_reserved_bit_*", 182: "DNS_flags_truncated", 183: "DNS_flags_recursion", 184: "DNS_flags_reserved_bit_**", 185: "DNS_flags_z", 186: "DNS_flags_AD", 187: "DNS_flags_non-auth", 188: "DNS_flags_reserved_bit_3", 189:

"DNS_flags_reserved_bit_2", 190: "DNS_flags_reserved_bit_1", 191: "DNS_flags_reserved_bit_0",

192: "DNS_questions_bit_15", 193: "DNS_questions_bit_14", 194: "DNS_questions_bit_13", 195: "DNS_questions_bit_12", 196: "DNS_questions_bit_11", 197: "DNS_questions_bit_10", 198: "DNS_questions_bit_9", 199: "DNS_questions_bit_8", 200: "DNS_questions_bit_7", 201: "DNS_questions_bit_6", 202: "DNS_questions_bit_5", 203: "DNS_questions_bit_4", 204: "DNS_questions_bit_3", 205: "DNS_questions_bit_2", 206: "DNS_questions_bit_1", 207: "DNS_questions_bit_0",

208: "DNS_answer-RR_bit_15", 209: "DNS_answer-RR_bit_14", 210: "DNS_answer-RR_bit_13", 211: "DNS_answer-RR_bit_12", 212: "DNS_answer-RR_bit_11", 213: "DNS_answer-RR_bit_10", 214: "DNS_answer-RR_bit_9", 215: "DNS_answer-RR_bit_8", 216: "DNS_answer-RR_bit_7", 217: "DNS_answer-RR_bit_6", 218: "DNS_answer-RR_bit_5", 219: "DNS_answer-RR_bit_4", 220: "DNS_answer-RR_bit_3", 221: "DNS_answer-RR_bit_2", 222: "DNS_answer-RR_bit_1", 223: "DNS_answer-RR_bit_0",

224: "DNS_authority-RR_bit_15", 225: "DNS_authority-RR_bit_14", 226: "DNS_authority-RR_bit_13", 227: "DNS_authority-RR_bit_12", 228: "DNS_authority-RR_bit_11", 229: "DNS_authority-RR_bit_10", 230: "DNS_authority-RR_bit_9", 231: "DNS_authority-RR_bit_8", 232: "DNS_authority-RR_bit_7", 233: "DNS_authority-RR_bit_6", 234: "DNS_authority-RR_bit_5", 235: "DNS_authority-RR_bit_4", 236: "DNS_authority-RR_bit_3", 237: "DNS_authority-RR_bit_2", 238: "DNS_authority-RR_bit_1", 239: "DNS_authority-RR_bit_0",

```
            240: "DNS_additional-RR_bit_15", 241: "DNS_additional-RR_bit_14", 242:
"DNS_additional-RR_bit_13", 243: "DNS_additional-RR_bit_12", 244:
"DNS_additional-RR_bit_11", 245: "DNS_additional-RR_bit_10", 246:
"DNS_additional-RR_bit_9", 247: "DNS_additional-RR_bit_8", 248: "DNS_additional-
RR_bit_7", 249: "DNS_additional-RR_bit_6", 250: "DNS_additional-RR_bit_5", 251:
"DNS_additional-RR_bit_4", 252: "DNS_additional-RR_bit_3", 253: "DNS_additional-
RR_bit_2", 254: "DNS_additional-RR_bit_1", 255: "DNS_additional-RR_bit_0",
        }
        feature_names=[]
        for i in range(4):
            for j in range(256):
                feature_names.append("pkt-"+str(i)+"_"+field_names[j])


        # %% [markdown]
        # Trustee


        # %%
        trustee = ClassificationTrustee(expert=new_model)
        print(X_train[:,1].shape)
        trustee.fit(X_train, Y_train[:,1], num_iter=50, num_stability_iter=10,
samples_size=0.3, verbose=True)
```

```
# %%

dt, pruned_dt, agreement, reward = trustee.explain()


# %%

dt_y_pred = dt.predict(X_test)


# %%

def find_misclassified(y_actual, y_pred):
    TP = 0
    FP = 0
    FP_list=[]
    TN = 0
    FN = 0
    FN_list=[]
    for i in range(len(y_pred)):
        if y_actual[i] == 1 and y_pred[i] >= 0.5:
            TP += 1
        if y_pred[i] >= 0.5 and y_actual[i] == 0:
            FP += 1
            FP_list.append(i)
        if y_actual[i] == 0 and y_pred[i] < 0.5:
            TN += 1
        if y_pred[i] <= 0.5 and y_actual[i] == 1:
```

```
            FN += 1
            FN_list.append(i)

    return(TP, FP, TN, FN, FP_list, FN_list)

# %%
TP,FP,TN,FN,FP_list,FN_list=find_misclassified(Y_test[:,1],dt.predict(X_test))

# %%
print("accuracy: %f"%((TP+TN)/(TP+TN+FP+FN)))
print("F1: %f"%((TP)/(TP+0.5*(FP+FN))))
print("DR: %f"%((TP)/(TP+FN)))
print("FPR: %f"%((FP)/(TN+FP)))
print(FP_list)
print(FN_list)

# %%
dot_data = tree.export_graphviz(dt, feature_names=feature_names,
out_file=None)
graph = graphviz.Source(dot_data)
graph.render("DT-trustee-fields")
dot_data = tree.export_graphviz(dt,out_file=None)
graph = graphviz.Source(dot_data)
```

```
graph.render("DT-trustee")


# %%
fi=open("DT-trustee.pkl",'wb')
pickle.dump(dt,fi)
fi.close()


# %%
fi=open("DT-trustee.pkl",'rb')
dt=pickle.load(fi)
fi.close()


# %%
dt, pruned_dt, agreement, reward = trustee.explain()
print(f'Model explanation training (agreement, fidelity): ({agreement},
{reward})")
print(f"Model Explanation size: {dt.tree_.node_count}")
print(f"Top-k Prunned Model explanation size: {pruned_dt.tree_.node_count}")


# %%
dot_data = tree.export_graphviz(
    dt,
    feature_names=feature_names,
```

```
    filled=True,
    rounded=True,
    special_characters=True,
)
graph = graphviz.Source(dot_data)
graph.render("dt_explanation")

# Output pruned decision tree to pdf
dot_data = tree.export_graphviz(
    pruned_dt,
    feature_names=feature_names,
    filled=True,
    rounded=True,
    special_characters=True,
)
graph = graphviz.Source(dot_data)
graph.render("pruned_dt_explation")
```

