University of Padova

Department of Mathematics "Tullio Levi-Civita"

Master Thesis in Cybersecurity

Ransomware detection with Machine Learning algorithms using file segments and statistics

Supervisor
Prof. Mauro Conti
University of Padova

Co-supervisor
PhD. Alberto Giaretta
Örebro University

Master Candidate
Emanuele Miotto

Student ID
2023575

Academic Year

2023-2024

“Of what there is, nothing is missing.”
— Antonio Peruzzo, My GrandFather

Abstract

The growing danger posed by ransomware has been a significant concern for both the public and private sectors. The emergence of new strains of this malware has outpaced the development of effective defense mechanisms. Despite the numerous proposed frameworks that employ static and dynamic analysis, these approaches frequently prove ineffective in the face of advanced obfuscation and evasion techniques.

One common characteristic among different ransomware strains is the need to encrypt the filesystem data at some point. The bytes' distribution of encrypted files seems random, while in normal files it tends to be more structured. By measuring such unpredictability through statistical tools, it is possible to leverage this characteristic and distinguish between encrypted and normal files. One of the metrics used to perform this task is the Shannon Entropy.

Researchers tend to compute the Entropy of the bytes' distribution using the entire file, which is not precise, slow, and requires a lot of resources. To overcome these limits, Davies et al. [1] proposed the use of only a fixed segment at the start of the file, called the header. Given the promising results of their ransomware classification method, it seems that computing the files' header Entropy provides relevant information to successfully deploy a working defense mechanism.

However, computing the Entropy of a bytes sequence, whether for the entire file or only the header, is prone to Entropy neutralization techniques. Such attacks aim to reduce the Entropy of the encrypted file by encoding it in a different format, for example, Base64. Various works have explored sophisticated neutralization strategies, and over the years it has become clear that if a defense mechanism uses some form of Entropy values, its performance needs to be tested against such techniques. Among past works, only two Entropy-based ransomware detection methods proposed by Lee et al. [2] and Venturini et al. [3] included such verification in their proposals, leaving all the others potentially vulnerable.

By collecting small fixed-length segments of the files, this thesis proposes a lightweight, fast, and reliable ransomware detection method. The proposed defense mechanism uses only small portions of the files, from which Entropy or the Differential Areas (between real and ideal file's Entropy graphs) are computed and provided to a machine learning algorithm. The use of such features allows for effectively distinguishing between ransomware-encrypted files and legitimate files, and requires very few system resources. To strengthen the feature extraction process and make it more resistant to Entropy tampering, three additional random file segments selection strategies were implemented.

Unlike past works, each feature, machine learning algorithm, and feature extraction strategy were tested against different Entropy neutralization techniques to highlight which combination is the most resilient against such attacks. To do so, the ransomware headers were tampered with to lower their Entropy, and the models were tested once more. This allows the development of an Entropy-based ransomware detection method capable of adapting to both known ransomware strains and future ransomware designed to neutralize their header Entropy values.

Contents

Listing of figures

Listing of tables

Listing of acronyms

AES	Advanced Encryption Standard
Entropy	Shannon Entropy
DA	Differential Area
DAA	Differential Area Analysis
2F	Two Fragments
3F	Three Fragments
4F	Four Fragments
SVM	Support Vector Machine
SVC	Support Vector Classification
RF	Random Forest
NN	Neural Network
H	Header
H+RS	Header and a Random Segment
H+2RS	Header and Two Random Segments
H+3RS	Header and Three Random Segments
Entropy+DA	Entropy and Differential Area combined

1 Introduction

From 2005 and on, ransomware have been a growing threat, damaging private companies and public institutions for millions and millions of dollars. A report from Chainanalysis (https://www.chainalysis.com/blog/ransomware-2024/) states that the capital moved by ransomware attacks in 2023 alone is near 1 billion dollars.

Over the years, cybersecurity companies and researchers developed different defense mechanisms against this threat, but it is difficult to keep up with the quick evolving nature of ransomware.

However, independently of the techniques they use and their complexity, all the different ransomware strains need to encrypt filesystem data at some point during their execution. When a file is encrypted, its bytes' distribution changes from a predictable sequence due to the file type and content, to a seemingly random one. Leveraging this bytes' distribution change, it is possible to build a ransomware detection method able to ideally adapt to unforeseen ransomware strains. The logic is simple: detect if a file is being encrypted or not, and alert the user if it is.

Measuring the randomness of bytes to tell apart ransomware-encrypted files from legitimate ones is common in ransomware detection. This typically involves using statistical tools on the file bytes' distribution and checking the resulting metric. Tools like the Arithmetic Mean, Chi-Square, and Shannon Entropy are often used. In this thesis, I argue that in real-world situations, waiting for the whole file to be encrypted before detecting ransomware is not efficient. Also, calculating these metrics on thousands or tens of thousands of bytes can be resource-intensive.

Fortunately, Davies et al. [1] obtained promising results proposing a ransomware classification analysis based on Shannon Entropy of only the files' header, a fixed segment at the start of the file.

Davies et al. [1] work, and in general all the ransomware detection strategies based on Entropy, are vulnerable to Entropy neutralization techniques, as proved by McIntosh et al. [5], Lee et al. [6, 7], Venturini et al. [3] and Bang et al. [8].

This thesis innovative approach compared to previous methodologies is the use of the Shannon Entropy computed on small fixed-length segments paired with machine learning, for a quick and light defense mechanism. The use of machine learning allows the model to be trained on different neutralization techniques and eventually adapt to their presence, compensating most Entropy-based ransomware detection methods vulnerability.

This thesis is structured as follows. After this Introduction, Chapter 2 gives some background information to better understand the concepts used throughout this thesis. Chapter 3 discusses the methodology used to structure the experiment, starting with the initial hypotheses, followed by how the experiment was designed and implemented in order to verify them. Lastly, Chapter 4 reports all the obtained results and proofs.

More specifically, in the Introduction, in Section 1.1 ransomware is briefly introduced, detailing its origin and evolution from the initial emergence in 1989 to becoming a significant threat post-2005. Then, the two principal types of ransomware are explained, and the different ransomware analyses used to defend against such threats are explored.

In Section 1.2, various past proposed detection methodologies are briefly discussed, to provide context around the objective of this thesis. In Subsection 1.2.1 the reader can find a general definition of encryption and which tools are used to measure data predictability.

In Section 1.3 the most relevant ransomware detection researches are reported more in detail, with a focus on the ones using some form of Entropy computations. Next, past ransomware defense mechanisms involving the use of machine learning are discussed, and the works highlighting the vulnerability tied to the use of Entropy features are described. In Subsection 1.3.1, some works in a different but related field are introduced.

1.1 Ransomware

The term "ransomware" is derived from the fusion of "ransom" and "malware." It refers to malicious software that typically operates by demanding payment in exchange for restoring functionality that it has compromised [9].

Although the first ransomware was used in 1989 by Joseph Popp, who distributed a program named ”AIDS” via floppy disks, it was not until after 2005 that ransomware attacks began to emerge as a significant threat [10]. Over time, the targets of these attacks have evolved, shifting from individual users to larger enterprise systems, where the potential for damage —and thus the ransom demands— can be substantially higher.

Ransomware attacks can be classified into two principal categories: Locker-Ransomware and Crypto-Ransomware. Locker-Ransomware functions by locking the victim's device, preventing access to the PC in exchange for money. Until the ransom is paid, it typically leaves the underlying system and files intact. In contrast, Crypto-Ransomware employs encryption algorithms to cipher the victim's data, rendering them inaccessible until the victim pays for the decryption key [11]. Locker-Ransomware is effective only when it employs social engineering tactics to pressure victims into paying, whereas the direct attack on data by Crypto-Ransomware makes them a far more dangerous threat.

A pivotal moment in the evolution of ransomware attacks occurred in May 2017 with the emergence of the Crypto-Ransomware WannaCry [12]. This ransomware attacked major industries, healthcare facilities, and government institutions on a global scale. It was a significant shift from its predecessors, which principally targeted private citizens and requested generally small fees. In the span of just a few days, WannaCry infected over 230,000 Microsoft Windows systems across 150 countries [13].

After WannaCry, a series of similarly infamous ransomware attacks followed. NotPetya, which was specifically targeted at Ukrainian businesses and government entities, BadRabbit, which was known for its spread through enterprise networks, and SamSam, which focused on healthcare and government institutions [12].

In the modern interconnected world, the successful deployment of Crypto-Ransomware can have severe consequences, as digital information is fundamental to the operations of both public and private organizations. The WannaCry attack, along with others of similar nature [11], serve as a clear signal of the potential impact that a ransomware can have on essential infrastructure and services. To mitigate the potential for harm posed by Crypto-Ransomware, increased levels of vigilance and robust cybersecurity measures are required.

1.2 Ransomware Detection

Security researchers have mainly directed their attentions toward the category of Crypto-Ransomware due to its effectiveness and efficiency. This intensified focus has resulted in the emergence of three principal detection methodologies.

Signature or static analysis is a technique employed in cybersecurity to examine executable code prior to its execution, with the objective of identifying known malicious patterns or sequences of bytes. The key aspects of signature/static analysis include:

- **Pattern matching**: Primary method employed in the identification of malware. This involves the scanning of executable code for specific sequences of bytes that match a database of known malware signatures. The database is updated on a continuous basis with new signatures as new malware/ransomware is discovered.

- **Hashing**: The application of hash functions to executable files results in the generation of unique identifiers (hashes) for each file. Hashes are then compared against a list of hashes that are known to be associated with malicious files. The most commonly employed hashing algorithms include MD5, SHA-1, and SHA-256.

- **Heuristic analysis**: It involves examining the code for suspicious features or behaviors that are commonly observed in malware, even if there is no exact match with a known signature. Such an examination may include the identification of anomalous instructions, the use of obfuscation techniques, and the presence of previously identified exploitation patterns.

- **Disassembly**: The executable code is converted into assembly language in order to facilitate a more detailed examination of its structure and operations. Subsequently, analysts are able to manually inspect the disassembled code in order to identify any potentially malicious instructions or logic.

- **String Analysis**: Search inside the executable for specific strings of text that are commonly found in malware, such as URLs, IP addresses, suspicious command-line arguments, or known malicious function names.

- **Control Flow Graph (CFG) Analysis**: The process of examining the control flow of a program in order to gain an understanding of its execution pathways and to identify potential anomalies or suspicious branching structures. This approach can help in identifying complex malware that may employ sophisticated control flow techniques to evade detection.

- **Static Behavioral Indicators**: Identify static indicators of malicious behavior, such as the presence of embedded scripts, macros, or other executable content within documents or other file types. This may also include the presence of known exploit payloads or shellcode.

Behavioral or dynamic analysis is a method of observing the actions and patterns of processes in order to identify any suspicious activities or potential threats.

This type of analysis is crucial, as it allows for the identification and prevention of malware infections and other cyberattacks. By comparing the behavior of processes against known attack profiles, the analysis can determine whether a process may be malicious.

The techniques for behavioral analysis can be classified into four broad categories:

- **File-based** detection: Process of identifying and analyzing specific files within a system by searching for suspicious behaviors, such as unusual file creation, modification, deletion, or access patterns, which may indicate the presence of malware.

- **System-based Behavior** detection: This category includes monitoring system calls, process creation and termination, as well as tracking the behavior of system services that could indicate an attack.

- **Resource-based Behavior** detection: This technique entails the observation of the utilization of system resources, including the central processing unit (CPU), random-access memory (RAM), storage media, and network bandwidth. It can be reasonably assumed that unusual or excessive resource usage may be indicative of malicious activity

- **Connection-based Behavior** detection: This approach employs the monitoring of network connections and communication patterns in order to identify any suspicious activities. Some indicators could be: incoming and outgoing traffic, unusual connection attempts and known malicious IP addresses or domains [14, 15, 16, 17].

Hybrid analysis is a method of ransomware analysis that combines both static and dynamic analysis techniques. The integration of both approaches provides a more comprehensive view of the ransomware and its capabilities.

Over time, new defensive mechanisms have been developed, in the form of complex frameworks and algorithms that integrate various techniques from both static and dynamic analysis. However, ransomware has evolved alongside these defensive mechanisms, deploying increasingly sophisticated attacks that incorporate polymorphism, encryption, multiprocess executables, and other techniques to evade detection.

Polymorphism allows ransomware to change its code with each infection, making signature-based detection methods less effective. Encryption ensures that the ransomware's payload is hidden from analysis until it is executed. Multiprocess executables complicate detection by distributing malicious activities across several processes [14, 17].

These advancements in ransomware imply that the majority of existing detection algorithms and frameworks struggle to adapt to different strains of ransomware, particularly those with novel execution patterns. Consequently, detection systems are prone to high false positive rates, where legitimate files are erroneously identified as malicious, and false negative rates, where actual ransomware goes undetected.

To compensate the deficiencies and enhance detection accuracy, these systems may necessitate the gathering of extensive data, which may result in significant overhead on machine resources such as CPU and RAM [18, 19, 20, 21, 22, 23, 24, 25, 26, 27].

However, even the most complex ransomware variants have a common requirement: they need to encrypt data and write it to the disk, eventually [19]. The stronger the encryption, the more random the resulting file data distribution will be. Recognizing this characteristic, past security researchers have used various statistical metrics to measure the randomness of encrypted files, distinguishing them from normal files. Encryption and the metrics to measure data predictability are discussed in Subsection 1.2.1. Among the different possibilities, some metrics widely used are different forms of the Entropy. In particular, high Entropy values are typically indicative of the presence of an eventual ransomware in action [15, 19, 28, 29, 22, 18, 30, 21, 22, 23, 24, 25, 26, 27].

Some defensive mechanisms use the Shannon Entropy and aim to quantify the randomness of an entire file bytes' distribution in order to identify encrypted data [18, 19, 20, 21, 22, 31, 24]. However, this approach requires a significant amount of computational resources, specifically for large files, and is not effective in differentiating between file types where byte distribution seems almost as random as encrypted files, such as compressed or PDF files [30].

To address these limitations, Davies et al. [1] proposed to focus on a fixed-length segment of the file, the header, where the difference in Entropy between files should be more prominent. This is in general due to the presence of metadata bytes, a descriptive sequence of bytes tied to the file type. This, paired with the removal of the dependence of the metric on the file length, could improve and fasten the computations. Their findings looked promising, achieving impressive results in terms of accuracy and F1 score on the considered dataset.

Kim et al. [32] proposed a technique based on a similar idea. They combined the computed Entropy of the header with another feature derived from segment byte probabilities and fed these features into a neural network (NN), a support vector machine (SVM), and a threshold-based algorithm. Their approach yielded notable results, obtaining good detection accuracy scores.

Despite the advantages of computing Entropy for fixed-size file segments, both segment and entire file Entropy analyses are vulnerable to Entropy neutralization techniques. If a ransomware implemented such methods, it would be able to standardizes data distribution in order to lower the Entropy of the encrypted files, rendering the analyses ineffective.

McIntosh et al. [17] demonstrated that by using Base64 encoding and distributed partial encryption. Such algorithms manipulate the bytes' distribution, lowering the Entropy and bypassing Entropy-based defenses. Lee et al. [6] improved on this by developing an algorithm that selects encoding types to match original file Entropy, enhancing attack performance.

Davies et al. [33] tested 53 Entropy computation methods, concluding that pure mathematical techniques like Shannon Entropy are ineffective for distinguishing ransomware-encrypted files from compressed files due to their similarities. They suggested combining Entropy calculations with byte correlation coefficients and incorporating encoding detection.

Lee et al. [2] proposed machine learning algorithms to counteract ransomware using encoding to neutralize Entropy, achieving 98% accuracy. Then, in a subsequent work [7], they developed a format-preserving encryption neutralization method which should be stronger against machine learning detection techniques.

Bang et al. [8] introduced Entropy sharing, a lightweight method that integrates with cryptographic functions, masking high Entropy blocks and resisting reversing techniques thanks to a secret parameter called ”order of shares”.

Venturini et al. [3] showed that tampering with file header bytes could defeat the Differential Area Analysis (DAA) proposed by Davies et al. [1]. They suggested three mitigations against header tempering techniques. Such mitigations are based on random sampling along the files to reduce DAA's dependency on the header distribution.

This thesis exploits the effectiveness of using Shannon Entropy or Differential Area (relative to an ideal random file) computed only from the file header bytes as features for various machine learning algorithms. Contrary to other methods which either use a lot of features or slowly compute the entropy of the entire file, extrapolating only the header features is a lightweight procedure. Fewer operations mean less overhead on machine resources, quick ransomware detection and more files saved. Subsequently, it will simulate an Entropy neutralization attack where the header is tampered, recollect the Entropy features and test them again using the same algorithms. To compensate the vulnerability of collecting the features only from the header, a sample strategy based on selecting random file segments is implemented. Similarly to the mitigations proposed by Venturini et al. [3]), two, three and four segments were randomly sampled along the files. The performances of the machine learning algorithms with the new features collected are then reevaluated. By doing so, this thesis aims to test if segments' Entropy-based machine learning detection methods work against known ransomware strains and also against Entropy neutralization strategies.

1.2.1 Encryption and Randomness

The objective of encryption algorithms is to produce a sequence, known as ciphertext, from which it is extremely difficult to deduce the original input, or plaintext, without the appropriate decryption key.

To achieve this, one requirement is that the ciphertext and the plaintext data distribution have to be statistically independent as much as possible and also that the ciphertext data should be uniformly distributed.

The stronger the encryption algorithm is, the more uniform and independent of the input the ciphertext data are. The uniform distribution ensures that an attacker can only make random guesses about the original values, significantly increasing the difficulty of breaking the encryption.

The primary distinctions between normal binary files and encrypted binary files is that the bytes' distribution in normal files tends to be more structured and repeated compared to encrypted files.

Several statistical metrics can be employed to measure the randomness of a bytes' distribution:

- **Arithmetic Mean**: This metric involves summing the individual byte values of a file and dividing by the total number of bytes. Byte values range from 0 to 255, so an arithmetic mean close to 127.5 suggests that the data is more random. This is because, in a perfectly random distribution, the byte values would be evenly spread across the entire range.

- **Chi-Square**: The Chi-Square test compares the actual distribution of byte values to a model distribution. For randomness detection, the model distribution is typically the uniform distribution, where each byte value is equally likely. A significant deviation from the uniform distribution can indicate non-randomness.

- **Monte Carlo**: This method involves repeatedly sampling the data uniformly and performing a statistical analysis on the time-averaged results. The Monte Carlo method can provide insights into the overall randomness of the data by assessing how well the sample data fits a random model.

- **Serial Byte Correlation Coefficient**: This coefficient measures the relationship between consecutive byte values. In a random sequence, there should be little to no correlation between successive bytes. High correlation values suggest a lack of randomness.