Appendix D

Field Name for SHAP Each Bit

0:IP_ver_bit_3

1:IP_ver_bit_2

2:IP_ver_bit_1

3:IP_ver_bit_0

4:IP_header-len_bit_3

5:IP_header-len_bit_2

6:IP_header-len_bit_1

7:IP_header-len_bit_0

8:IP_diff-service_bit_7

9:IP_diff-service_bit_6

10:IP_diff-service_bit_5

11:IP_diff-service_bit_4

12:IP_diff-service_bit_3

13:IP_diff-service_bit_2

14:IP_diff-service_bit_1

15:IP_diff-service_bit_0

16:IP_total-len_bit_15

17:IP_total-len_bit_14

18:IP_total-len_bit_13

19:IP_total-len_bit_12

20:IP_total-len_bit_11

21:IP_total-len_bit_10

22:IP_total-len_bit_9

23:IP_total-len_bit_8

24:IP_total-len_bit_7

25:IP_total-len_bit_6

26:IP_total-len_bit_5

27:IP_total-len_bit_4

28:IP_total-len_bit_3

29:IP_total-len_bit_2

30:IP_total-len_bit_1

31:IP_total-len_bit_0

32:IP_id_bit_15

33:IP_id_14

34:IP_id_13

35:IP_id_12

36:IP_id_11

37:IP_id_10

38:IP_id_9

39:IP_id_8

40:IP_id_7

41:IP_id_6

42:IP_id_5

43:IP_id_4

44:IP_id_3

45:IP_id_2

46:IP_id_1

47:IP_id_0

48:IP_flags_reserved

49:IP_flags_no-frag

50:IP_flags_more-frag

51:IP_frag-offset_bit_12

52:IP_frag-offset_bit_11

53:IP_frag-offset_bit_10

54:IP_frag-offset_bit_9

55:IP_frag-offset_bit_8

56:IP_frag-offset_bit_7

57:IP_frag-offset_bit_6

58:IP_frag-offset_bit_5

59:IP_frag-offset_bit_4

60:IP_frag-offset_bit_3

61:IP_frag-offset_bit_2

62:IP_frag-offset_bit_1

63:IP_frag-offset_bit_0

64:IP_ttl_bit_7

65:IP_ttl_bit_6

66:IP_ttl_bit_5

67:IP_ttl_bit_4

68:IP_ttl_bit_3

69:IP_ttl_bit_2

70:IP_ttl_bit_1

71:IP_ttl_bit_0

72:IP_protocol_bit_7

73:IP_protocol_bit_6

74:IP_protocol_bit_5

75:IP_protocol_bit_4

76:IP_protocol_bit_3

77:IP_protocol_bit_2

78:IP_protocol_bit_1

79:IP_protocol_bit_0

80:IP_header-chksum_bit_15

81:IP_header-chksum_bit_14

82:IP_header-chksum_bit_13

83:IP_header-chksum_bit_12

84:IP_header-chksum_bit_11

85:IP_header-chksum_bit_10

86:IP_header-chksum_bit_9

87:IP_header-chksum_bit_8

88:IP_header-chksum_bit_7

89:IP_header-chksum_bit_6

90:IP_header-chksum_bit_5

91:IP_header-chksum_bit_4

92:IP_header-chksum_bit_3

93:IP_header-chksum_bit_2

94:IP_header-chksum_bit_1

95:IP_header-chksum_bit_0

96:UDP_src-port_bit_15

97:UDP_src-port_bit_14

98:UDP_src-port_bit_13

99:UDP_src-port_bit_12

100:UDP_src-port_bit_11

101:UDP_src-port_bit_10

102:UDP_src-port_bit_9

103:UDP_src-port_bit_8

104:UDP_src-port_bit_7

105:UDP_src-port_bit_6

106:UDP_src-port_bit_5

107:UDP_src-port_bit_4

108:UDP_src-port_bit_3

109:UDP_src-port_bit_2

110:UDP_src-port_bit_1

111:UDP_src-port_bit_0

112:UDP_dst-port_bit_15

113:UDP_dst-port_bit_14

114:UDP_dst-port_bit_13

115:UDP_dst-port_bit_12

116:UDP_dst-port_bit_11

117:UDP_dst-port_bit_10

118:UDP_dst-port_bit_9

119:UDP_dst-port_bit_8

120:UDP_dst-port_bit_7

121:UDP_dst-port_bit_6

122:UDP_dst-port_bit_5

123:UDP_dst-port_bit_4

124:UDP_dst-port_bit_3

125:UDP_dst-port_bit_2

126:UDP_dst-port_bit_1

127:UDP_dst-port_bit_0

128:UDP_len_bit_15

129:UDP_len_bit_14

130:UDP_len_bit_13

131:UDP_len_bit_12