- **Shannon Entropy**: In information theory, Entropy is a measure of the uncertainty associated with a given input. Shannon Entropy quantifies the amount of information contained within each byte. Higher Entropy values indicate greater randomness and more information content per byte, which is characteristic of encrypted data [34].

1.3 Related Works

Previous works for ransomware detection that used Entropy have either applied Entropy analysis to the entire file [18, 19, 20, 21, 22, 31, 24] or, if they focused on a segment header [1, 32], they did not propose a mitigation for the eventual tampering of the analyzed file segment, except for Venturini et al. [3] and Lee et al. [2]. This oversight has left a significant gap in the robustness of these detection methods.

Computing the Entropy of an entire file has been a longstanding approach, particularly in the field of file identification. For example, Hall et al. [28] successfully demonstrated that English text files typically have an Entropy between 3.25 and 4.5 bits, while compressed files such as zip files exhibit higher Entropy values, generally over 6 bits. This difference in Entropy values helps distinguish between various file types based on their data distribution patterns. However, Zhao et al. [30] identified a major challenge when using full-file Entropy for file type classification: compressed and encrypted data often show similar high Entropy characteristics. This similarity complicates the differentiation between these file types, underscoring the need for further investigation into the applicability of Entropy metrics for these purposes.

In the context of malware detection, Entropy has often been used as one of several features to identify malicious activity. Typically, the Entropy of the entire file is calculated and combined with other features such as the file's magic number, file extension, and behavioral patterns to improve detection accuracy. This approach can be found in a lot of systems. Some of the first technique proposed were ShieldFS [19], DropIT [18] and Unveil [20], among many others. However, it has been shown that, in a way or another, these additional features can be manipulated by attackers to evade detection. In particular, file extension and magic numbers are prone to tempering [35]. Thus, to compensate the additional features vulnerabilities and since computing the Entropy of the entire file can be unreliable, as demonstrated by Zhao et al. [30] and Davies et al. [33], they need to collect and analyze extensive supplementary data which leads to significant overhead in terms of CPU and RAM usage, impacting system performances.

Addressing these limitations, Davies et al. [1] proposed a novel approach that involves analyzing fixed-length segments of a file, specifically from the beginning, referred to as the file header. They hypothesized that the beginning of a file often contains metadata bytes, which would naturally have a lower Entropy compared to the rest of the file. By focusing on these segments, they aimed to reduce computational load and improve detection accuracy. They computed the Entropy of different header lengths, compared it with the Entropy of an ideal file through the use of Differential Area (DA) and tested the obtained Bit-Byte area with different thresholds values. If the DA was low, it meant that the Entropy of the original file was high and close to the random ideal file, therefore a possible ransomware encrypted file. To measure the accuracy of the proposed technique, they compared the computed Bit-Byte areas values with different thresholds. This comparison was called the Differential Area Analysis and is better explained in Section 2.4. With a header length between 128 and 256 bytes and a threshold value between 32 and 56 Bit-Bytes the used metrics (accuracy, f1, precision and recall) were near or over 99.5%. However, not only their method did not account for the possibility of attackers deliberately modifying the ransomware header to lower its Entropy, but they also modified a specific ransomware bytes' distribution (Phobos) by removing all the zeros at the beginning. Both of these scenarios would have resulted in a ransomware with low Entropy, thus evading the proposed detection technique.

In the same year, Lee et al. [21, 22] proposed a ransomware detection technique for cloud storage services. They combined machine learning and Entropy analysis to classify ransomware infected files. In particular, some user's reference files are used by the backup system to extrapolate the Entropy features and train the algorithms. The Entropy features extracted are: the most common value estimate, the collision test estimate, the Markov test estimate, and the compression test estimate. Once the training phase is over, a threshold is extracted and synchronized with the local user machine. Such threshold is used by the user local software client to distinguish between ransomware or legitimate files. Even in the event of a user's system being compromised, the proposed technique can restore original files from backups by identifying the infected files that have been synced to the backup. Thanks to the adaptability given by the tailored threshold parameter computed specifically from each user' files, the analysis results demonstrated that this method offers a high detection rate with low error rates in comparison to existing detection methods. Unfortunately, not only the feature extraction depends on the file format which can be easily spoofed by an attacker [35], but also multiple Entropy values of the entire files are computed, slowing the process significantly.

Subsequently, Fei et al. [23] proposed an introspection based approach to detect Crypto-Ransomware, called RansomSpector. Such detection approach is based on a virtual machine introspection technique, and it resides in the hypervisor layer under the operating system where the ransomware eventually executes. It monitors both the filesystem and the network activities and once both the files access pattern and the network pattern match the precomputed ransomware patterns, it computes the average Entropy of the received written data. Then, if the value is bigger than a threshold, the user is alerted. This method obtains good results, but the used ransomware dataset was small and thus need to be tested on more samples before further evaluating its performances.

Starting from the ransomware analysis proposed by Davies et al. [1], Venturni et al. [3] proved a critical vulnerability against such method, which is to focus only on the file header to compute the Entropy and proposed some strengthen versions that could mitigate the problem. Such alternatives sampled random segments along the file and through an averaging strategy performed classification.

Hsu et al. [24] proposed a ransomware detection approach based on the extraction of different Entropy values and the classification thanks to Support Vector Machine and Support Vector Machine with poly trick. The dataset used was a combination of 1000 encrypted files from four different ransomware (WannaCry, Phobos, GrandCrab and GlobeImposter) and an unspecified number of normal files (DLL, GIF, etc.). Their claim is that thanks to the seventeen different features extracted, to which different type of Entropy computations are included, the SVC with poly trick is able to obtain a detection rate of 92.33% in the worst case. However, due to the missing number of normal files and the high number of features computed from a single file, it is not clear if such system could be used in real life scenarios.

Kim et al. [32] developed a technique similar to that of Davies et al., which also relies on header Entropy, but with an innovative integration of machine learning. Their ransomware classification method was based on the Entropy of the header and another feature derived from segment byte probabilities used as input for various machine learning models. Such model were a neural network (NN), a support vector machine (SVM), and a threshold-based algorithm. This comprehensive approach resulted in impressive outcomes, achieving high detection accuracy scores and demonstrating the effectiveness of their method in identifying anomalies.

Kim et al. [25] proposed a hybrid framework consisting of the combination between decoy deployment and file traversal Entropy. The decoy files are ad-hoc files used to appeal ransomware and generally to monitor suspicious process. File traversal Entropy is the Entropy of each process's file pattern path. Highly random paths, thus high Entropy values, could indicate the indiscriminate access of a ransomware process trying to quickly encrypt the most possible files. By combing these two values they claimed to obtain 99% accuracy against the tested ransomware.

A recently proposed ransomware detection approach by Ayub et al. [26] is called RWArmor. Their defense mechanism collect ransomware static features thanks to different sandbox emulators. This features, including Entropy, are fed into a Random Forest and a Decision tree classifiers which output two probability scores of the input sample being a ransomware. The highest probability is selected to estimate the confidence of a running ransomware.

Generally speaking, in the ransomware detection field, there has been a steady shift towards the use of machine learning algorithms.

In a survey wrote by Razulla et al. [36] which considered 125 papers from 2016 to 2022, 70% of the analyzed ransomware detection mechanisms have used some form of machine learning methods. In particular, 82.6% of these used simpler machine learning techniques, with the more frequent ones being Support Vector Machine, Random Forest, Decision Trees and Logistic Regression. Support Vector Machines (SVM) and Random Forests have been particularly prominent. Random Forests, in particular, have been generally preferred, with a 16.9% out of the 125 papers analyzed by [36], and generally show good performances as shown in a 2019 study by Noorbehbahani et al. [37].

However, the trend seems to be moving in favor of deep learning algorithms like Deep Neural Network, Multi Layer Perceptron, Long-Short Term Memory, Recurrent Neural Network and Convolutional Neural Network. This is confirmed by a recent survey wrote by Ispahany et al. [38] in which, even though they analyzed only the papers from 2018 to 2024 that specifically centered around the Windows operating system, the Deep Learning approaches were used in 41% of the selected papers.

Based on these findings, the selected machine learning algorithms used in this thesis are Support Vector Machine, Random Forest and Neural Network.

Some examples of features used by the machine learning algorithms are the process I/O requests, the system calls, the DLL activities [39], the Assembly codes [39], the network traffic [40] and the API calls [41]. If the reader is interested in a complete list, please refer to the surveys wrote by Razulla et al. [36] and by Ispahany et al. [38].

Notably, RansomWall [31], studies by Lee et al. [21, 22], Hsu et al. [24], Ayub et al. [26] and Kim et al. [32], among others features incorporated also the Entropy. However, in these cases, the Entropy was still calculated over the entirety of the file, which limits their effectiveness given the similitude between compressed and encrypted files and slow the computations.

Therefore, based on the findings of Davies et al. [1] and Kim et al. [32], the selected features for the used machine learning algorithms are derived from the Entropy of only the file segments, enabling a quick and lightweight defense mechanism.

Independently of using a mathematical or an artificial intelligence approach to develop a defense mechanism, if the Entropy is used, the mechanism has a critical vulnerability. If a ransomware was able to modify the encrypted files, header included, to standardize the data distribution and lower the Entropy, the analyses become ineffective.

This weakness was highlighted by multiple studies in recent years, both in the case of defense mechanisms using the Entropy of the entire file and on the specific case of using files' header Entropy.

First it was addressed by the work of McIntosh et al. [5], in which they state that by using Base64-Encoding and Distributed Non-Selective Partial Encryption the Entropy of the encrypted files could be manipulated, and the defense mechanisms based on Entropy computation bypassed.

Subsequently, Lee et al. [6] took the work of McIntosh et al. [5], recognized the potential and improved the attack performances. In particular, they stated that, even if the use of encoding to lower the Entropy of the encrypted files works, some files maintained a clear difference between the legitimate original file and the respective encryption-encoded counterpart. Thus, they developed an algorithm that, once the legitimate file is encrypted, it chooses from different types of encoding based on which encoding algorithm obtains the closest possible Entropy value with respect to the original file Entropy.

To tackle the problem from the root, Davies et al. [33] tested 53 distinct Entropy computation methods and compared their accuracy in distinguishing ransomware encrypted files and legitimate files. The general conclusion was that using pure mathematical techniques, such as Shannon Entropy, computed on the entire file, is not an ideal indicator for identifying Crypto-Ransomware encrypted files. This was principally due to the difficulties on distinguishing encrypted files from compressed or archived files. They found that using the mathematical computations paired with serial byte correlation could improve the results. They also stated that, since the works from McIntosh et al. [5] and Lee et al. [6] were effective in creating ransomware able to bypass Entropy-based defensive mechanism, future works can still use file Entropy features, but should include encoding detection techniques.

The work of Lee et al. [2] focused exactly on this vulnerability. They proposed some machine learning algorithms as countermeasures against ransomware that use encoding after the encryption to neutralize the ciphertext Entropy. Various features were used, Entropy included, and different machine learning algorithm tested. They evaluated the algorithm's ability to distinguish between the same file in different stages, in particular: plaintext, ciphertext, Base64 encoding, optimal encoding, system file. They found that the average accuracy was 98% and stated that using machine learning is an effective way to counter neutralization technologies and detect ransomware correctly.

Another ransomware detection neutralization technique, proposed by Lee et al. [7] focus on the use of more sophisticated neutralization technologies aside encoding which they state that can be detected. Given this motivation, the formulated three requirements for a neutralization method: it must not be decoded, it must support encryption using secret information and the Entropy of the generated ciphertext must be similar to that of a plaintext. To satisfy such conditions, format-preserving encryption was used.

Lastly, Bang et al. [8] stated that previous neutralization techniques required a lot of computations and therefore could be easily predicted. To overcome this limit, they proposed a new concept called Entropy sharing. A method that can be easily integrated with standard cryptographic function, is composed of lightweight operations which mask the high Entropy blocks and cannot be easily nullified without knowing a parameter that they called order of shares.

In the case of only file headers tempering, Venturini et al.[3], given the Differential Area Analysis of file header Entropy proposed by Davies et al., proved how a simple tampering of the header bytes distribution could defeat the proposed Entropy-based detection method. Subsequently, they proposed some mitigations based on the random sampling along the files, compensating the dependency of the DAA from the header distribution.

To the best of my knowledge, only the works from Lee et al. [2] and Venturni et al. [3] tested the proposed Entropy-based method against ransomware detection neutralization technologies. Thus, differently from the past works which use the Entropy as a metric or a feature to propose a defense mechanism and do not test its resilience against such vulnerability, this thesis also uses three neutralization techniques and test the proposed models to strengthen the ransomware detection mechanism.

1.3.1 File Segments Analysis

File fragment classification is a related field of research that focuses on determining the type of file by analyzing only a portion or fragment of it. This method is particularly useful in digital forensics. Techniques used in this field include Entropy analysis, n-gram analysis, statistical analysis and machine learning.

McDaniel and Heydari [35] developed a method that uses byte frequency analysis of a file's header and footer to create a fingerprint for identifying similar file types. Their findings indicated that this approach is viable but should be combined with other methods to improve accuracy. They also suggested that file header/trailer analysis could be used for ransomware detection, requiring only a small sample for successful identification.

Li et al. [42] used n-gram analysis on the first 20 or 200 bytes of a file to determine file types, reporting successful identification with better computational performance compared to whole-file analysis. However, their tests did not include compressed or encrypted files.

Hall et al. [28] noted that certain file types with structured formats, such as GIF or PPT, exhibit varying Entropy levels within different parts of the file. For example, a file with a text-format header and a compressed data body differs from a binary file with random values. To distinguish these structured formats from other files with similar overall Entropy, a sliding window approach to measurement has been evaluated.

Jung and Won [29] analyzed Entropy on file fragments, including headers and trailers, with promising results but limited their study to PDF documents.

2 Background

In this chapter, general knowledge is provided on the topics used in this thesis. In particular, it is explained what the statistical tool Shannon Entropy is and how it is computed in Section 2.1. The Composite Trapezoidal Rule and the DA are also briefly summarized, in Section 2.2 and in Section 2.3, since they are used by the ransomware classification analysis proposed by Davies et al. [1], which is also here reported and explained in Section 2.4. Lastly, the general theory behind the concepts of machine learning is introduced and a quick round down of the used algorithms is performed in Section 2.5.

2.1 Shannon Entropy

The concept of entropy is defined as a measure of randomness or disorder. In the field of information theory, specifically defined by Claude Shannon [34], the entropy of a variable is defined as the average level of uncertainty associated with the potential outcomes of the variable itself. Given a discrete random variable X that can assume values between 1 to n with their relatives probabilities of appearance $p(x_i)$, the Shannon entropy can be expressed using the following formula:

$$H(X) = -\sum_{i=1}^{n} P(x_i) log_2 P(x_i). \tag{2.1}$$

In this thesis, file bytes' distribution are considered. Therefore, the entropy can be seen as a measure of the predictability of the next file's bytes, based on the previous ones. A structured and regular file presents high predictability, hence low entropy, whereas a file composed of random values will exhibit high entropy. Equation 2.1 provide a value measured in bits.

2.2 Composite Trapezoidal Rule

The area under the curve is defined as the region bounded by the function and vertical lines representing the function's bounds and the x-axis. This can be computed by integrating the function in the given interval. If the integral is considered in a definite interval then an approximation of the results can be obtained thanks to the Trapezoidal Rule, which can be seen as the average of the left and right Riemann sums. A technique to better approximate the integral is to partition the original interval into smaller subinterval and then apply the trapezoidal rule to each subinterval. This technique is also known as the Composite Trapezoidal Rule and a graphical representation can be seen at Figure 2.1 and formal definition can be found in [43]. Let [a, b] be the interval of integration with a partition $a = x_0 < x_1 < \quad < x_n = b$. Then the formal Composite Trapezoidal rule is defined as:

$$\oint_a^b f(x)dx \approx \frac{1}{2}\sum_{j=1}^{n}(x_j - x_{j-1})[f(x_{j-1}) + f(x_j)]. \tag{2.2}$$

If the partition is uniformly spaced, i.e. $x_j - x_{j-1} = h$ for all $j \in \{1...n\}$, then the Composite trapezoidal rule is also given by:

$$\oint_a^b f(x)dx \approx \frac{h}{2}\left[f(a) + 2\sum f(x + h_i) + f(b)\right]. \tag{2.3}$$

Since the Differential Area Analysis for ransomware classification, introduced by Davies et al. [1] and briefly explained in Section 2.4, compute the area of functions which domains are discrete with the subintervals evenly spaced apart, the equation used to compute such areas is the Equation 2.3.

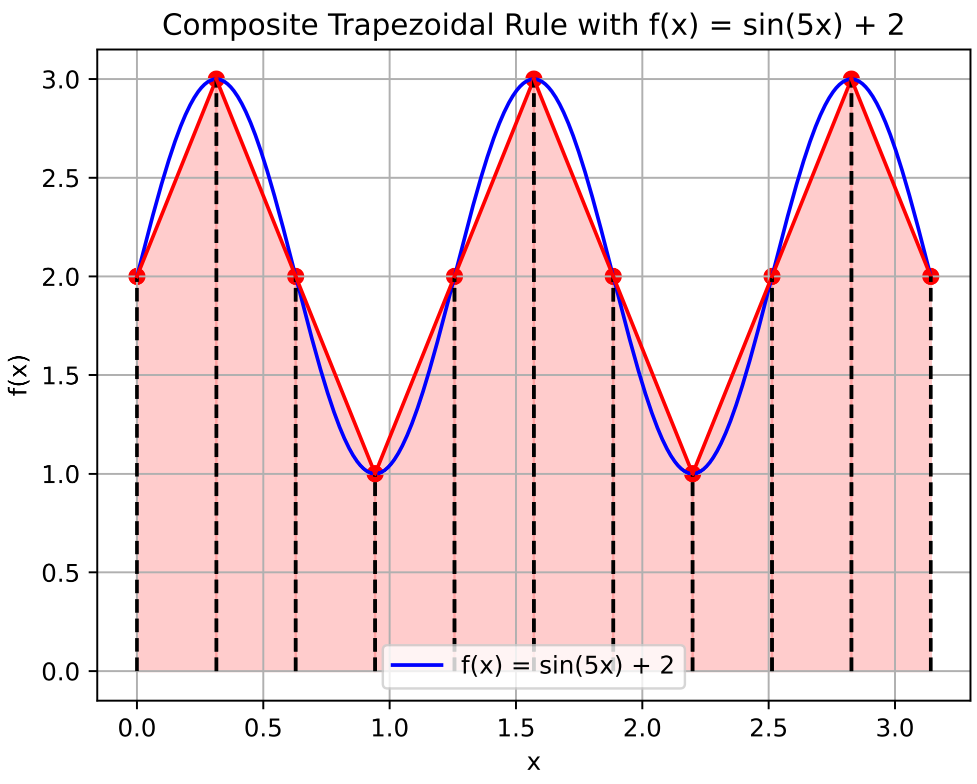


Figure 2.1: Composite Trapezoidal Rule with uniform subintervals

2.3 Differential Area (DA)

The Differential Area (DA) between two functions $f(x)$ and $g(x)$ over an interval $[a, b]$ can be defined as the area of the region bounded by these two functions. Mathematically, it can be expressed as:

$$\text{DA} = \int_a^b |f(x) - g(x)|\, dx$$

.

This integral calculates the total area where the two functions differ, considering the absolute value to ensure all areas between the curves are summed positively.

The differential area between two generic functions, $f(x) = \sin(x) + 1$ and $g(x) = \cos(x)$, over the interval $[0, 2\pi]$ is illustrated in Figure 2.2 as the gray shade.

Since this thesis computes the Difference Areas of discrete Entropy functions, the areas are computed using the equation Equation 2.3 and the DAs are the sum of the absolute values of the difference between the areas for each subinterval.

2.4 Ransomware Classification with Differential Area Analysis (DAA)

Differential Area Analysis (DAA) is a method that can be used for ransomware classification and was firstly proposed by Davies et al. [1].

As the name suggests, it is based on the Differential Area (DA) defined in Section 2.3. In the particular case proposed by Davies et al. [1] for ransomware detection, the two considered functions are the same, which can informally be called the Entropy function. The Entropy function is a map that given a byte sequence compute the corresponding entropy value, in bits, as defined in Section 2.1. The functions are the same, but the output values are different because the inputs are different. In particular, one input is the selected file to eventually classify as legitimate or ransomware, the other is a reference ideal file composed of random values. An example can be seen in Figure 2.5. The reason behind comparing the files to and random file is because encrypted files bytes' distribution is ideally uniform, thus the encrypted bytes are closer to a seemingly random sequence than a legitimate more structured file bytes distribution.

Davies et al. [1] focused only on the header of the file, analyzing header lengths from 8 bytes up to a maximum of 256 bytes, and their approach is straightforward [1]:

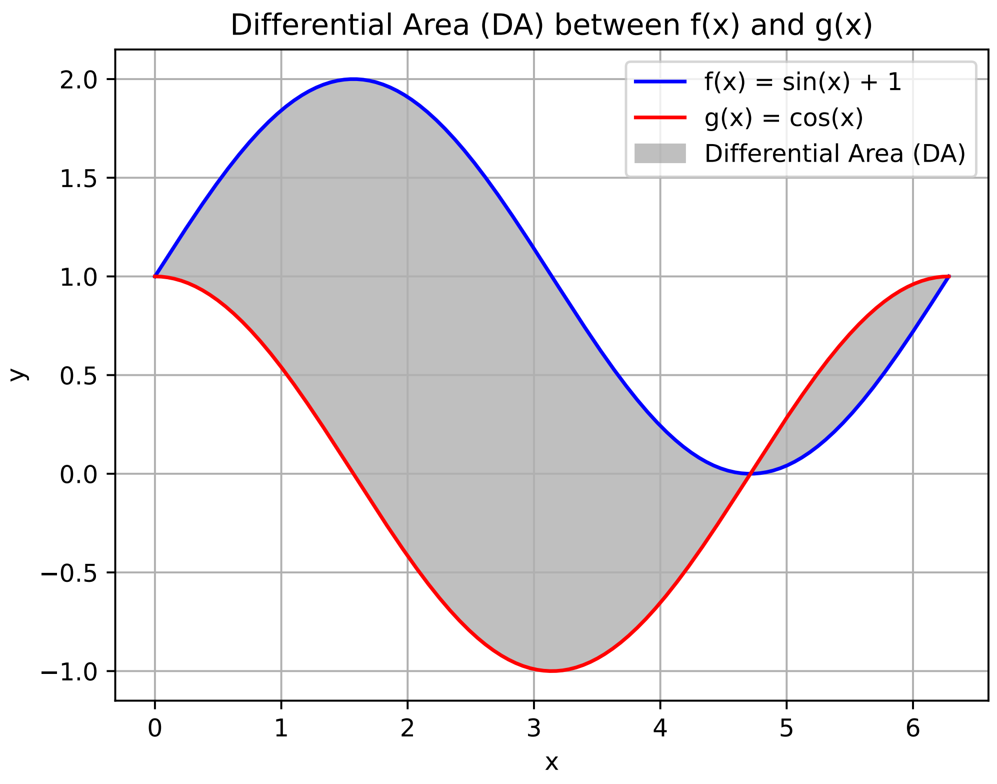
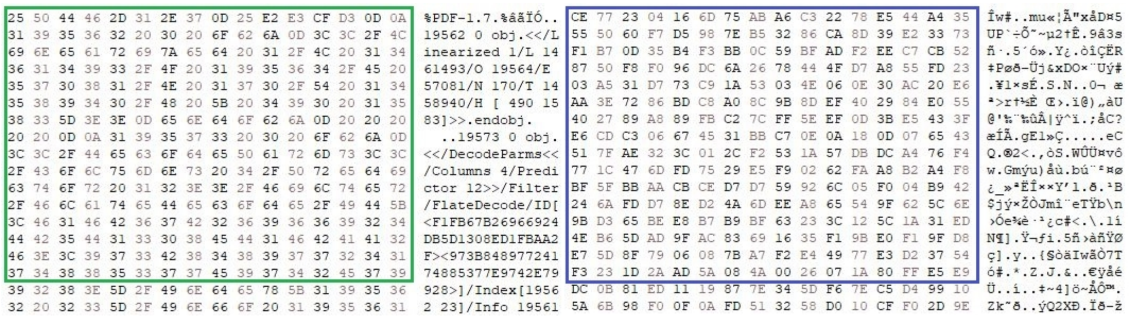


Figure 2.2: Differential area between two generic functions f(x) and g(x)



```
25 50 44 46 2D 31 2E 37 0D 25 E2 E3 CF D3 0D 0A  %PDF-1.7.%âãÏÓ..  CE 77 23 04 16 6D 75 AB A6 C3 22 78 E5 44 A4 35  Îw#..mu«¦Ã"xåD¤5
31 39 35 36 32 20 30 20 6F 62 6A 0D 3C 3C 2F 4C  19562 0 obj.<</L  55 50 60 F7 D5 98 7E B5 32 86 CA 8D 39 E2 33 73  UP`÷Õ˜~µ2†Ê.9â3s
69 6E 65 61 72 69 7A 65 64 20 31 2F 4C 20 31 34  inearized 1/L 14  F1 B7 0D 35 B4 F3 BB 0C 59 BF AD F2 EE C7 CB 52  ñ·.5´ó».Y¿.òîÇËR
36 31 34 39 33 2F 4F 20 31 39 35 36 34 2F 45 20  61493/O 19564/E   87 50 F8 F0 96 DC 6A 26 78 44 4F D7 A8 55 FD 23  ‡Pøð–Üj&xDO×¨Uý#
35 37 30 38 31 2F 4E 20 31 37 30 2F 54 20 31 34  57081/N 170/T 14  03 A5 31 D7 73 C9 1A 53 03 4E 06 0E 30 AC 20 E6  .¥1×sÉ.S.N..0¬ æ
35 38 39 34 30 2F 48 20 5B 20 34 39 30 20 31 35  58940/H [ 490 15  AA 3E 72 86 BD C8 A0 8C 9B 8D EF 40 29 84 E0 55  ª>r†½È Œ›.ï@)„àU
38 33 5D 3E 3E 0D 65 6E 64 6F 62 6A 0D 20 20 20  83]>>.endobj.     40 27 89 A8 89 FB C2 7C FF 5E EF 0D 3B E5 43 3F  @'‰¨‰ûÂ|ÿ^ï.;åC?
20 20 0D 0A 31 39 35 37 33 20 30 20 6F 62 6A 0D    ..19573 0 obj.  E6 CD C3 06 67 45 31 BB C7 0E 0A 18 0D 07 65 43  æÍÃ.gE1»Ç.....eC
3C 3C 2F 44 65 63 6F 64 65 50 61 72 6D 73 3C 3C  <</DecodeParms<<  51 7F AE 32 3C 01 2C F2 53 1A 57 DB DC A4 76 F4  Q.®2<.,òS.WÛÜ¤vô
2F 43 6F 6C 75 6D 6E 73 20 34 2F 50 72 65 64 69  /Columns 4/Predi  77 1C 47 6D FD 75 29 E5 F9 02 62 FA A8 B2 A4 F8  w.Gmýu)åù.bú¨²¤ø
63 74 6F 72 20 31 32 3E 3E 2F 46 69 6C 74 65 72  ctor 12>>/Filter  BF 5F BB AA CB CE D7 D7 59 92 6C 05 F0 04 B9 42  ¿_»ªËÎ××Y’l.ð.¹B
2F 46 6C 61 74 65 44 65 63 6F 64 65 2F 49 44 5B  /FlateDecode/ID[  24 6A FD D7 8E D2 4A 6D EE A8 65 54 9F 62 5C 6E  $jý×ŽÒJmî¨eTŸb\n
3C 46 31 46 42 36 37 42 32 36 39 36 36 39 32 34  <F1FB67B26966924  9B D3 65 BE E8 B7 B9 BF 63 23 3C 12 5C 1A 31 ED  ›Óe¾è·¹¿c#<.\.1í
44 42 35 44 31 33 30 38 45 44 31 46 42 41 41 32  DB5D1308ED1FBAA2  4E B6 5D AD 9F AC 83 69 16 35 F1 9B E0 F1 9F D8  N¶].Ÿ¬ƒi.5ñ›àñŸØ
46 3E 3C 39 37 33 42 38 34 38 39 37 37 32 34 31  F><973B848977241  E7 5D 8F 79 06 08 7B A7 F2 E4 49 77 E3 D2 37 54  ç].y..{§òäIwãÒ7T
37 34 38 38 35 33 37 37 45 39 37 34 32 45 37 39  74885377E9742E79  F3 23 1D 2A AD 5A 08 4A 00 26 07 1A 80 FF E5 E9  ó#.*.Z.J.&..€ÿåé
39 32 38 3E 5D 2F 49 6E 64 65 78 5B 31 39 35 36  928>]/Index[1956  DC 0B 81 ED 11 19 87 7E 34 5D F6 7E C5 D4 99 10  Ü..í..‡~4]ö~ÅÔ™.
32 20 32 33 5D 2F 49 6E 66 6F 20 31 39 35 36 31  2 23]/Info 19561  5A 6B 98 F0 0F 0A FD 51 32 58 D0 10 CF F0 2D 9E  Zk˜ð..ýQ2XÐ.Ïð-ž
```

Figure 2.3: Header bytes distribution between a PDF file (left, green) and a pseudorandom file (right, blue)

1. Take a segment (bytes) at the beginning of the file, called header
2. Compute the entropy (in bits) by taking subsequences of bytes starting from the 8th byte and moving up by multiple of 8, until the end of the sampled header
3. Compute the entropy (in bits) of an ideal file composed of random bytes
4. Subtract the entropy values (in bits) of the real file from the entropy values (in bits) of the ideal file
5. With the obtained values compute the area under the graph by using the Composite Trapezoidal Rule and a partition distance equal to 8 bytes, see Section 2.2, obtaining a Bit-Bytes area value
6. Compare the obtained Bit-Bytes area with a threshold
7. If the area is less or equal than the threshold, the header probably belongs to a ransomware encrypted file, otherwise is from a legitimate file

The first three steps create a graph composed of two entropy functions, the ideal file and the real file header Entropy function. Step 4 and 5 computed the DA between these two functions. Please look at Figure 2.5 for a better understanding of the resulting DA. The last two steps perform the classification, deciding if the selected file is either a ransomware encrypted or legitimate file.

The aforementioned process is repeated with multiple files and multiple thresholds in order to assess the best thresholds in terms of classification accuracy.

The DAA by Davies et al. [1] works really well because of two reasons. One is because, as explained in Section 2.1, the higher the randomness of a sequence of bytes is, the higher the Shannon Entropy value will be. Therefore, in a sequence where all bytes probabilities are equally likely, as in encrypted files, the Entropy value is high. Conversely, if some byte values

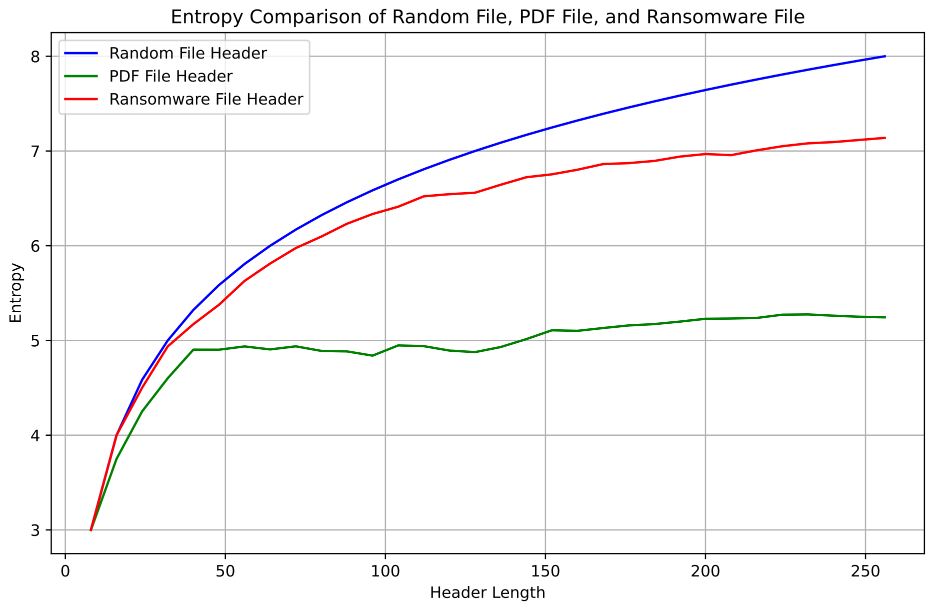


Figure 2.4: Graphical representation of the Entropy function for a random file (blue), PDF file (green) and ransomware file (red)

are repeated, the Entropy will be low. Examples of bytes distribution can be seen in Figure 2.3 and the respective Entropy functions graph is Figure 2.5. The second reason is that legitimate files tends to have a special sequence of metadata bytes at the beginning, while encrypted files are random right from the start. Therefore, the difference between Entropy values is more prominent at the start of the files rather than at later positions.

These two differences between ransom-encrypted and legitimate files are used by the DAA. If the Bit-Byte area of the file is close to the random file ideal area, the byte values are uniformly distributed and so the file is probably encrypted, otherwise it is legitimate (closeness between graphs is evaluated through the difference of the areas and a threshold parameter). An example of this can be seen in Figure 2.4 where it is clear that the ransom file graph is closer to the ideal file than the PDF file.

The reason to why the ransom Bit-Bytes entropy graph is close to the ideal file header graph is due to ransomware strains generally using AES Section A.1 which is a strong encryption algorithm. Thus, the encrypted files header have almost uniform data distributions, as explained in Subsection 1.2.1, and are closer to an ideal random file than legitimate and more structured files' header.

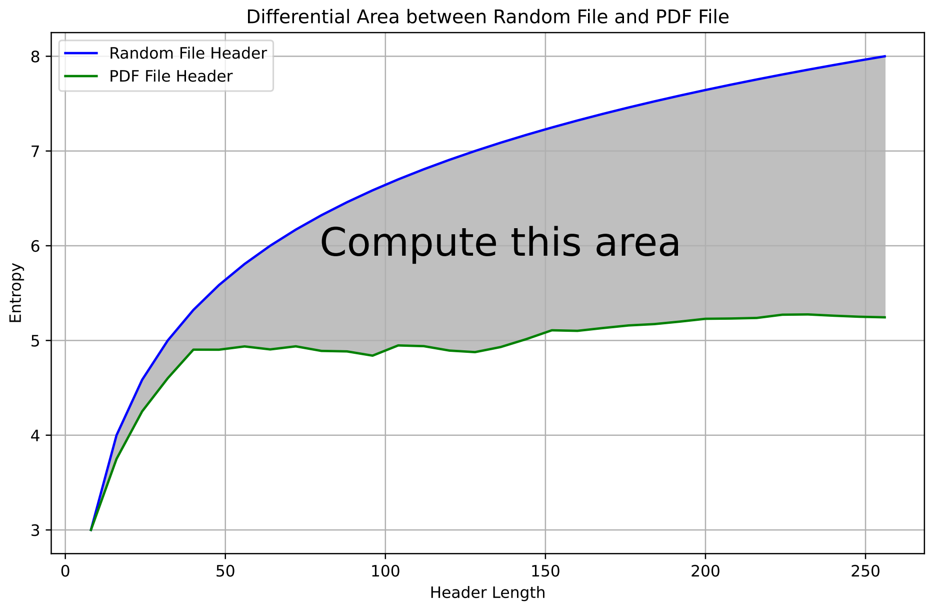


Figure 2.5: Differential Area between Ideal and Real PDF File

To briefly summarize, the ransom file has gone through an encryption algorithm, which is an algorithm that given an input called plaintext has the objective of computing a sequence of bytes called ciphertext difficult to inverse. This implies that, among other things, the output bytes' distribution probabilities has to be close to the uniform distribution, denying any possible correlation between plaintext and ciphertext. Therefore, the resulting ciphertext files have higher entropy values w.r.t. to the unencrypted files.

Davies et al. [1] implied that since ransomware will at one point cipher the data on the file system, if we are able to spot the encryption of a file the attack could be prevented.

Therefore, in the case of ransomware detection the encryption is considered an adversary operation and the proposed DAA seems to be effective if used to identify such operation in order to distinguish ransomware from legitimate files and ultimately detect an ongoing attack.

2.5 Machine Learning

In this section, Machine Learning as a concept is shortly introduced and the theory behind the algorithms used in this thesis is briefly explained. All the concept reported in this section and much more can be found at [4, 44].

The term machine learning is typically employed to describe the modifications to systems that perform tasks associated with artificial intelligence (AI). These tasks involve recognition, diagnosis, planning, robot control, prediction, and other related functions. The "changes" may be either enhancements to existing systems or the creation of entirely new systems.

There are three main categories when approaching machine learning: output domain, statistical nature of the technique and type of training.

Output domain is divided in discrete (classification) or continuous (regression). Discrete is when the data fall into one or N classes, continuous in a real number line.

The statistical nature of a given phenomenon may be either probabilistic or non-probabilistic. A technique is probabilistic when it incorporates some data using a summary in the form of a statistical distribution, thus not all data are required for the training process. In contrast, a technique is defined as non-probabilistic when it employs the data to perform a specific action, such as determining the proximity of new data to existing data within a given dataset.

The process of machine learning can be divided into three distinct phases: training, validation and evaluation. The training phase may be either supervised or unsupervised. In supervised learning, the data are known and labelled, and the training process entails the model learning the relationship between input and output data. In unsupervised learning, the data are unknown, and the objective is to identify a summary or description that will facilitate an understanding of the distribution. The validation phase is employed to optimize the model hyperparameters, while the evaluation phase enables the assessment of the model's performance when presented with previously unseen data.