132:UDP_len_bit_11

133:UDP_len_bit_10

134:UDP_len_bit_9

135:UDP_len_bit_8

136:UDP_len_bit_7

137:UDP_len_bit_6

138:UDP_len_bit_5

139:UDP_len_bit_4

140:UDP_len_bit_3

141:UDP_len_bit_2

142:UDP_len_bit_1

143:UDP_len_bit_0

144:UDP_chksum_bit_15

145:UDP_chksum_bit_14

146:UDP_chksum_bit_13

147:UDP_chksum_bit_12

148:UDP_chksum_bit_11

149:UDP_chksum_bit_10

150:UDP_chksum_bit_9

151:UDP_chksum_bit_8

152:UDP_chksum_bit_7

153:UDP_chksum_bit_6

154:UDP_chksum_bit_5

155:UDP_chksum_bit_4

156:UDP_chksum_bit_3

157:UDP_chksum_bit_2

158:UDP_chksum_bit_1

159:UDP_chksum_bit_0

160:DNS_id_bit_15

161:DNS_id_bit_14

162:DNS_id_bit_13

163:DNS_id_bit_12

164:DNS_id_bit_11

165:DNS_id_bit_10

166:DNS_id_bit_9

167:DNS_id_bit_8

168:DNS_id_bit_7

169:DNS_id_bit_6

170:DNS_id_bit_5

171:DNS_id_bit_4

172:DNS_id_bit_3

173:DNS_id_bit_2

174:DNS_id_bit_1

175:DNS_id_bit_0

176:DNS_flags_response

177:DNS_flags_opcode_bit_3

178:DNS_flags_opcode_bit_2

179:DNS_flags_opcode_bit_1

180:DNS_flags_opcode_bit_0

181:DNS_flags_reserved_bit_*

182:DNS_flags_truncated

183:DNS_flags_recursion

184:DNS_flags_reserved_bit_**

185:DNS_flags_z

186:DNS_flags_AD

187:DNS_flags_non-auth

188:DNS_flags_reserved_bit_3

189:DNS_flags_reserved_bit_2

190:DNS_flags_reserved_bit_1

191:DNS_flags_reserved_bit_0

192:DNS_questions_bit_15

193:DNS_questions_bit_14

194:DNS_questions_bit_13

195:DNS_questions_bit_12

196:DNS_questions_bit_11

197:DNS_questions_bit_10

198:DNS_questions_bit_9

199:DNS_questions_bit_8

200:DNS_questions_bit_7

201:DNS_questions_bit_6

202:DNS_questions_bit_5

203:DNS_questions_bit_4

204:DNS_questions_bit_3

205:DNS_questions_bit_2

206:DNS_questions_bit_1

207:DNS_questions_bit_0

208:DNS_answer-RR_bit_15

209:DNS_answer-RR_bit_14

210:DNS_answer-RR_bit_13

211:DNS_answer-RR_bit_12

212:DNS_answer-RR_bit_11

213:DNS_answer-RR_bit_10

214:DNS_answer-RR_bit_9

215:DNS_answer-RR_bit_8

216:DNS_answer-RR_bit_7

217:DNS_answer-RR_bit_6

218:DNS_answer-RR_bit_5

219:DNS_answer-RR_bit_4

220:DNS_answer-RR_bit_3

221:DNS_answer-RR_bit_2

222:DNS_answer-RR_bit_1

223:DNS_answer-RR_bit_0

224:DNS_authority-RR_bit_15

225:DNS_authority-RR_bit_14

226:DNS_authority-RR_bit_13

227:DNS_authority-RR_bit_12

228:DNS_authority-RR_bit_11

229:DNS_authority-RR_bit_10

230:DNS_authority-RR_bit_9

231:DNS_authority-RR_bit_8

232:DNS_authority-RR_bit_7

233:DNS_authority-RR_bit_6

234:DNS_authority-RR_bit_5

235:DNS_authority-RR_bit_4

236:DNS_authority-RR_bit_3

237:DNS_authority-RR_bit_2

238:DNS_authority-RR_bit_1

239:DNS_authority-RR_bit_0

240:DNS_additional-RR_bit_15

241:DNS_additional-RR_bit_14

242:DNS_additional-RR_bit_13

243:DNS_additional-RR_bit_12

244:DNS_additional-RR_bit_11

245:DNS_additional-RR_bit_10

246:DNS_additional-RR_bit_9

247:DNS_additional-RR_bit_8

248:DNS_additional-RR_bit_7

249:DNS_additional-RR_bit_6

250:DNS_additional-RR_bit_5

251:DNS_additional-RR_bit_4

252:DNS_additional-RR_bit_3

253:DNS_additional-RR_bit_2