The techniques used in this thesis are Support Vector Machines, Random Forests which are specifically non-probabilistic, and Neural Network that can also be modelled as non-probabilistic.

The thesis addresses the problem of ransomware detection, which can be modelled as follows: given some file features belonging either to a ransomware-encrypted file or to a legitimate file, the objective is to understand the relationship between the features and the label. Therefore, all three models are included within the discrete (binary), non-probabilistic supervised category.

2.5.1 SVM

SVMs are useful for a wide range of classification and regression problems, and form part of a family of techniques known as margin methods. The fundamental objective of margin methods, and specifically SVMs, is to create as much separation between data points and decision

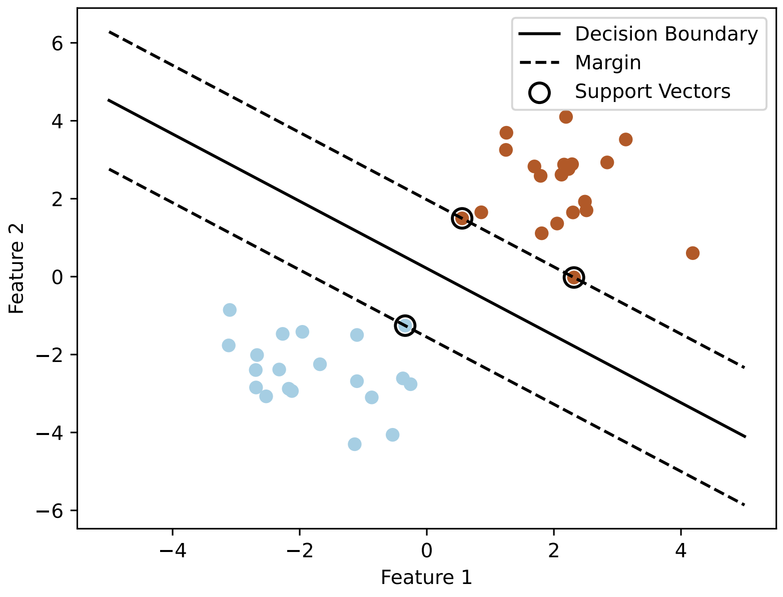


Figure 2.6: SVM with hard margins

boundaries as possible.

The fundamental concept underlying SVMs is that we should construct a linear hyperplane in our feature space that maximally separates our classes. This means that the different classes should be as far from that hyperplane as possible. The distance of the data from the hyperplane is referred to as the margin, and it is shown in Figure 2.6. Larger margins often lead to more generalizable models, since they provide more room to correctly classify unseen data (new data can be seen as a perturbation on current data).

Margins are broadly divided into two categories: hard margin and soft margin. The term ”hard margin” refers to the condition in which no data is classified incorrectly. If it is not possible to find a hyperplane that perfectly separates the data based on class, the hard margin classifier will return no solution. This approach is effective when classes are linearly separable, which is the basis of the hard margin formulation for SVMs. However, what if they are not? It is necessary to relax the constraints. The soft margin approach permits some of the data points to be closer to or even on the incorrect side of the hyperplane, as seen in fig Figure 2.7. To enable the soft margin formulation, slack variables are introduced, which simply relax the constraint from what is imposed in the hard margin formulation.

Given the margins definitions, it’s possible to understand what a support vector is. Simply

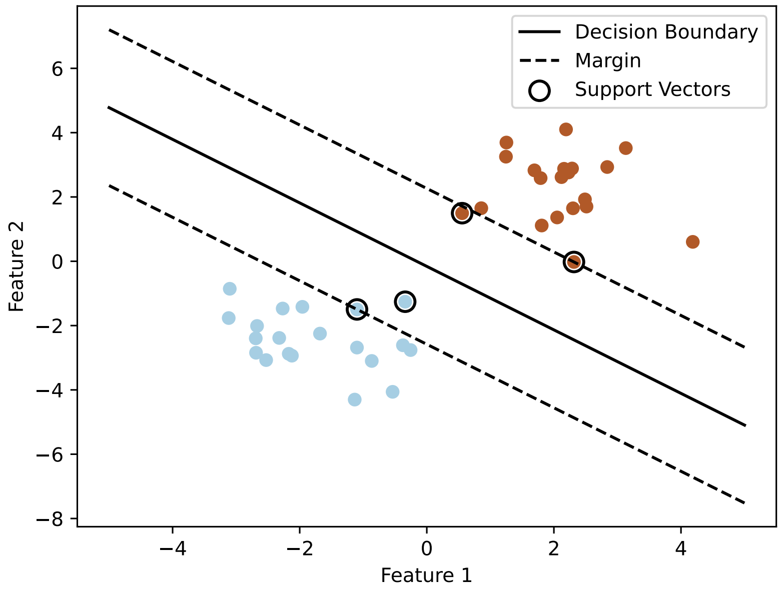


Figure 2.7: SVM with soft margins

put, support vectors are the points that are the closest to the hyperplane.

Accordingly, a support vector in a hard-margin SVM formulation must be a data point that is on the margin boundary of the optimal solution, whereas in a soft margin formulation the support vectors are permitted to eventually be within the margins.

SVMs are typically used in a non-probabilistic settings with discrete outputs, and labeled training data are needed to identify the relevant hyperplane in an SVM model.

2.5.2 Random Forests

A random forest is defined as a collection of classification and regression decision trees (CaRT). They fall within the broad category of ensemble learning, which is a technique that combines weak classifiers to enhance performance. This approach emerged from the insight that, rather than utilizing a single decision tree for learning, aggregating multiple decision trees, and subsequently enabling each classifier to cast a vote for a class, would enhance the overall accuracy. To encourage diversity among the trees and prevent overfitting, some randomness is involved during the computations. The most prevalent methodology for the construction of a random forest is the combination of Bagging with random Classification and Regression Trees (rCaRTs).

Bagging, also known as Bootstrap aggregation, is a technique of ensemble learning that is employed to circumvent overfitting and enhance stability and accuracy. The bagging technique comprises two distinct phases. The bootstrap of a sample set and the subsequent aggregation constitute the fundamental steps of the Bagging technique. In essence, bootstrap can be defined as the generation of smaller, randomly selected data subsets, derived from a given initial set. In more specific terms, given the initial learning set L, K smaller or equal learning sets are generated by uniform sampling with replacement from L (it is possible that some samples may be repeated among the smaller sets). Generally, the subsets generated by the random forest algorithm have the same dimension as the starting set L. Subsequently, the learning process is performed, whereby a random CaRT is learned for each subset generated.

First, all the labeled samples are initially assigned to the root node. Then given a random subset of features, find the feature-threshold parameters that better split the assigned sampled into two subsets, left and right, and also that maximize the label purity within these subsets. Purity could be computed in different ways, one of which is the Shannon Entropy [34], briefly described in Section 2.1, of the subset.

Once the feature-threshold parameters have been identified, they should be assigned to the node, with the root node serving as the starting point. The aforementioned procedure should be repeated for both the left and right splits, with each iteration involving the assignment of a feature-threshold parameter to the left and right nodes. This process should continue until the splits are deemed too small to be divided. In such a case, two child leaf nodes should be attached to the current node, with the left leaf being tagged with the most prevalent label in the left split, and the same process repeated with the right leaf.

A random drawing of features is repeated at each node to avoid overfitting and reduce computational load.

A useful image describing the process of decision tree creation, taken by [4], is Figure 2.8. For a formal explanation, please refer to [45].

Once the Random CaRTs are created, if new samples are provided, the Aggregation step is performed. Given a new sample, a decision is formulated based on the majority vote among the K predictions over the total votes.

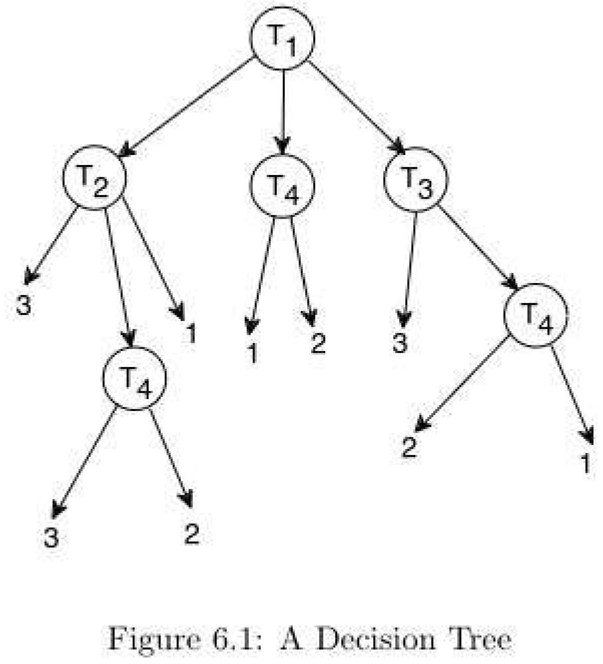


Figure 2.8: A general Decision Tree model taken by [4]

2.5.3 Neural Networks

Neural Networks are known as universal function approximators. This means that with a large enough network, it is possible to approximate any function.

Neural networks can operate over discrete or continuous outputs and are primarily used to solve regression or classification problems, which involve training on data sets with example inputs and outputs, making this a supervised technique. While there exist probabilistic extensions for neural networks, they primarily operate in the non-probabilistic setting.

A multitude of neural network types exist, and in this thesis, the model utilized is a fully connected feed-forward neural network. A feed-forward neural network is a series of interconnected layers that transform an input data point, into an output data point. Each layer is composed of nodes, and the term ”fully connected” indicates that every node of a layer is connected to all the nodes of the successive layer. Since the ransomware classification problem in this thesis is tackled as a binary classification, a single output node is employed to represent the predicted probability of the positive class (K outputs would have been used in case of multi-classification). An example of a network used for binary classification, similar but smaller to

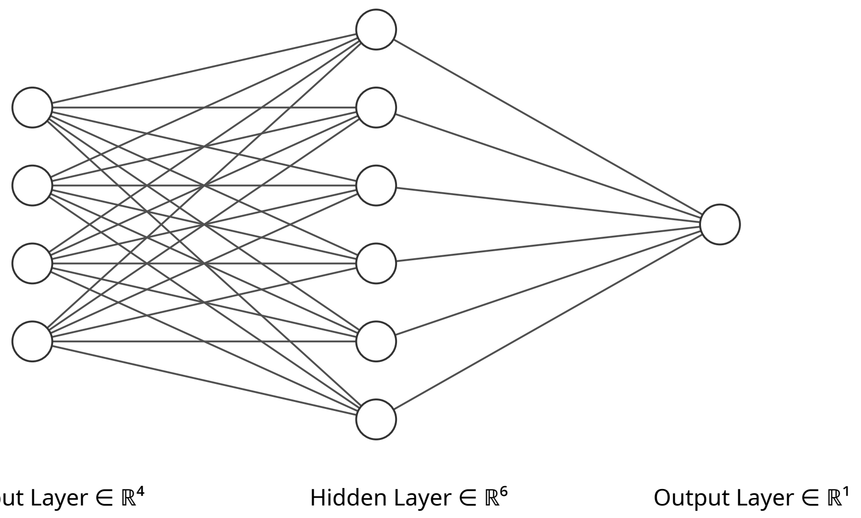

Input Layer $\in \mathbb{R}^4$
Hidden Layer $\in \mathbb{R}^6$
Output Layer $\in \mathbb{R}^1$

Figure 2.9: A simple Neural Network

the one implemented in this thesis, can be seen in Figure 2.9. In the context of a regression problem, the number of nodes may vary depending on the dimensions of the output.

Except for the input layer, the value of a node in a layer for a fully-connected network can be computed by multiplying all the nodes' values from the previous layer by their corresponding weight, adding them together, and using the resulting sum as input for a function called the activation function, which will then produce an output.

A binary classification task is usually modeled through a single sigmoid output activation unit and with the negated log-likelihood (or cross-entropy) as the loss function. The function is described in Figure 2.5.3.

Training consist in finding the weight parameters that minimize the objective function. To do so, first we need to compute the errors of the current weights, and then we need to update the weights accordingly. The technique used to compute the errors is called Backpropagation.

Backpropagation refers specifically to the portion of neural network training during which the derivative of the objective function with respect to the weight parameters is computed. This is done by "propagating errors backwards" through the network, which in practice means using the chain rule property applied to the computed derivatives. Once the errors are computed, the weights are updated accordingly to the Gradient descent algorithm or some variant of it.

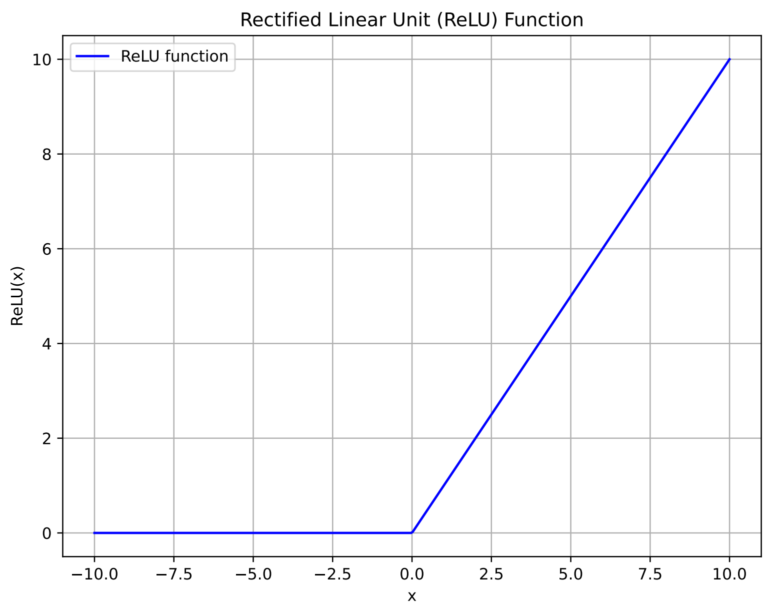


Figure 2.10: The Rectified Linear Unit (ReLU) function

Activation Functions

The outputs of the hidden units is associated with an activation function. Typical activation functions are the *sigmoid* function, the *tanh* function and the Rectified Linear Unit, *ReLU* function.

The *ReLU* activation function is defined as follows:

$$ReLU(x) = max(0, x). \tag{2.4}$$

A graphical representation can be seen in the Figure 2.10

The *sigmoid* activation function is defined as:

$$\sigma(x) = \frac{1}{1 + e^{-x}}. \tag{2.5}$$

Which graphically results in Figure 2.11

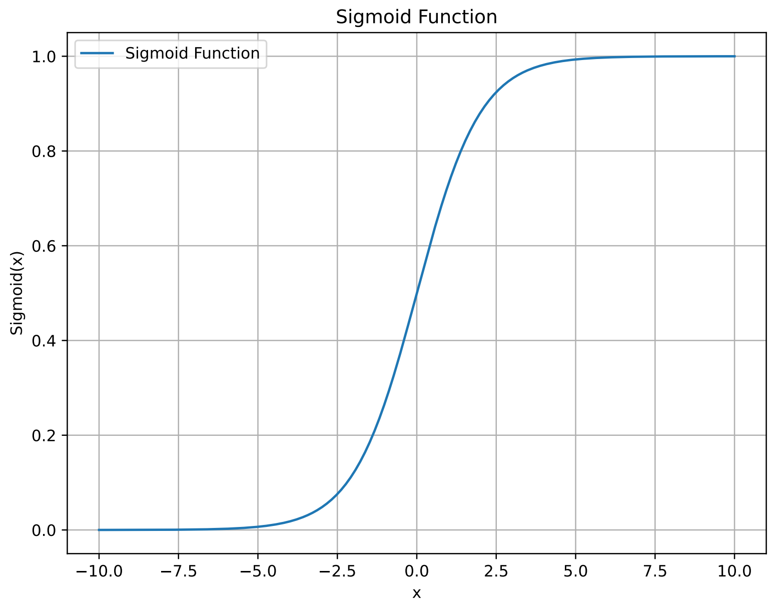


Figure 2.11: The Sigmoid function

Loss functions

Loss function or objective function refers to the same concept: the function to optimize to train the model.

Among all the possible functions, since the problem in this thesis is formulated as a binary classification problem where data are either in class 0 or class 1, the function to be optimized is the Binary Cross-Entropy(BCE), which is defined as:

$$BCE(y) = (y \log(p) + (1 - y) \log(1 - p)). \tag{2.6}$$

Where *log* is the natural log, y is the actual binary label (0 or 1) of the observed data and p is the predicted probability of the observed data of being in class 1.

2.5.4 Metrics

Throughout this thesis, four standard metrics were used to evaluate the performance of attacks, countermeasures and models. Namely, accuracy, precision, recall, and F1 score. In this section, they are briefly summarized and adapted to the ransomware classification problem as described

Table 2.1: Parameters used in the metrics.

Classification	Description
True Positive (TP)	Encrypted files correctly classified
False Positive (FP)	Unencrypted files erroneously classified as encrypted
True Negative (TN)	Unencrypted files correctly classified
False Negative (FN)	Encrypted files erroneously classified as normal unencrypted

by Venturni et al. [3]. For the sake of completeness, Table 2.1 clarifies the terms used in the metrics.

Accuracy is the total number of files that have been classified correctly.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}. \tag{2.7}$$

Precision represents the ratio of the files correctly identified as encrypted by ransomware among the total number of files classified as encrypted by a ransomware.

$$Precision = \frac{TP}{TP + FP}. \tag{2.8}$$

Recall also known as sensitivity, represents the ratio of the files correctly identified as encrypted by ransomware among the total number of files really encrypted by a ransomware. Both Precision and Recall are therefore based on relevance.

$$Recall = \frac{TP}{TP + FN}. \tag{2.9}$$

F1 Score is the harmonic mean of the Precision and Recall.

$$F1 = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall}. \tag{2.10}$$

3 Methodology

Ransomware have been a steadily growing threat since 2005, causing billions of dollars of damage during the years, as previously described in Section 1.1.

The ongoing arm race between security researchers and ransomware has evolved a lot over the time, but given the adaptive nature of such ransomware attacks, the proposed mitigations techniques fall quickly behind. Therefore, there is always the need for new and improved defense mechanisms, adapted to the ransomware evolution.

The ransomware nature imply that at one point, some filesystem data will be encrypted. A general used feature by defense mechanisms to detect ongoing encryption is the Entropy. However, computing the Entropy of the entire file is time-consuming, a crucial resource against ransomware quick execution.

The aim of this thesis is to achieve a fast, lightweight ransomware classification method, able to adapt to different ransomware strains. To do so, inspired by the work of Davies et al. [1], a defense mechanism based on files' header Entropy and the Differential Area (DA) is proposed. To understand what the DA is and how it is computed, please refer to Section 2.3 and Section 2.4. Once the aforementioned features are extracted, three machine learning algorithms are trained and tested.

Using the Entropy as a distinctive feature is prone to Entropy neutralization techniques. To compensate this vulnerability, inspired by the work of Venturini et al. [3], three random file segments selection strategies were applied. Then, the selected features were tested against three Entropy neutralization strategies tailored to target the files' header.

This allows to strengthen the findings because, as Davies et al. specified in [33], to develop a reliable defense mechanism based on the Entropy, it needs to be tested against Entropy neutralization techniques.

In this chapter, Section 3.1 is used to formalize this thesis hypotheses. In Section 3.2 each part of the experiment design is discussed. Section 3.3 describe the used datasets. In Section 3.4 it is assessed how each component was implemented.

3.1 Hypotheses

Generally, ransomware detection methods require a lot of resources to obtain significant results. The main hypothesis that this thesis explores is the possibility of building a reliable, lightweight ransomware defense mechanism that can quickly and accurately identify a working ransomware.

The use of machine learning seemed a good starting point, given the algorithm's ability to learn data correlation and generalize to future unseen data, including unknown ransomware strains.

Another point was that the features selected needed to be limited, since collecting a lot of data could slow the detection.

By looking at past works, the DAA by Davies et al. [1] was able to obtain very good results using fewer features than other ransomware classification methods. They sampled a fixed length segment at the beginning of the file, extrapolated the Entropy, and compared it against the Entropy of an ideal random file of the same length. The comparison was done by first subtracting them, thus computing the Differential Area, and then comparing the results against a threshold. For more information, please refer to Chapter 2.

Thus, the Entropy and the Differential Area (DA) of the files' header, which can be computed quickly, seemed to carry relevant information for ransomware classification and therefore can theoretically be used as features to learn from for machine learning algorithms.

As a result of using these features, the sub-hypothesis explored was if using the Entropy or the DA paired with machine learning could be effective at distinguishing ransomware encrypted files from legitimate files or if the extracted information were not enough to perform such task.

3.2 Design

Some steps were required to prove the hypothesis expressed in Section 3.1.

The first step was to assess if the Entropy, the DA and the DAA defined by Davies et al. are computed correctly. Checking if the Entropy and the DA evaluated rightfully is useful because they are later used as features for the machine learning algorithms.

Regarding the DAA, checking if it is computed as intended allows to later prove that the Entropy and the DA still carries relevant information for the known ransomware strains classification.

The second step was divided into two sub-steps. First sub-step, using step 1, prove that the DAA still works with known ransomware. By doing so, it is demonstrated that Entropy and DA can still be useful with current ransomware. The second sub-step is proving that the DAA is vulnerable to files' header Entropy neutralization techniques, pointing out the need for stronger defense mechanisms.

The third step was to implement the mitigations proposed by Venturini et al. [3]. Proving that their segment selection strategies carries meaningful information and can mitigate the header Entropy neutralization techniques. This is useful because the same features extraction processes are later used by machine learning algorithms.

The fourth step was to provide to the machine learning algorithms the Entropy and the DA, extrapolated either as defined by the DAA from Davies et al. or as defined by the selection strategies defined by Venturini et al.

The fifth and last step was to test such features against three header entropy normalization techniques to assess if the robustness of the proposed features-model combo.

The fourth step is the concrete implementation of a lightweight ransomware defense mechanism, but the prior steps are needed to verify if the features used by the algorithms carry relevant information for ransomware classification and also the need for stronger, more generalizable approaches. The fifth step is essential in proving if the extracted features and models, which depend on the Entropy, could be strong against the Entropy neutralization techniques.

3.3 Datasets

To complete the steps and verify the hypotheses written in Section 3.1 three different datasets were used. One flaw of the major researches in the literature is the use of limited datasets, if not on the raw number of files itself, surely on the different file types. Fortunately, Davies et al. created a dataset, called NapierOne [46], composed not only of ransomware strains up to 2022, but also with a lot of different file classes. In particular, the dataset focus on legitimate file extensions with high entropy values. This allows to enhance the results obtained, because if the computed entropy between ransomware encrypted files and legitimate files were too different, the classification problem would have been trivial. The dataset has three possible sizes: tiny, small and total. The tiny size includes 100 files for each category, the small 1000 and the total comprehend all the files. To allows for repeated and quicker tests, the dataset size chosen was the tiny. Future works, instead of testing multiple parameters, could focus on this thesis findings by selecting the best performing ones and extend the dataset to better train the algorithms and improve the findings.

The three different datasets in this thesis are addressed with the names: OldNapierOne, NapierOne and Attack Dataset.

OldNapierOne, since it is the same used from Venturini et al. [3], was used to confirm the correctness of the computed Entropy and DA and to check if the rewritten DAA algorithm was correct by comparing the obtained results with the results of Davies et al. [1] and Venturini et al. [3].

NapierOne is an up-to-date and extended version of the previous dataset. It has more categories of ransomware and legitimate files, and thus more files in general.

Since in machine learning the datasets are divided into three splits, generally named train, validation and test, this dataset was split according to this standard approach. This was done because even if it is used by the rewritten DAA and the 2F, 3F, 4F mitigations, which do not select and test the parameters due to their analytic nature, it was also used by the machine learning algorithms.

The train split was used to evaluate the DAA and the 2F,3F,4F methods and to compute the input features for training the machine learning algorithms, the validation split was used to tune the machine learning hyperparameters and the test split was used to evaluate the machine learning algorithm's ability to "generalize" and predict unseen data.

The DAA and the 2F,3F and 4F were evaluated on the train split and not on the validation or test split because neither Davies et al. [1] nor Venturini et al. [3] provided a selection method for the best parameters but only proposed the methods as possible ransomware classification strategies. Thus, the selection of the best parameters and the validation/test on new data to evaluate the generalization ability of the proposed methods evade the scope of this thesis, which only aims to use the DAA and the 2F, 3F and 4F to demonstrate the correct computation of the features and as a comparison reference between different ransomware classification approaches.

Both the OldNapierOne and the NapierOne to create encrypted files used common files. In particular, DOC, DOCX, JPG, PDF, PPTX, XLS, and XLSX categories were selected and applied to different ransomware strains.

Since each ransomware when encrypting the file disk use the original AES 128/256-bit algorithm or a more modern version of it, please refer to Section A.1 to understand which operations are performed during a ransomware file encryption phase.

As stated before, the original NapierOne is divided in three sizes: tiny, small and total. The tiny version consists of 100 files for each file types of the dataset. More information on the original dataset, size, file structure, procedures and more can be found at [46].

The dataset can be downloaded, prior to a notification to the researchers, at `http://napierone.com/Website/index.html`.

Lastly, the Attack Dataset is the dataset used to simulate the attacks described in detail in later Subsection 3.4.4. This is done by simply taking the ransomware files in the NapierOne dataset, modifying the headers in the three ways as described in Subsection 3.4.4 and adding them to the final dataset, which will basically be the NapierOne dataset plus the modified ransomware files.

3.3.1 OldNapierOne

This dataset is better described in [3] and is an older version of the tiny version of the used NapierOne dataset. It contains around 2900 files, 100 files per category, which are:

- Legitimate files: 7ZIP (BZIP2, ENCRYPTED, HIGH-COMPRESSED), CSS, DLL, DOC, DOCX, GIF, JPG, MKV, MP3, MP4, PDF, PNG, PPTX, RAR, TAR, XLS, XLSX, XML
- Ransomware encypted files: BADRABBIT, DHARMA, MAZE, NETWALKER, NOTPETYA, PHOBOS, RYUK, SODINOKIBI, WANNACRY.

3.3.2 NapierOne

This dataset is the up-to-date tiny version of the NapierOne dataset. It contains 100 files for 122 categories, for a total of 12200 files.

The categories are:

- Legitimate files: 7ZIP (BZIP2, ENCRYPTED, HIGH-COMPRESSION, LZMA, LZMA2, PPMD), APK, BIN, BMP-FROM-WEB, CSS, CSV, DLL, DOC (NOMAGIC, PASSWORD), DOCX (NOMAGIC, PASSWORD), DWG-FROM-WEB, ELF, EPUB, EPS-FROM-WEB, EXE, GIF-FROM-WEB, GZIP, HTML, ICS, JAVASCRIPT, JPG(from-web, q001, q025, q050, q075, q100), JSON, MKV, MP3, MP4-FROM-WEB, ODS, OXPS, PDF (NOMAGIC, PASSWORD), PNG (c0, c3, c5, c7, c9, FROM-WEB), PPT (NOMAGIC, PASSWORD), PPTX (NOMAGIC, PASSWORD), PS1, RANDOM (PSEUDO, PURE), RAR, SVG (FROM-WEB), TAR, TIF-RESIZED, TXT, WEBP (lossless-c0, lossless-c2, lossless-c4, lossless-c6, q50-c0, q50-c2, q50-c4, q50-c6), XLS (NOMAGIC, PASSWORD), XLSX (NOMAGIC, PASSWORD), XML, ZLIB, ZIP (BZIP2, DEFLATE, ENCRYPTED, HIGH-COMPRESSION, LZMA, PPMD).

- Ransomware encypted files: JIGSAW, DARKSIDE, WASTEDLOCKER, BLACKMATTER, BLACKCAT, NETWALKER, BLACKBASTA, RANSOMEXX, LOCKBIT, CUBA, MAZE, BADRABBIT, DHARMA, LORENZ, SODINOKIBI, WANNACRY, PHOBOS, MEDUZALOCKER, CERBER, CHIMERA, AVOSLOCKER, TESLACRYPT, CONTI, RYUK, GANDCRAB, SUNCRYPT, HELLOKITTY, CRYPTOLOCKER, NOTPETYA, CLOP.

In total the common files categories are 92 with 100 files each, thus the total legitimate files are 9200.

The ransomware encrypted files counts 30 ransomware strains, each having 100 files, for a total of 3000 ransomware files.

As mentioned before, the dataset was split in three parts: train, validation and test.

In machine learning, this is done to follow the concept of generalization, which is to learn from known data in order to predict new data. Generally, given that data are difficult to obtain, the dataset is divided into two splits, one where the machine learning algorithm can learn and the other split to evaluate the model ability to generalize on unseen data.

Hence, the training split from which the model can learn, and the test split to address the model performance. The validation split is a subpart of the training split and is used during the training phase to check if the model is over-fitting (good on predicting seen data, bad at predicting unseen data) and adjust the hyperparameters (parameters which influence the model behavior) and tune the hyperparameters. The split rule followed the classic 60% of total as training data, 20% as validation data and 20% as test data.

Given that the 4F countermeasure, described in Subsection 3.4.5, and the H+3RS file segments selection strategy, described in Subsection 3.4.6, select 4 segments in total, to keep the data coherent between different algorithms and runs, all the files shorter than 4 times the maximum segment length chosen were excluded. Since the greater segment length analyzed is 256 bytes, file shorter than 1024 bytes were filtered and not taken into consideration.

The total number of filtered files is 345, no ransomware encrypted files were excluded. Each category and the relative number of eliminated files can be found at Table 3.1.

The resulting number of files for each split can be found at Table 3.2.

Table 3.1: Filtered Files per Category

Category	**Filtered Files**
TXT	20
ICS	33
JSON	2
PS1	17
CSS	27
CSV	27
RANDOM-PURE	31
JPG-from-web	1
JAVASCRIPT	52
RANDOM-PSEUDO	46
PNG-FROM-WEB	3
GIF-FROM-WEB	8
XML	70
SVG-FROM-WEB	8
Total	345

Table 3.2: Number of files for each NapierOne split

Split	Ransomware	Legitimate	Total
Train	1849	5471	7320
Filtered Train	**1849**	**5258**	**7107**
Validation	588	1852	2440
Filtered Validation	**588**	**1786**	**2374**
Test	563	1877	2440
Filtered Test	**563**	**1811**	**2374**

Table 3.3: Number of files for each Attacks Dataset split

Split	Ransomware	Tempered Ransomware	Legitimate	Total
Train	1849	5547	5471	12867
Filtered Train	**1849**	**5547**	**5258**	**12654**
Validation	588	1764	1852	4204
Filtered Validation	**588**	**1764**	**1786**	**4138**
Test	563	1689	1877	4129
Filtered Test	**563**	**1689**	**1811**	**4063**

3.3.3 Attacks Dataset

To simulate the results that ransomware strains would obtain by applying the three attacks proposed by Venturini et al. [3], NapierOne dataset Subsection 3.3.2 was enhanced by adding modified encrypted files. The header of ransomware files encrypted from different strains was modified at run times in order to obtain three different versions:

- Files prepending a low-entropy 256 bytes block (i.e., a repetition of the character 'a');
- Files repeating the first 8 bytes for 32 times;
- Files substituting the first 256 bytes of the file with a random succession of various low-entropy sequences. The uppercase alphabet, the lowercase alphabet, some common words, numbers between 0 and 22, and a repetition of the number zero were used.

Given that the original ransomware encrypted files were 3000 and each file produced three modified files, 9000 tempered ransomware encrypted files are added. Therefore, the total number of files become 21200.

As for the NapierOne dataset, the files smaller than 1024 bytes were filtered. In total, the eliminated files are 345, and in fact are the same legitimate files excluded in NapierOne dataset.

The resulting number of files for each split can be found at Table 3.3.

3.4 Implementation

In this section, it is assessed how each component was implemented.

As described in Section 3.1 the general hypothesis that this thesis aims to explore is if it is possible to build a reliable lightweight ransomware defense mechanism and therefore given the chosen features if the Entropy or the DA can be used as features for different machine learning models for ransomware detection. This section, will report how the steps described in Section 3.2 were implemented, what limitations were encountered and what approach has been used to overcome them.

All the following described algorithms were implemented with the use of the TensorFlow python library (version 2.14). Specifically, it was used the tensorflow-2.14-gpu docker on the datacenter of the Örebro university ORU-GDX. The NapierOne dataset was manually downloaded from `http://napierone.com/Website/index.html` and configured using the TensorFlow Dataset API library. The Attack dataset was implemented by starting from the NapierOne dataset and by modifying the ransomware files at run time with the use of the TensorFlow APIs. The Entropy, the DAA and the Mitigations were implemented in python using the TensorFlow library. The machine learning algorithms SVC and Random Forest were implemented by importing the models from the scikit-learn python library. The Neural Network was realized thanks to the combination of the Keras python library and TensorFlow.

3.4.1 Entropy

The Entropy was computed only on some file's segments, similarly as Davies et al. [1]. In particular, until the segment of length *segment_length* is reached, the Entropy is computed starting from the beginning of the segment on smaller chunks of the segment in 8 bytes increments. An exampled can help understand the described process. If the segment is 32 bytes long, then the entropy is computed on the first 8 bytes, then on the first 16 bytes, on the first 24 and so on up until 32. A visual explanation of the aforementioned exampled is provided in Figure 3.1.

A basic outline of the experimental steps performed to obtain the Entropy values is reported in Algorithm 3.1

3.4.2 DA

Once the Entropy values have been computed, the Differential Area (DA) is computed by subtracting the *entropy_values* vector from the *ideal_entropy_values*. *ideal_entropy_values* is computed by applying the Algorithm 3.1 to an ideal file composed of random bytes. The pseudocode of the DA algorithm is Algorithm 3.2.

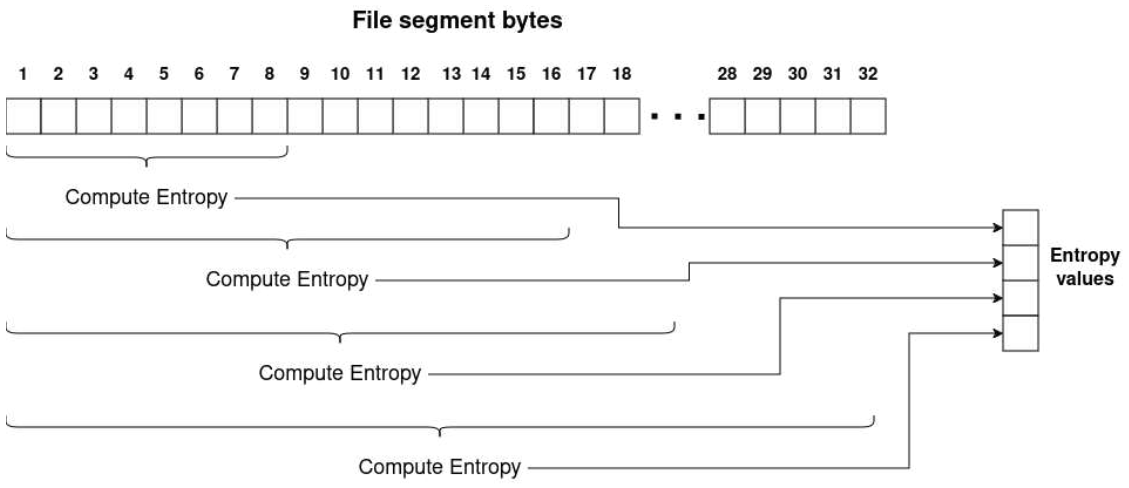


Figure 3.1: Entropy values computation of a 32 bytes long file's segment

Algorithm 3.1 Pseudocode of the Entropy computation method

```
Require: file_segment: Tensor containing the file's segment bytes for which the Entropy is to
    be calculated.
Ensure: entropy_values: Tensor containing the entropy values for different partitions of the
    file.
    Begin
    $entropy\_values \leftarrow []$
    for ( $i \leftarrow 8$ ; $i < length(file\_segment)$ ; $i \leftarrow i + 8$ )
        $entropy \leftarrow compute\_entropy(file[: i])$
        $entropy\_values.append(entropy)$
    end for
    return $entropy\_values$
    End
```

Algorithm 3.2 Pseudocode of the DA

```
Require: entropy_values: Tensor containing the file Entropy values for which the DA is to be calculated.
Require: ideal_file: Tensor containing the ideal file bytes for which the Entropy is to be calculated.
Ensure: da_values: Tensor containing the DA values for different partitions of the file.
  Begin
  $ideal\_entropy\_values \leftarrow []$
  for ( $i \leftarrow 8$ ; $i < 256$ ; $i \leftarrow i + 8$ )
    $ideal\_entropy \leftarrow compute\_entropy(ideal\_file[: i])$
    $ideal\_entropy\_values.append(ideal\_entropy)$
  end for
  $da\_values \leftarrow []$
  for ( $i \leftarrow 0$ ; $i < len(entropy\_values)$ ; $i \leftarrow i + 1$ )
    $da \leftarrow composite\_trapezoidal\_rule((ideal\_entropy\_values[: i] - entropy\_values[: i]))$
    $da\_values.append(da)$
  end for
  return $da\_values$
  End
```

3.4.3 DAA

The pseudocode described in Algorithm 3.3 is the general approach to compute the DAA. However, if it was implemented exactly as described, the complexity would quadratically depend on the number of thresholds tested and the number of DA elements. Fortunately, it is possible to reduce the number of computations by expanding the analyzed DA vector into a matrix, with the number of rows equal to the number of thresholds and each matrix row composed of the initial DA vector. Then each row is compared with a different threshold value in parallel.