254:DNS_additional-RR_bit_1

255:DNS_additional-RR_bit_0

Appendix E

Code for SHAP Analysis

```
# %%

import os

import numpy as np

import tensorflow as tf

from tensorflow_addons.metrics import F1Score

from tensorflow import keras

import argparse

from keras import layers

import pickle

import pandas as pd

import shap

from IPython.core.display import display, HTML

import matplotlib.pyplot as plt


# %%

if __name__ == "__main__":

    parser=argparse.ArgumentParser()

    parser.add_argument(

        '--n',

        type=int,
```

```
        required=False,
        help='ID of training.'
    )
    parser.add_argument(
        '--k',
        type=int,
        required=False,
        help='ID of model. The models to load should be placed in the model
directory, named \"classifier-N-K-[0,1,2,3].h5\"'
    )
    FLAGS, unparsed =parser.parse_known_args()


    gpus = tf.config.list_physical_devices('GPU')
    if gpus:
        try:
            # Currently, memory growth needs to be the same across GPUs
            for gpu in gpus:
                tf.config.experimental.set_memory_growth(gpu, True)
            logical_gpus = tf.config.experimental.list_logical_devices('GPU')
            print(len(gpus), "Physical GPUs,", len(logical_gpus), "Logical GPUs")
        except RuntimeError as e:
            # Memory growth must be set before GPUs have been initialized
            print(e)
```

```
    else:
        tf.config.threading.set_inter_op_parallelism_threads(8)
        tf.config.threading.set_intra_op_parallelism_threads(8)

    n_epoch = 10
    n_batch = 50
    FLAGS.k = 0
    FLAGS.n = 3
    ss=FLAGS.k//90

# %%

os.environ["CUDA_VISIBLE_DEVICES"] = "-1"
X_train=np.load(os.path.join("data","X_train-%.3d.npy"%ss),allow_pickle=True)
X_test=np.load(os.path.join("data","X_test-%.3d.npy"%ss),allow_pickle=True)
Y_train=np.load(os.path.join("data","Y_train-%.3d.npy"%ss),allow_pickle=True)
Y_test=np.load(os.path.join("data","Y_test-%.3d.npy"%ss),allow_pickle=True)


# %%
train_shape=X_train.shape
X_train_flattened=X_train.reshape((train_shape[0],train_shape[1]*train_shape[2]
*train_shape[3]))
```

```
test_shape=X_test.shape
X_test_flattened=X_test.reshape((test_shape[0],test_shape[1]*test_shape[2]*test_
shape[3]))


# %%
gpus = tf.config.list_physical_devices('GPU')
if gpus:
  try:
    # Currently, memory growth needs to be the same across GPUs
    for gpu in gpus:
      tf.config.experimental.set_memory_growth(gpu, True)
    logical_gpus = tf.config.list_logical_devices('GPU')
    print(len(gpus), "Physical GPUs,", len(logical_gpus), "Logical GPUs")
  except RuntimeError as e:
    # Memory growth must be set before GPUs have been initialized
    print(e)
else:
  print("Using CPU")
  tf.config.threading.set_inter_op_parallelism_threads(8)
  tf.config.threading.set_intra_op_parallelism_threads(8)


# %%
predictions_train=[]
```

```
        predictions_test=[]
        for i in range(1):
            model=tf.keras.models.load_model(os.path.join("model","classifier-