For the sake of the reader and simplicity, since the outputs are the same, the following pseudocodes reported are the base version, which are inefficient but easier to understand.

When the DA is computed, the DAA is performed by comparing the values to different thresholds. Here is the pseudocode of the algorithm Algorithm 3.3.

Algorithm 3.3 Pseudocode of the DAA

```
Require: da_values: Tensor containing the DA values between the real and the ideal file for
   which the DAA is to be calculated.
Ensure: daa_values: Tensor containing True if the DA is less or equal than a threshold, false
   if the DA is greater.
   Begin
   $daa\_values \leftarrow []$
   for ( $threshold \leftarrow 0$ ; $threshold < 256$ ; $threshold \leftarrow threshold + 1$ )
      for ( $i \leftarrow 0$ ; $i < 32$ ; $i \leftarrow i + 1$ )
         if $da\_values[i] \leq threshold$
            $daa\_values.append(True)$
         else
            $daa\_values.append(False)$
         end if
      end for
   end for
   return $daa\_values$
   End
```

3.4.4 Attacks

Subsequently, the attacks proposed by Venturini et al. [3] were implemented via the Attacks dataset, explained in Subsection 3.3.3).

The goal of the attacks is to lower the files' header entropy and thus enlarge the DA between ransomware encrypted files and ideal file. Neutralizing the Entropy is performed to evade Entropy-based ransomware detection methods. This can be achieved in several ways, and in this thesis the three alternatives proposed by Venturini et al. [3] were implemented:

- Prepending to the file header a block of 256 bytes with low entropy, such as a single letter repeated to fill the block;
- Filling up the 256 bytes header with 32 repetitions of the first 8 bytes;
- Inserting low-entropy entries in random or fixed positions of the header, for example common words or repetitions of the alphabet.

The general approach in the literature is to use encoding to implement neutralization techniques, as described in [6, 33, 2, 7, 8].

3.4.5 Mitigations

The attacks mentioned in Subsection 3.4.4 are made possible by the fact that DAA focuses on analyzing only a predefined and limited part of a file.

Even though effective, this makes the approach predictable and easy to circumvent. Therefore, it is necessary to strengthen DAA in such a way that it can resist to malicious manipulation of file headers.

Thus, the Mitigations proposed by Venturini et al. [3] were implemented and to better understand them, it could be useful to briefly describe the parameters that are used by such mitigations.

Threshold

In order to perform the Differential Area Analysis, it is necessary to establish the range within the computed entropy value that is considered too close to the curve of a random file Entropy graph line. This signal indicates that the file in question is encrypted.

This is the same parameter used in the DAA by Davies et al. [1], more information can be found in Section 2.4

Distance

In regular unencrypted files, fragments positioned far away from the header have a relatively higher chance to exhibit high entropy with respect to the header, since structural information is stored between file portions close to the beginning.

Given that the mitigations analyze the header fragment and one (or more) additional fragments, it is essential to consider the higher entropy (and consequently, the smaller Differential Area) of the random-position fragment. This is crucial to prevent any skewed behavior of the algorithms in favor of the random fragments. To this end, a parameter designated ”distance” is introduced, which reduces the Differential Area associated with the header fragment when compared to the Differential Area of the random-position fragment(s). This has the effect of increasing the priority of header entropy information over the randomly selected fragment, which is of particular importance when analyzing unencrypted files. By adjusting the distance parameter, the number of false positives is reduced, thereby reducing the likelihood of incorrectly classifying regular files as ransomware-encrypted files.

Subfragment length

As for the DAA, which analyses 256 bytes in incremental subfragments that grow from 8 up to 256 bytes at 8 bytes at a time, the mitigations' accuracy is monitored at each incremental 8 bytes step. It is useful to correlate the results with the subfragments length and compare same length results for the three different mitigations.

2-Fragments (2F) Mitigation

As previously described, DAA analyses one fragment, the header, of 256 bytes. With 2F, two fragments are analyzed, one being the header, and one picked at a random byte position, within the range $[length(header), ..., length(file) - length(random_segment)]$. A brief analysis of the 2F mitigation assuming fixed threshold, distance, and fragment length.

First, a vector called *ideal_file* containing o random bytes is created, which is used as a reference fragment for performing the Differential Area analysis. For each subfragment $i \in [i \cdot 8]$, first the algorithm computes the entropy. Then each entropy value is used to calculate the Differential Area for the header *areaH* and for the fragment selected at a random position *areaR*. Then, given the distance parameter d, the algorithm subtracts d from *areaH*. If $areaH - d$ is less than *areaR* then, *areaH* is chosen and assign to *area*, otherwise *areaR* is selected and assigned.

The distance d helps to increase the weight of the header on the final result, since Venturini et al. [3] observed that in some occasions (such as with unencrypted files) the header tends to be more reliable than the result obtained from the random fragment. In fact, they state that without subtracting d, the algorithm would correspond to simply choosing the smaller Differential Area, hence the result that yields the greatest entropy, implying more false positive for high entropy legitimate files.

Finally, the algorithm compares the selected *area* to the threshold t and returns *True* if the area is smaller than t, signalling the presence of a ransomware. Otherwise, it returns *False*. For the pseudocode of the 2F refer to [3].

3-Fragments (3F) Countermeasure

In this approach, instead of analyzing the header and one additional fragment selected at a random point of the file, two fragments were analyzed in addition to the header.

As previously described, the vector *ideal_file* is created containing o random bytes. Then, two segments at random positions of length o are sampled. Similarly to the 2F, the algorithm computes the Differential Area of the header *areaH* and of the segments $areaR1$ and $areaR2$. But in this case, since there is more than one segment, the area taken is the average of the two random areas, $areaR1$ and $areaR2$, called *avg*. Then, given the distance parameter *d*, the algorithm subtracts *d* from *areaH*. If $areaH - d$ is less than $areaR$, *areaH* is stored inside *area*, $areaR$ otherwise.

The main difference between 2F and 3F lies in how the Differential Areas are used. In 2F, the Differential Area of the header was directly compared to the Differential Area of the (only) additional fragment. In 3F, the Differential Area of the header fragment *areasH* minus the distance *d* (i.e., $areasH - d$) is compared with the average *avg* of the Differential Areas of the additional two random fragments. Again, the smaller value is stored inside *area* and finally compared with the threshold *t*.

To better understand how the fragments are chosen, the example used by Venturini et al. [3] could be useful. Assume that the Differential Area calculated for the header is $D_H = 71$, for the first random fragment is $D_{F1} = 32$ and for the second random fragment is $D_{F2} = 54$. In this case, the predefined value for the threshold is $t = 46$ and for the distance is $d = 35$. The goal is to select one of the three Differential Areas and compare it with the threshold to determine if the file is suspicious or not. The selection process follows these steps:

1. Compute the Differential Areas of the random fragments and calculate the average *avg*. In our example, $(D_{F1} + D_{F2})/2 = (32 + 54)/2 = 43$;

2. Subtract distance *d* from the Differential Area of the header D_H, $D_H - d = 36$. Then, compare the latter result with $avg = 43$;

3. $D_H - dist$ is smaller than *avg* (i.e., $36 < 43$). Therefore, the Differential Area selected is $D_H = 71$.

Last, to classify the file as encrypted or unencrypted, the algorithm compares the selected area with the threshold. Referring to our example, $D_H = 71$, $t = 46$. Since $D_H > t$, the file is classified as unencrypted.

The example showcases the importance of introducing the distance value for preventing the occurrence of false positives. In this example, without the use of a *d* parameter, the minimum value chosen would have been $avg = 43$ which is smaller than the threshold $t = 46$, leading to erroneously classify the normal PDF file as an encrypted file. This scenario is not unusual, as in normal files the entropy tends to grow larger with the increasing distance between a random fragment and a header fragment. Therefore, the distance parameter helps to give a higher weight to the header fragment as it is generally more reliable for classification purposes, especially with unencrypted files. For the pseudocode of the 3F please refer to [3].

4-Fragments (4F) Countermeasure

The steps are the same as the 3F countermeasure explained in Subsection 3.4.5, with the only difference that, while 3F uses the header and two additional random fragments, 4F analyses the header and other three random fragments. For the pseudocode of the 4F, please refer to [3].

The mitigations implemented by Venturini et al. [3] are similar to the DAA of Davies et al. [1] but with an added parameter called distance, which makes the computation explode cubically since the DA elements now need to be tested for each distance and for each threshold. Like the DAAneededrow, these algorithms were parallelized thanks to the TensorFlow methods that are based on matrix computations. This time the parallelization is a bit more complicated: First the selected file header is expanded into a matrix where each row is the header minus the distance, for all the distances, obtaining a matrix that has the number of rows equal to the number of distances. Then, since each row needed to be tested with all the threshold values, every row is repeated as much as many threshold values there are. Lastly, the threshold vector is repeated until it match the rows dimension of the matrix. The resulting matrices are then compared, thus evaluating the different distances with each threshold, significantly reducing the computation time.

Since the results are very similar, for the sake of the reader, please refer to the pseudocodes of [3]. They implemented the slower, less efficient version, but is significantly easier to read and understand.

3.4.6 Machine Learning

A benefit of using machine learning and one of the main reasons behind this thesis research is that in theory the machine learning models are able to learn from known data and generalize for unknown data. In the case of ransomware classification, this means that the models could predict unseen ransomware strains.

In this thesis, the problem of ransomware classification was formulated as a binary classification problem, with the ransomware encrypted files labeled as 0 and the legitimate files labeled as 1.

The approach used in this thesis to tackle the problem of learning is divided into three steps. The first step taken is the training step, then the validation step and ultimately the test step.

The training is needed to allow the algorithm to learn the relation between data, the validation step is performed on different data and used to check the hyperparameters (parameters specific for each model) values and tune them accordingly, and the test is used to evaluate the model and how it would behave with new unseen data.

In the specific case of this thesis, some data preprocessing was also performed before the training phase. Instead of feeding directly the files' bytes into the algorithms, the objective was to compute the Entropy of the files' header and the DAs as defined by Davies et. [1] and by Venturini et al. [3]. The Davies et al. selection strategy focused only on the header, while the Venturni et al. selection strategies sampled the header and some random segments, as described in Subsection 3.4.5.

From now on, the collection of only the header is defined as H, the header and a random segment will be referred as H+RS, the header and two ransom segments as H+2RS and the header and three random segments as H+3RS. Finally, to end the preprocess phase, for each segment, the Entropy and the DA as defined by Davies et al. [1] were computed.

Subsequently, the selected features were tested with three machine learning algorithms in an ascendant fashion: At the beginning, only the header features were considered. Then the features of H+RS were tested. Then it was the turn of the H+2RS, and in the end the H+3RS were evaluated.

The machine learning algorithms used during all the tests were SVC (a subclass of SVM specifically for Classification), Random Forest and Neural Network.

Given that the NapierOne dataset is unbalanced in favor of legitimate files (75%) w.r.t. ransom encrypted files (25%) undersampling, which is the practice of reducing the size of the over-represented class to the size of the under-represented, was tested. However, noticeable results were evaluated only on the Neural Network, therefore SVC and Random Forest are trained on the totality of the original dataset.

After the proposed features, features selection strategies and models were tested via the NapierOne dataset, the same parameters were tested on the Attack dataset, which simulate three neutralization strategies and is used to demonstrate the robustness of the models.

Preprocessing

Given that the files in the datasets are by definition a sequence of bytes, some preprocessing was needed to provide the Entropy and the DA as input features for the machine learning algorithms. A general scheme is shown in Figure 3.2.

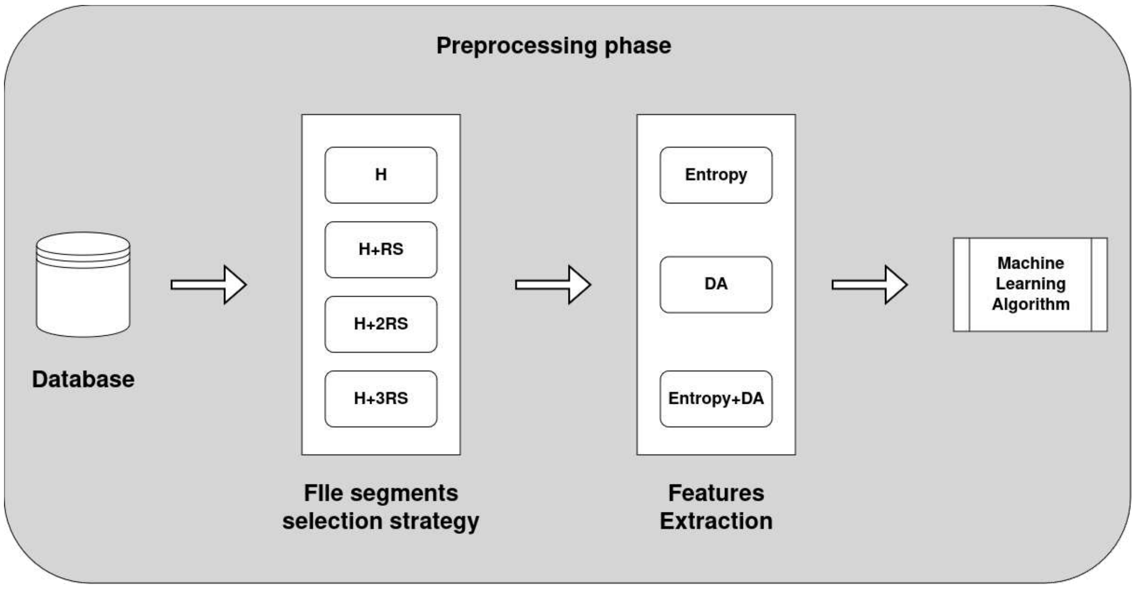


Figure 3.2: Data preprocessing phase to compute machine learning algorithm input features

The first step was sampling a given number of segments. Such segments could be only the header, referred as the decision strategy H, the header and a random segments called H+RS, header and two random segments identified by H+2RS and the header with three random segments as H+3RS. Together with the choice of the selection strategy, this first part included also the choice of the length of the segments, which started from a minimum of 8 bytes to a maximum of 256 bytes.

Once the segments and their lengths were chosen and sampled, the Entropy, the DA or a combination of both were computed for each segment separately. In the case of the Entropy, the computation was performed as described in Subsection 3.4.1 with the Algorithm 3.1, while the DA was computed as defined in Subsection 3.4.2 by the Algorithm 3.2. An example to better understand this process could help the reader.

Let's suppose that the selection strategy was the H+2RS with a length of 32 bytes. This means that the header and two random segments 32 bytes long were collected from each file in the dataset, for a total of three segments. Then the Entropy or the DA were computed for each segment, starting from the header. Imagine that the feature chosen to be computed was the Entropy. Since it is computed at 8 bytes increment re-starting from the beginning of the segment each time, the computed values are 4. The first is for the bytes from 0 to 8, the second is for the bytes from 0 to 16, the third for the bytes from 0 to 24 and the fourth for the bytes from 0 to 32. The described computations are shown in Figure 3.1. This process was repeated for all the segments, obtaining a vector composed of 4 elements for all the 3 segments, thus a 12 elements vector. Similar steps were performed if the chosen feature was the DA, computed as described in Subsection 3.4.2. If the feature chosen were Entropy and DA combined, the resulting vectors are simply attached to one another.

The maximum number of possible input features is with the H+3RS segments selection strategies with segment lengths of 256 bytes and the use of both the Entropy and the DA as features. This is because, sampling 4 segments of 256 bytes obtains 32 values of Entropy and 32 values of DA for each segment, resulting in 64 times 4, thus 256, features.

After the features were computed, the resulting vector was provided in input to the machine learning algorithms.

SVC

The Support Vector Classification (SVC) is a subclass of the Support Vector Machine (SVM) algorithms.

The training was performed on the training split of the NapierOne or the Attack dataset and can be summarized in four steps:

1. The parameters are initialized
2. The kernel matrix is precomputed
3. The construction of the best decision function that best separate the data
4. The return of the model.

A general idea of the training can be extracted from the pseudocode Algorithm A.1, found in the Appendix A.

The validation phase is performed using the validation split of the datasets and allowed to tune the hyperparameters. The hyperparameters used to define the model and their meanings can be found at `https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html`. The main one to be noted here is the C parameter, which indicates the regularization applied. The higher the value the lower the regularization is, obtaining "softer" margins SVC, vice versa otherwise. For the difference between soft and hard margin, please refer to Chapter 2. The used value for C was 1.0.

All the other hyperparameters values were provided by the default values of the Scikit-learn library, please refer to the aforementioned URL for more information.

The testing phase was used to evaluate how the model would behave with unseen ransomware-encrypted and legitimate files. The test split of the datasets was used.

Random Forest

Random forest training design is quite straightforward from the theory. It was performed on the training split of the NapierOne or the Attack dataset and can be summarized as:

1. Initialization of the empty decision tree list
2. Filling of such list with different trees trained on different bootstrap sample
3. Construction of the Random Forest model using the decision trees list just built
4. Return of the trained model.

The training phase of a random forest is briefly reported thanks to the following pseudocode Algorithm A.2 which can be found in Appendix A.

The validation phase is performed using the validation split of the datasets and allowed to tune the hyperparameters. The hyperparameters used to define the model and their meanings can be found at `https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html`. The main one to be noted here is the *n_estimator* parameter, which indicates the number of trees in the forest.

All the other hyperparameters values were provided by the default values of the Scikit-learn library, please refer to the aforementioned URL for more information.

The testing phase was used to evaluate how the model would behave with unseen ransomware-encrypted and legitimate files. The test split of the datasets was used.

Neural Network

Neural Network are a bit more complicated and involve more hyperparameters and steps during the training. The general idea is to update the weights in order to minimize the loss function. The steps can be briefly summarized as:

1. Definition of the network structure,
2. Weights and biases of each layer are initialized,
3. Forward propagation, computation of the loss and backward propagation for a certain number of times equal to the specified epochs

4. Trained model is returned.

The implemented Neural Network model is composed of an Input layer which size change based on the number of features in input, computed as described in Subsection 3.4.6.

Then there is one hidden layer of fixed size equals to 256 nodes, all defined with the ReLU activation function, and lastly the output layer which has only one node with the Sigmoid as the output function. The functions are explained in Section 2.5.

As mentioned before, to balance the dataset and improve the results, a specific type of undersampling was performed. At each epoch, all the ransomware files and an equal number of random legitimate files were selected. This was repeated for 500 epochs and paired with the early stopping practice with max tolerance of 10 epochs on the F1 score implemented. Even if for each epoch the dataset was undersampled, since the selection of the files was random, configuring many epochs allow to almost deterministically train with all the legitimate files sooner or later. And if the model was starting to overfit the data, the early stopping mechanism stops the training and pick the weights that had the best F1 score.