"+str(FLAGS.n)+"-"+str(FLAGS.k)+"-
"+str(i)+".h5"),custom_objects={"metric":F1Score(num_classes=2)})
            '''Added'''



            new_input = tf.keras.layers.Input(shape=(X_train_flattened.shape[1],))

            reshaped_input = tf.keras.layers.Reshape((train_shape[1], train_shape[2],
train_shape[3]))(new_input)

            # Get the output of your trained model
            output = model(reshaped_input)

            # Create a new model
            new_model = tf.keras.Model(inputs=new_input, outputs=output)
```

```
    '''
    '''
    predictions_train.append(new_model.predict(X_train_flattened))
    predictions_test.append(new_model.predict(X_test_flattened))
new_model.summary()


# %%
background = X_train[np.random.choice(X_train.shape[0], 100, replace=False)]
orignal_explainer = shap.DeepExplainer(model, background)
orignal_shap_values = orignal_explainer.shap_values(X_test)


# %%
def perf_measure(y_actual, y_pred, skip_list=None):
    TP = 0
    FP = 0
    TN = 0
    FN = 0
    for i in range(len(y_pred)):
        if skip_list and i in skip_list:
            continue
        if y_actual[i] >= 0.5 and y_pred[i] >= 0.5:
            TP += 1
        if y_pred[i] >= 0.5 and y_actual[i] <= 0.5:
```

```
            FP += 1
        if y_actual[i] <= 0.5 and y_pred[i] < 0.5:
            TN += 1
        if y_pred[i] <= 0.5 and y_actual[i] >= 0.5:
            FN += 1

    return(TP, FP, TN, FN)

# %%
TP, FP, TN,
FN=perf_measure(Y_test[:,1],new_model.predict(X_test_flattened)[:,1])

# %%
print((TP,FP,TN,FN))

# %%
print("accuracy: %f"%((TP+TN)/(TP+TN+FP+FN)))
print("F1: %f"%((TP)/(TP+0.5*(FP+FN))))
print("DR: %f"%((TP)/(TP+FN)))
print("FPR: %f"%((FP)/(TN+FP)))

# %%
X_test_ex=shap.sample(X_test_flattened,nsamples=1000)
```

```
# %%
Y_test_ex=new_model.predict(X_test_ex)


# %%
print(Y_test_ex[1])


# %%
field_names={
    0: "IP_ver_bit_3", 1: "IP_ver_bit_2", 2: "IP_ver_bit_1", 3: "IP_ver_bit_0",
    4: "IP_header-len_bit_3", 5: "IP_header-len_bit_2", 6: "IP_header-len_bit_1",
7: "IP_header-len_bit_0",
    8: "IP_diff-service_bit_7", 9: "IP_diff-service_bit_6", 10: "IP_diff-
service_bit_5", 11: "IP_diff-service_bit_4", 12: "IP_diff-service_bit_3", 13: "IP_diff-
service_bit_2", 14: "IP_diff-service_bit_1", 15: "IP_diff-service_bit_0",
    16: "IP_total-len_bit_15", 17: "IP_total-len_bit_14", 18: "IP_total-len_bit_13",
19: "IP_total-len_bit_12", 20: "IP_total-len_bit_11", 21: "IP_total-len_bit_10", 22:
"IP_total-len_bit_9", 23: "IP_total-len_bit_8", 24: "IP_total-len_bit_7", 25: "IP_total-
len_bit_6", 26: "IP_total-len_bit_5", 27: "IP_total-len_bit_4", 28: "IP_total-len_bit_3",
29: "IP_total-len_bit_2", 30: "IP_total-len_bit_1", 31: "IP_total-len_bit_0",
    32: "IP_id_bit_15", 33: "IP_id_14", 34: "IP_id_13", 35: "IP_id_12", 36:
"IP_id_11", 37: "IP_id_10", 38: "IP_id_9", 39: "IP_id_8", 40: "IP_id_7", 41: "IP_id_6",
42: "IP_id_5", 43: "IP_id_4", 44: "IP_id_3", 45: "IP_id_2", 46: "IP_id_1", 47: "IP_id_0",
```