The chosen batch size was of 256 elements.

The training phase was performed on the training split of the NapierOne or the Attack dataset, and its objective is to learn the weights by minimizing the Loss function. Since the implemented problem is a binary classification problem, the chosen function to reduce is the Binary-Cross Entropy, defined in Figure 2.5.3. The training phase pseudocode of a Neural Network is briefly reported Algorithm A.3, in Appendix A.

The validation phase is performed using the validation split of the datasets and allowed to tune the hyperparameters of the model. Among others, some are the activation function, number of layers and number of neurons for each layer. For a complete list of the possible hyperparameters, please refer to `https://keras.io/api/`.

The test phase in the binary classification problem is used to evaluate the model by counting how many files are correctly labeled. The test split of the datasets was used.

4 Evaluation

In this chapter, the results obtained from the experiment steps described in Section 3.2 are discussed.

The results of the first step, described in Section 3.2, are discussed in Section 4.1 and are needed to prove the correct computations of the used features.

The results of the second step, a definition of which can be found in Section 3.2, are discussed in Section 4.2. Such findings are used to prove the usefulness of the used features in classifying known ransomware strains.

The third step's results, always described in Section 3.2, are discussed in Section 4.3. These were used to prove the effectiveness of using random segments sampling strategies to compensate for ransomware deploying header entropy normalization techniques, while maintaining good information if used with the known ransomware strains.

The fourth and fifth step's results, which are defined as the others in Section 3.2, are discussed in Section 4.4. These results are used to assess the performances of the different hyperparameters: features used, number of segments collected and algorithm combinations. The ones that performed well enough on both the known ransomware and the Entropy normalizing ransomware were highlighted as possible foundations for a ransomware detection method.

4.1 Entropy, DA and DAA Correctness

In this section, the step one reported on the Design in Section 3.2 is shown and proved. The proof that the Entropy, the DA and the DAA were computed correctly is provided.

Since the Entropy, the DA and the DAA were rewritten from scratch, to prove that they work as intended by the original author Davies et al. [1] is sufficient to compare the accuracy results obtained using the rewritten DAA and the original DAA on the same dataset.

However, since the dataset used on the original work from Davies et al. [1] has prohibitive dimensions, testing the rewritten DAA on that precise dataset would bring this thesis out of the original scope, recreating the original study by Davies et al. [1]. Fortunately, Venturini et al. [3] obtained similar results to the one of Davies et al. [1] but on a smaller dataset, in this thesis referred as OldNapierOne.

Thanks to Table 4.1 it is possible to see that the rewritten DAA and the original DAA obtain similar results, around 98% accuracy, with the rewritten method actually obtaining slightly better results. The difference in the parameters and the slight difference in the results is probably due to both the complexity oriented improvements that were applied during the rewrite of the DAA and the different libraries/variable types used.

Thanks to this comparison, it is possible to safely state that the Entropy, the DA and the DAA are computed as defined by Davies et al. [1]. The Entropy, the DA and the DAA correctness is proved.

If the reader is curious on how the accuracies change for the rewritten DAA based on different header lengths and thresholds, please take a look at Figure A.2 in Appendix A. Instead, for the original DAA, please look at Figure A.3 always in Appendix A.

4.2 DAA

In this section, the second step reported on the Design in Section 3.2 is shown and proved. First is proved that the DAA still works for the known ransomware strains, implying that the Entropy and the DA, later used by machine learning algorithms, carry relevant information for ransomware classification. Then, simple entropy neutralization techniques are tested and the DAA is rendered useless, thus the need for stronger defense mechanism is proved.

Due to the proof in Section 4.1 the rewritten DAA has been demonstrated to work as intended by the original writers Davies et al. [1].

Table 4.1: Comparison of the best results obtained by the handwritten DAA and the DAA by Venturini et al. on the OldNapierOne dataset

	Rewrote DAA	DAA by Venturini et al.
Length	216	152
Threshold	124	40
Accuracy (%)	98.55	98.24
Precision (%)	99.88	99.30
Recall (%)	95.44	95.00
F1 (%)	97.61	97.10

Thanks to this result, we can now test the rewritten DAA on the newer dataset NapierOne and on the Attack Dataset. Testing the rewritten DAA on the NapierOne dataset has a double scope. First, if the DAA is able to perform well enough, it means that the Entropy and the DA, from which the DAA and the proposed machine learning models depend on, still carry relevant information for the ransomware classification problem. Second, it allows the comparison between different approaches, one more mathematical and the other more machine learning oriented.

Testing on the Attack dataset is useful to prove that ransomware detection methods Entropy-based are prone to Entropy normalization techniques and if a technique used some form of Entropy values it has to take in account this vulnerability, otherwise is rendered useless.

Figure 4.1 shows the results for different header lengths on the NapierOne dataset and Figure 4.2 for the Attack dataset. At Table 4.2 the best results in terms of F1 score is displayed.

From Table 4.2 it is clear that in the case of known ransomware strains the DAA still performs really well, obtaining 94.37% accuracy, and thus the Entropy and the DA carry relevant information. But if some ransomware strains were able to modify the headers' entropy of the encrypted files, the DAA and thus a mathematical interpretation of the Entropy and the DA, would be rendered useless, as demonstrated by the DAA having 58.45% accuracy on the Attack dataset. Thus, the need for stronger ransomware detection methods arise.

4.3 Mitigations

In this section, the third step defined in Section 3.2 is proved. This is done to understand if random segment selection strategies are useful against Entropy neutralizing strategies focused on the header.

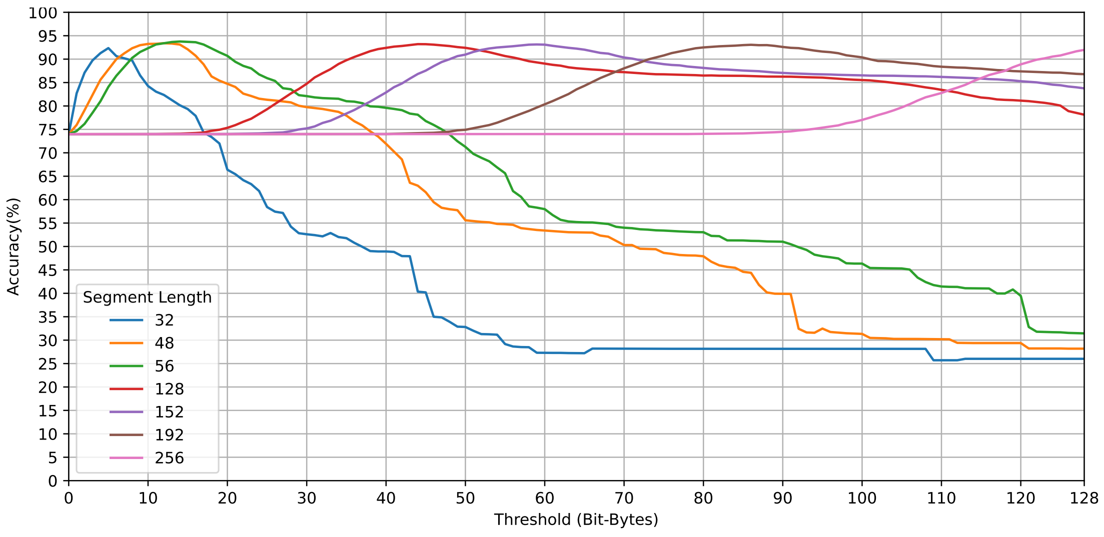


Figure 4.1: Accuracy of the rewritten DAA on the NapierOne Dataset

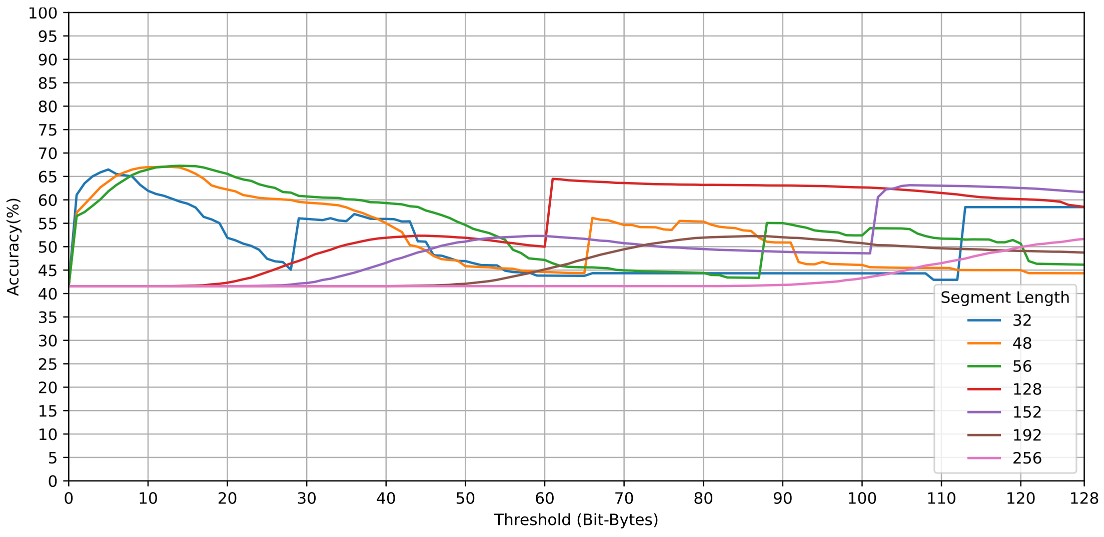


Figure 4.2: Accuracy of the rewritten DAA on the Attack Dataset

Table 4.2: Comparison of the best F1 Score results between NapierOne and Attack dataset.

	DAA on NapierOne	DAA on Attack
Length	80	8
Threshold	23	12
Accuracy (%)	94.37	58.45
Precision (%)	94.41	58.45
Recall (%)	84.31	100
F1 (%)	**88.63**	**73.78**

The idea behind the proposed mitigations by Venturini et al. [3] was to strengthen the DAA proposed by Davies et al. [1].

In this thesis, they were implemented to verify if the random segments sampling strategies, implemented as defined by Venturni et al. [3], were useful in countering Entropy neutralizing techniques and thus carry relevant information. This proof is useful because such sampling strategies will be later used to select the features for the machine learning algorithms. Also, by obtaining the results on the same datasets as the machine learning algorithms allows comparing different detection methods.

Given that the mitigations involves the selection of random segments, multiple runs are needed, which in the case of this thesis are ten, in order to exclude outsiders and effectively evaluate the performances.

Since ten runs were executed, the reported metrics are chosen by picking the triplet of parameters (length, distance and threshold) that obtain the best result on average in terms of F1 score. Such results can be found at Table 4.3. The full graphs for all the threshold and distance values are shown in Figure A.4 for the 2F, in Figure A.5 for 3F and Figure A.6 for 4F, located in the Appendix A.

All the three mitigations performs similarly, with the 3F having slightly better results, which is coherent with the findings of Venturini et al. [3]. A peculiarity that is worth nothing is that, even if for different runs the best distance and threshold in F1 score terms slightly change, the best performing length was always 72 bytes an all the ten runs.

All the three sampling strategies obtain an F1 score close to 90% as shown in Table 4.3, thus the usefulness of the random segments selecting strategies is proved.

Having verified that in the specific case of ransomware strains which neutralize the entropy of the header the mitigations performs well, there is the need to test if such methods perform well for the known ransomware or if their usefulness is limited to this particular attack scenar-

Table 4.3: Mitigations mean results and standard deviation of ten runs on the Attack dataset

	2F	3F	4F
Length	72	72	72
Distance	114	113	115
Threshold	21	20	20
Accuracy (%)	87.55 ± 0.19	88.02 ± 0.11	88.16 ± 0.21
Precision (%)	85.87 ± 0.17	87.03 ± 0.10	87.88 ± 0.18
Recall (%)	94.20 ± 0.19	93.42 ± 0.12	92.50 ± 0.22
F1 (%)	**89.84 ± 0.15**	**90.11 ± 0.09**	**90.13 ± 0.18**

Table 4.4: Mitigations mean results and standard deviation of ten runs on the NapierOne dataset

	2F	3F	4F
Length	40	40	40
Distance	72	74	72
Threshold	9	20	20
Accuracy (%)	93.22 ± 0.09	93.47 ± 0.06	93.58 ± 0.06
Precision (%)	86.68 ± 0.27	87.61 ± 0.14	88.01 ± 0.15
Recall (%)	87.35 ± 0.14	87.25 ± 0.11	87.19 ± 0.10
F1 (%)	**87.02 ± 0.17**	**87.43 ± 0.11**	**87.60 ± 0.10**

ios.

All the three sampling strategies obtain an F1 score close to 87% as shown in Table 4.4, thus it is proved that the random segments selecting strategies works also with the known ransomware strains.

The results were selected in the same way as what is described in the previous paragraph. As before, the best run results for different distance and threshold values, are located in Appendix A and shown in Figure A.7 for 2F, in Figure A.8 for 3F and in Figure A.9 for 4F.

Similarly to the peculiarity reported before, the distance and the threshold that performed the best during the ten runs changed, but the best sequence length was always 40 bytes.

4.4 Machine Learning Algorithms

In this section, the results from step four and five from the experiment design explained in Section 3.2 are reported.

The problem of ransomware classification was formulated as a binary classification problem, where the files were either labeled as ransomware or as legitimate. Then the header and some random segments were extrapolated, and the Entropy or the DA extracted and provided as input features both singularly and in combination (referred to as Entropy+DA).

In particular, in this section when only the header is used to extrapolate the features it will be referred as H, in the case of the header and a random segment H+RS, the header and two random segments as H+2RS and finally the header and three segments as H+3RS.

Thanks to the results obtained in Section 4.2 for the DAA and in Section 4.3 for the mitigations, it is possible to infer that the Entropy and the DA can be used as features to effectively distinguish between ransomware and legitimate files. In particular, in the case of H+RS, H+2RS and H+3RS, the Entropy and DA carry information both for the know ransomware and for the entropy neutralization ransomware. Instead in the case of only H, when neutralization strategies are applied, selecting only the header do not obtain substantial results. This mean that in theory, in the case of the known ransomware strains, all segments selection strategies could work if paired with machine learning. Instead, if a ransomware deploy entropy neutralization techniques, only the H+RS, the H+2RS and the H+3RS selection strategies should work.

The chosen algorithms to test the features were SVC, Random Forest, referred to as RF, and Neural Network shorted to NN. As for the mitigations, to exclude outsiders ten runs were executed for each types of segments selection (H, H+RS, H+2RS, H+3RS, thus 4), for each segment length (from 8 to 256 bytes with 8 bytes jumps, thus 32), for each type of feature extracted (Entropy, DA or Entropy+DA, thus 3), for each algorithm (SVC, RF and NN, thus 3) and for each and for datasets (NapierOne and Attack dataset). In total 7680 runs per algorithm were performed.

For the sake of clarity, in this section it will be reported only the general graphs based on the best averaged metrics values of the ten runs for each algorithm with both the NapierOne and Attack dataset. If the reader is interested on seeing the accuracy of a specific combination of segments selection strategy, feature extracted and algorithm used, please refer to the graphs in Appendix A. Keep in mind that each point in the graph is the mean of ten runs. The standard deviation is also reported.

Of all the metrics, the results discussed analyze the F1 Score, which is generally a good benchmark to evaluate a machine learning algorithm performance, for the accuracy refer to Appendix Appendix A.

Before diving deep to the single case analysis, it could be useful to compare the best results for each algorithm for the different datasets, always remembering that the DAA and the mit-

Table 4.5: Machine Learning best mean F1 Scores paired with the standard deviation on the NapierOne dataset

	SVC	NN	RF
Segments	H	H	H+3RS
Segments Length	152	48	144
Statistic	Entropy	Entropy+DA	DA
F1 Score (%)	**90.55 ± 13.07**	**91.62 ± 2.41**	**94.09 ± 1.33**

igations results are the best value on the training split of such datasets, while for the machine learning algorithms all the values are computed from the test set, which means on new unseen data. Nevertheless, it is still interesting comparing how and hypothetical mathematical approach would perform against a machine learning one.

Taking into consideration the NapierOne dataset, which represent the known unmodified ransomware strains, a quick comparison can be performed by looking at the best results of the DAA Table 4.2, of the mitigations Table 4.4 and of the machine learning algorithms Table 4.5. The NN obtained similar result to the 2F, 3F and 4F algorithms, while the RF was closer to the DAA. Therefore, the use of the Entropy+DA with the NN and the DA with the RF can be considered as a plausible working combination for the known ransomware strains.

For the Attack Dataset, which represent ransomware using header entropy neutralization technique, the DAA is not effective. However, all the features performed similar to the mitigations proposed by Venturini et al. [3], regardless of the machine learning algorithm used.

These results have an important implication. It seems that when using machine learning, a ransomware implementing entropy neutralization techniques can be effectively detected independently of the use of Entropy, DA or Entropy+DA. This means in order to have a working defense mechanism that uses Entropy, DA or Entropy+DA is sufficient to select the best performing feature-segments-algorithm combo against known ransomware strains and eventually train the model on encoded segments.

Two possible candidates which can be used to build a quick and reliable defense mechanism to detect both known ransomware and entropy neutralization ones is the DA-H+3RS-RF combo, as show by looking at Figure 4.5 and Figure 4.6 or the Entropy+DA-NN combo with one of all the segments selection strategy as shown by looking at Figure 4.7 and Figure 4.8.

Table 4.6: Machine Learning best mean F1 Scores paired with the standard deviation on the Attack dataset

	SVC	NN	RF
Segments	H	H	H+3RS
Segments Length	112	40	184
Statistic	Entropy	Entropy	Entropy + DA
F1 Score (%)	**95.29 $\pm$ 5.71**	**95.82 $\pm$ 2.46**	**97.09 $\pm$ 4.91**

4.4.1 Entropy

In this subsection, the use of Entropy as a feature taken by either the header or by the header and some random segments along the file is explored and evaluated. It was tested both against the known ransomware strains via the use of NapierOne and also against new ransomware strains that target Entropy specifically via the Attack Dataset.

From Figure 4.3, it's possible to see that on average, extrapolating the Entropy only from the header performs poorly with the chosen Neural Network. Little improvements can be noted when more segments are collected, with the best results obtained with the H+2RS. It is possible that this bad performances, given that all the selection strategy has similar F1 scores, could be due to the used NN model needing more tuning of the hyperparameters.

The Support Vector Classification (SVC) and the Random Forest (RF) algorithms obtain better results on average but have high variance between different runs, signaling high instability on the outcome when the Entropy is used as feature.