48: "IP_flags_reserved", 49: "IP_flags_no-frag", 50: "IP_flags_more-frag",

51: "IP_frag-offset_bit_12", 52: "IP_frag-offset_bit_11", 53: "IP_frag-offset_bit_10", 54: "IP_frag-offset_bit_9", 55: "IP_frag-offset_bit_8", 56: "IP_frag-offset_bit_7", 57: "IP_frag-offset_bit_6", 58: "IP_frag-offset_bit_5", 59: "IP_frag-offset_bit_4", 60: "IP_frag-offset_bit_3", 61: "IP_frag-offset_bit_2", 62: "IP_frag-offset_bit_1", 63: "IP_frag-offset_bit_0",

64: "IP_ttl_bit_7", 65: "IP_ttl_bit_6", 66: "IP_ttl_bit_5", 67: "IP_ttl_bit_4", 68: "IP_ttl_bit_3", 69: "IP_ttl_bit_2", 70: "IP_ttl_bit_1", 71: "IP_ttl_bit_0",

72: "IP_protocol_bit_7", 73: "IP_protocol_bit_6", 74: "IP_protocol_bit_5", 75: "IP_protocol_bit_4", 76: "IP_protocol_bit_3", 77: "IP_protocol_bit_2", 78: "IP_protocol_bit_1", 79: "IP_protocol_bit_0",

80: "IP_header-chksum_bit_15", 81: "IP_header-chksum_bit_14", 82: "IP_header-chksum_bit_13", 83: "IP_header-chksum_bit_12", 84: "IP_header-chksum_bit_11", 85: "IP_header-chksum_bit_10", 86: "IP_header-chksum_bit_9", 87: "IP_header-chksum_bit_8", 88: "IP_header-chksum_bit_7", 89: "IP_header-chksum_bit_6", 90: "IP_header-chksum_bit_5", 91: "IP_header-chksum_bit_4", 92: "IP_header-chksum_bit_3", 93: "IP_header-chksum_bit_2", 94: "IP_header-chksum_bit_1", 95: "IP_header-chksum_bit_0",

96: "UDP_src-port_bit_15", 97: "UDP_src-port_bit_14", 98: "UDP_src-port_bit_13", 99: "UDP_src-port_bit_12", 100: "UDP_src-port_bit_11", 101: "UDP_src-port_bit_10", 102: "UDP_src-port_bit_9", 103: "UDP_src-port_bit_8", 104: "UDP_src-port_bit_7", 105: "UDP_src-port_bit_6", 106: "UDP_src-port_bit_5", 107: "UDP_src-

port_bit_4", 108: "UDP_src-port_bit_3", 109: "UDP_src-port_bit_2", 110: "UDP_src-port_bit_1", 111: "UDP_src-port_bit_0",

112: "UDP_dst-port_bit_15", 113: "UDP_dst-port_bit_14", 114: "UDP_dst-port_bit_13", 115: "UDP_dst-port_bit_12", 116: "UDP_dst-port_bit_11", 117: "UDP_dst-port_bit_10", 118: "UDP_dst-port_bit_9", 119: "UDP_dst-port_bit_8", 120: "UDP_dst-port_bit_7", 121: "UDP_dst-port_bit_6", 122: "UDP_dst-port_bit_5", 123: "UDP_dst-port_bit_4", 124: "UDP_dst-port_bit_3", 125: "UDP_dst-port_bit_2", 126: "UDP_dst-port_bit_1", 127: "UDP_dst-port_bit_0",

128: "UDP_len_bit_15", 129: "UDP_len_bit_14", 130: "UDP_len_bit_13", 131: "UDP_len_bit_12", 132: "UDP_len_bit_11", 133: "UDP_len_bit_10", 134: "UDP_len_bit_9", 135: "UDP_len_bit_8", 136: "UDP_len_bit_7", 137: "UDP_len_bit_6", 138: "UDP_len_bit_5", 139: "UDP_len_bit_4", 140: "UDP_len_bit_3", 141: "UDP_len_bit_2", 142: "UDP_len_bit_1", 143: "UDP_len_bit_0",

144: "UDP_chksum_bit_15", 145: "UDP_chksum_bit_14", 146: "UDP_chksum_bit_13", 147: "UDP_chksum_bit_12", 148: "UDP_chksum_bit_11", 149: "UDP_chksum_bit_10", 150: "UDP_chksum_bit_9", 151: "UDP_chksum_bit_8", 152: "UDP_chksum_bit_7", 153: "UDP_chksum_bit_6", 154: "UDP_chksum_bit_5", 155: "UDP_chksum_bit_4", 156: "UDP_chksum_bit_3", 157: "UDP_chksum_bit_2", 158: "UDP_chksum_bit_1", 159: "UDP_chksum_bit_0",

160: "DNS_id_bit_15", 161: "DNS_id_bit_14", 162: "DNS_id_bit_13", 163: "DNS_id_bit_12", 164: "DNS_id_bit_11", 165: "DNS_id_bit_10", 166: "DNS_id_bit_9", 167: "DNS_id_bit_8", 168: "DNS_id_bit_7", 169: "DNS_id_bit_6", 170:

"DNS_id_bit_5", 171: "DNS_id_bit_4", 172: "DNS_id_bit_3", 173: "DNS_id_bit_2", 174: "DNS_id_bit_1", 175: "DNS_id_bit_0",

176: "DNS_flags_response", 177: "DNS_flags_opcode_bit_3", 178: "DNS_flags_opcode_bit_2", 179: "DNS_flags_opcode_bit_1", 180: "DNS_flags_opcode_bit_0", 181: "DNS_flags_reserved_bit_*", 182: "DNS_flags_truncated", 183: "DNS_flags_recursion", 184: "DNS_flags_reserved_bit_**", 185: "DNS_flags_z", 186: "DNS_flags_AD", 187: "DNS_flags_non-auth", 188: "DNS_flags_reserved_bit_3", 189: "DNS_flags_reserved_bit_2", 190: "DNS_flags_reserved_bit_1", 191: "DNS_flags_reserved_bit_0",

192: "DNS_questions_bit_15", 193: "DNS_questions_bit_14", 194: "DNS_questions_bit_13", 195: "DNS_questions_bit_12", 196: "DNS_questions_bit_11", 197: "DNS_questions_bit_10", 198: "DNS_questions_bit_9", 199: "DNS_questions_bit_8", 200: "DNS_questions_bit_7", 201: "DNS_questions_bit_6", 202: "DNS_questions_bit_5", 203: "DNS_questions_bit_4", 204: "DNS_questions_bit_3", 205: "DNS_questions_bit_2", 206: "DNS_questions_bit_1", 207: "DNS_questions_bit_0",

208: "DNS_answer-RR_bit_15", 209: "DNS_answer-RR_bit_14", 210: "DNS_answer-RR_bit_13", 211: "DNS_answer-RR_bit_12", 212: "DNS_answer-RR_bit_11", 213: "DNS_answer-RR_bit_10", 214: "DNS_answer-RR_bit_9", 215: "DNS_answer-RR_bit_8", 216: "DNS_answer-RR_bit_7", 217: "DNS_answer-RR_bit_6", 218: "DNS_answer-RR_bit_5", 219: "DNS_answer-RR_bit_4", 220:

```
"DNS_answer-RR_bit_3", 221: "DNS_answer-RR_bit_2", 222: "DNS_answer-
RR_bit_1", 223: "DNS_answer-RR_bit_0",
        224: "DNS_authority-RR_bit_15", 225: "DNS_authority-RR_bit_14", 226:
"DNS_authority-RR_bit_13", 227: "DNS_authority-RR_bit_12", 228: "DNS_authority-
RR_bit_11", 229: "DNS_authority-RR_bit_10", 230: "DNS_authority-RR_bit_9", 231:
"DNS_authority-RR_bit_8", 232: "DNS_authority-RR_bit_7", 233: "DNS_authority-
RR_bit_6", 234: "DNS_authority-RR_bit_5", 235: "DNS_authority-RR_bit_4", 236:
"DNS_authority-RR_bit_3", 237: "DNS_authority-RR_bit_2", 238: "DNS_authority-
RR_bit_1", 239: "DNS_authority-RR_bit_0",
        240: "DNS_additional-RR_bit_15", 241: "DNS_additional-RR_bit_14", 242:
"DNS_additional-RR_bit_13", 243: "DNS_additional-RR_bit_12", 244:
"DNS_additional-RR_bit_11", 245: "DNS_additional-RR_bit_10", 246:
"DNS_additional-RR_bit_9", 247: "DNS_additional-RR_bit_8", 248: "DNS_additional-
RR_bit_7", 249: "DNS_additional-RR_bit_6", 250: "DNS_additional-RR_bit_5", 251:
"DNS_additional-RR_bit_4", 252: "DNS_additional-RR_bit_3", 253: "DNS_additional-
RR_bit_2", 254: "DNS_additional-RR_bit_1", 255: "DNS_additional-RR_bit_0",
    }
    feature_names=[]
    for i in range(4):
        for j in range(256):
            feature_names.append("pkt-"+str(i)+"_"+field_names[j])

    # %%
```

```
explainer = shap.DeepExplainer(new_model, X_test_ex)



# %%



# %%
shap_values=explainer.shap_values(X_test_ex)

# %%
print(len(shap_values[0][0]))

# %%
def ranked_high_low(nparray,n_ranks=50):
    sorted_array=sorted(nparray)
    maxs=sorted_array[-n_ranks:]
    mins=sorted_array[:n_ranks]
    max_indices=[]
    min_indices=[]
    for i in range(n_ranks-1,-1,-1):
        max_indices.append(np.where(nparray==maxs[i])[0][0])
    for i in range(0,n_ranks,1):
        min_indices.append(np.where(nparray==mins[i])[0][0])
```

```
    return max_indices, min_indices
summed_shap=sum(shap_values[0])
max_ind, min_ind = ranked_high_low(summed_shap,n_ranks=50)
print(min_ind)

# %%
def ranked(nparray):
    sorted_array=sorted(nparray)
    return sorted_array

# %%
print(len(shap_values[0][1]))

# %%
vals = np.abs(shap_values[0]).mean(0)

feature_importance = pd.DataFrame(list(zip(feature_names,
vals)),columns=['col_name','feature_importance_vals'])
feature_importance.sort_values(by=['feature_importance_vals'],ascending=False,
inplace=True)

#display(feature_importance)
```

```
        #with pd.option_context('display.max_rows', None, 'display.max_columns',
None):  # more options can be specified also
        #    print(feature_importance)
        feature_importance.head(10)

        # %%
        shap.decision_plot(explainer.expected_value.numpy()[1],shap_values[1][0,:],X_t
est_ex[0,:].astype("float64"),feature_names=feature_names)

        # %%
        shap.decision_plot(explainer.expected_value.numpy()[1],
shap_values[1][0:99],X_test_ex[0,:].astype("float64"), feature_names=feature_names)

        # %%
        min_feature = [ feature_names[_] for _ in min_ind]
        max_feature = [ feature_names[_] for _ in max_ind]
        print(min_feature)
        print(max_feature)

        # %%
        print(explainer.expected_value[0].numpy())

        # %%
```

```
shap.initjs()

shap.force_plot(explainer.expected_value[0].numpy(),
shap_values[1][2,max_ind], features = max_feature,matplotlib=True,text_rotation=15)


# %%

shap.force_plot(explainer.expected_value[0].numpy(), shap_values[1][0], features
= max_feature,matplotlib=True,text_rotation=15)


# %%

shap.force_plot(explainer.expected_value[0].numpy(), shap_values[1][0], features
= feature_names,text_rotation=15,matplotlib=True,)


# %%

shap.force_plot(explainer.expected_value[0].numpy(),
shap_values[1][0,min_ind], features = min_feature,matplotlib=True,text_rotation=15)


# %%

shap.force_plot(explainer.expected_value[0].numpy(), shap_values[1], features =
feature_names,text_rotation=15)


# %%

shap.plots._waterfall.waterfall_legacy(explainer.expected_value[0].numpy(),
shap_values[1][0],feature_names = feature_names,max_display=20)
```

```
# %%

shap.summary_plot(shap_values[1],features=feature_names)


# %%

shap.image_plot(orignal_shap_values, X_test[1:5])


# %%

shap.plots.beeswarm(shap_values[0])
```