Both the SVC and the RF performs better only when the header is used, which means that in the case of known ransomware and files the biggest entropy values difference and thus information can be founded on the files' header. Specifically, the SVC obtain the best result on the H-Entropy combination, as reported Table 4.5. This is coherent both with the findings of Davies et al. [1], and with Venturini et al. [3] which observed a decrement in Accuracy when other random segments were collected along with the header, as opposed to the use of the header only.

However, while with the SVC algorithm, the more segments were collected the less the F1 Scores obtained were, the Random Forest algorithm, even if it obtained lower scores, it maintained similar enough performances.

Due to the high variance between different runs, the use of the Entropy, is not a suitable starting point to build a defense mechanism based on SVC, NN or RF. This is true independently of the segment selection strategy used.

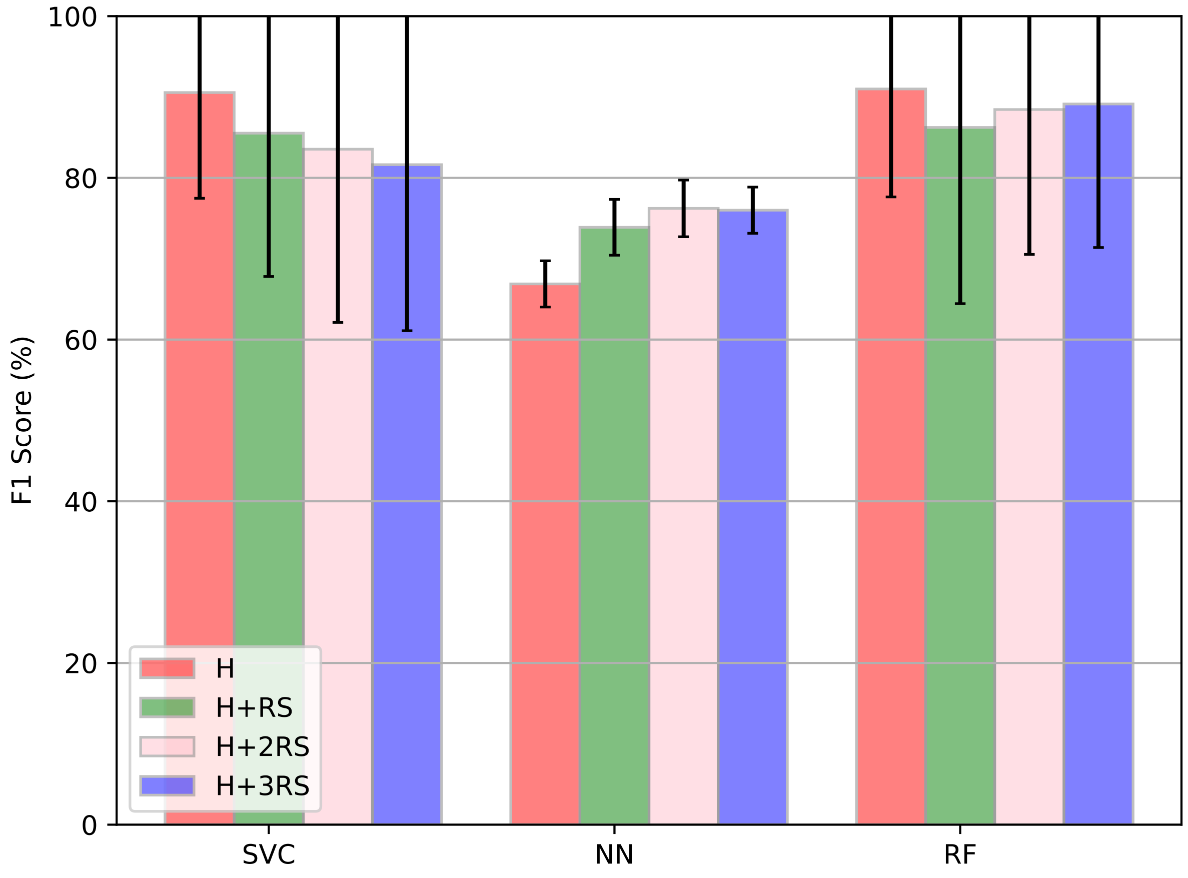


Figure 4.3: Best average F1 Scores for Support Vector Classification (SVC), Neural Network (NN) and Random Forest (RF) for each segment extraction strategy and Entropy as feature on the NapierOne dataset

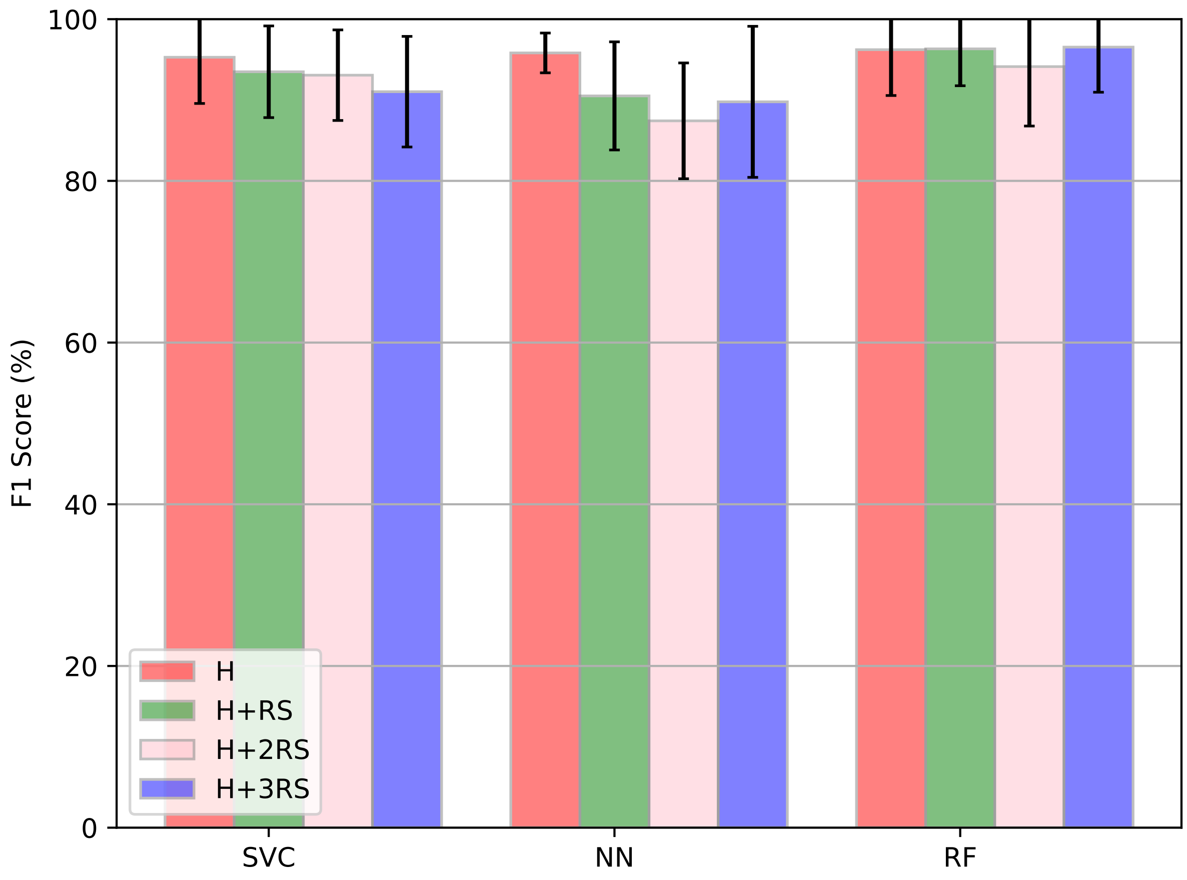


Figure 4.4: Best average F1 Scores for Support Vector Classification (SVC), Neural Network (NN) and Random Forest (RF) for each segment extraction strategy and Entropy as feature on the Attack dataset

On contrary of what the attacks have obtained for the DAA by Davies et al., which required some strengthen strategies like the ones proposed by Venturini et al. in the case of the three machine learning algorithms chosen they simplified the problem and the algorithms performed better with the Attack Dataset than with NapierOne dataset, basically stating that such neutralization attacks are nullified by the use of machine learning algorithms. Results for the use of the Entropy combined with the Neural Network for the attacks can be found at Figure 4.4.

Regarding the attacks, Random Forest on average obtained slightly better results than the other algorithms. However, on the specific case of sampling only the header (H), the Neural Network and the RF had a similar result, but the NN had less variance between the runs. This could mean that in this particular case, the model is steadily able to learn the relation between the extracted Entropy features and the label, converging to a meaningful model.

Thus, in the case of the simulated attacks, the Entropy combined with the used segments selection strategies can generally be used to effectively distinguish between modified ransomware and legitimate files, with some specific optimal results. In particular, the best performances obtained by the SVC and NN algorithms when trained and tested on the Attack Dataset are with the use of the Entropy, as depicted on Table 4.6. However, when using the same features and selection strategies for the real ransomware strains, even if there are some good scores, the variance between the runs is too big to consider this feature reliable in real world scenarios.

4.4.2 DA

In this subsection, the DA as feature for different machine learning models is explored.

It was tested both against the known ransomware strains via the use of NapierOne and also against new ransomware strains that target header Entropy specifically via the Attack Dataset.

Aside from some specific cases, using only the DA performed worse than the use of the Entropy for all the algorithms, as illustrated by comparing Figure 4.5 and Figure 4.3.

By looking at Figure 4.5 it is possible to see that the SVC algorithm, independently of the segments selection strategy used, had a lot of variance between different runs. Therefore, the use of the DA as a feature paired with the SVC algorithm can not be reliably used to distinguish between ransomware encrypted files and legitimate files.

The same reasoning can be applied to the NN algorithm with the specific case of the H+2RS segment selection strategy, which has a low variance value. However, since the corresponding F1 Score is slightly above 80% it can not be safely stated that such parameters' combination can be effectively used for ransomware classification.

In the case of the RF algorithm, similar conclusions to the previous paragraph can be formulated for the segments' selection strategies H, H+RS and H+2RS. However, unlikely as all the other results obtained with the DA as a feature, the H+3RS segments selection strategy paired with the DA has very good performances with the RF. Not only the average result is high, but also the variance is small, implying that if trained correctly, it could be used as a part of a ransomware defense mechanism in a real world scenario.

This result is also reported in Table 4.5, which means that for RF this is the best results among all the possible parameters combinations and in particular among all the tested features.

Given the RF results with the combination of DA and H+3RS, it seems that such features carry useful information and could be used to build a working defense mechanism. However, the current SVC and NN models are not able to detect meaningful correlation. Future work could focus on this DA and H+3RS features combo and better tune the SVC and NN hyper-parameters to improve results.

As in the prior case, the implemented attacks are countered by the use of machine learning algorithms alone. It does not matter if the extracted features are directly the Entropy values of the file or the DA values between the real and the ideal file Entropy, all the three algorithms are able to neutralize the attacks, rendering the tempering of the Entropy header ineffective against machine learning in general.

This is probably due to the simplicity of the perpetuated attacks, since they aim to simply nullify the entropy header, they probably create a new class of ransomware easily distinguishable from the legitimate files. This is because real world files, even with low entropy values, never truly touch the zero mark.

Results of the best performing F1 score are illustrated in Figure 4.6.

This implication carries an important discovery, which is that if the perpetuated attacks were simple and a ransomware detection method was based on machine learning, the attacks are quickly rendered useless.

More sophisticated attacks that temper the header, like copying and attaching the header of other legitimate files inside the file system, or the ones proposed by [6, 7, 8], would probably work better. Future works are required to distinctively address these alternatives.

4.4.3 Entropy + DA

Generally speaking, machine learning algorithms performs better if they have more data and thus features to be trained on. This is the simple idea behind combining the Entropy and the

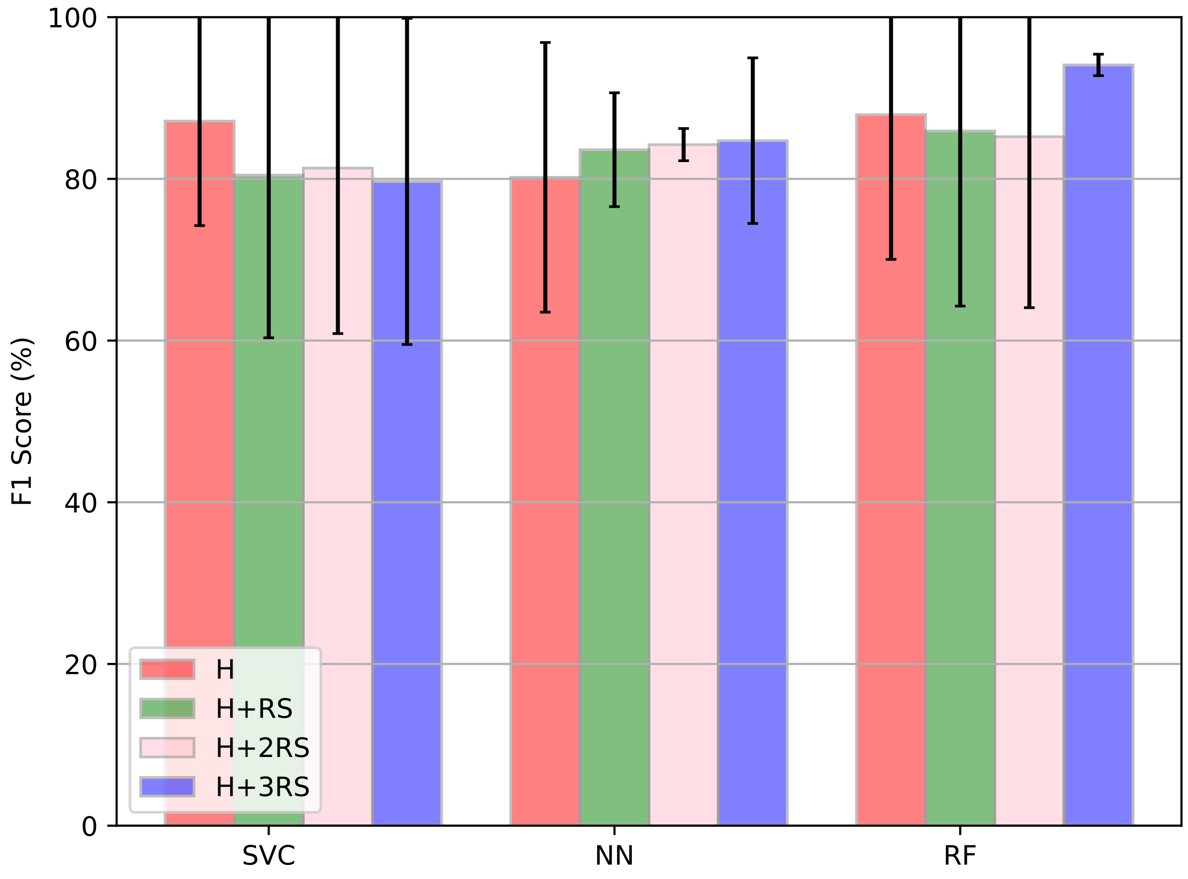


Figure 4.5: Best average F1 Scores for Support Vector Classification (SVC), Neural Network (NN) and Random Forest (RF) for each segment extraction strategy and DA as feature on the NapierOne dataset

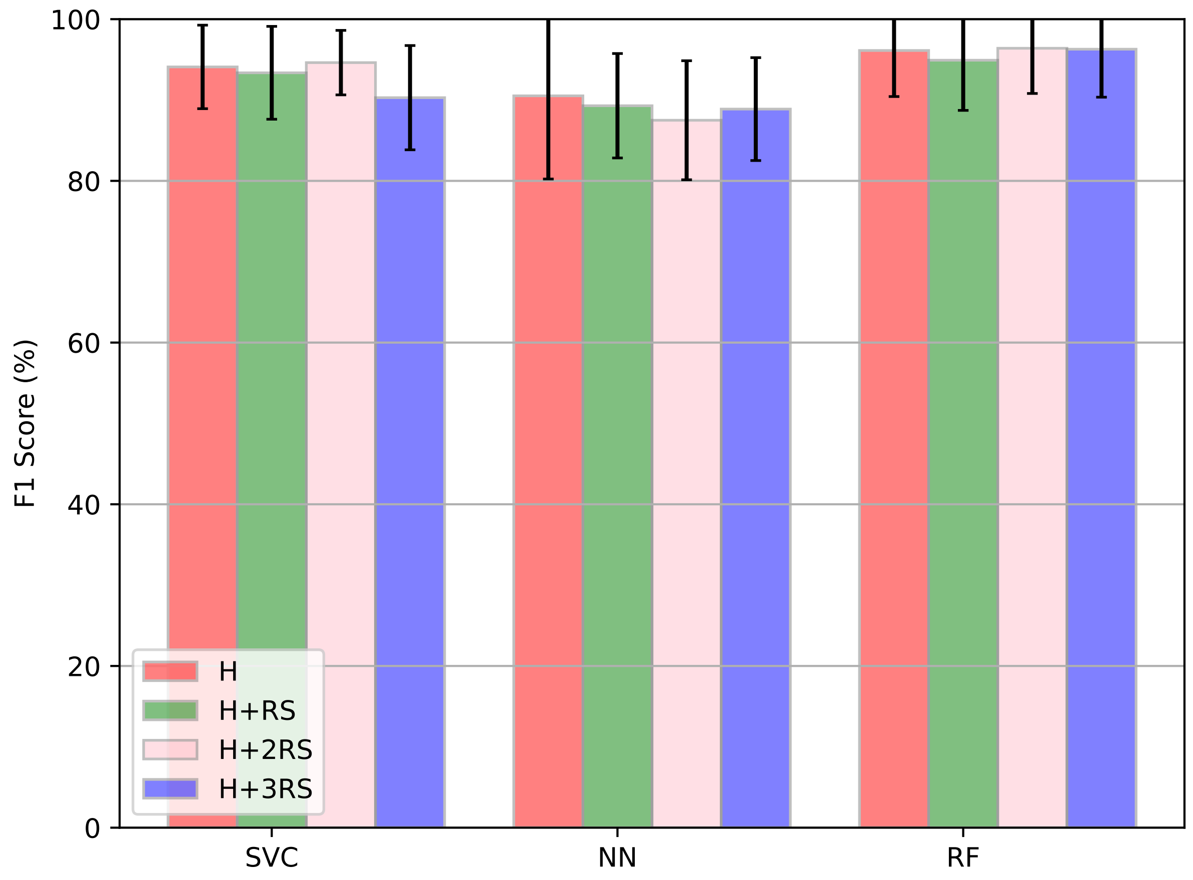


Figure 4.6: Best average F1 Scores for Support Vector Classification (SVC), Neural Network (NN) and Random Forest (RF) for each segment extraction strategy and DA as feature on the Attack dataset

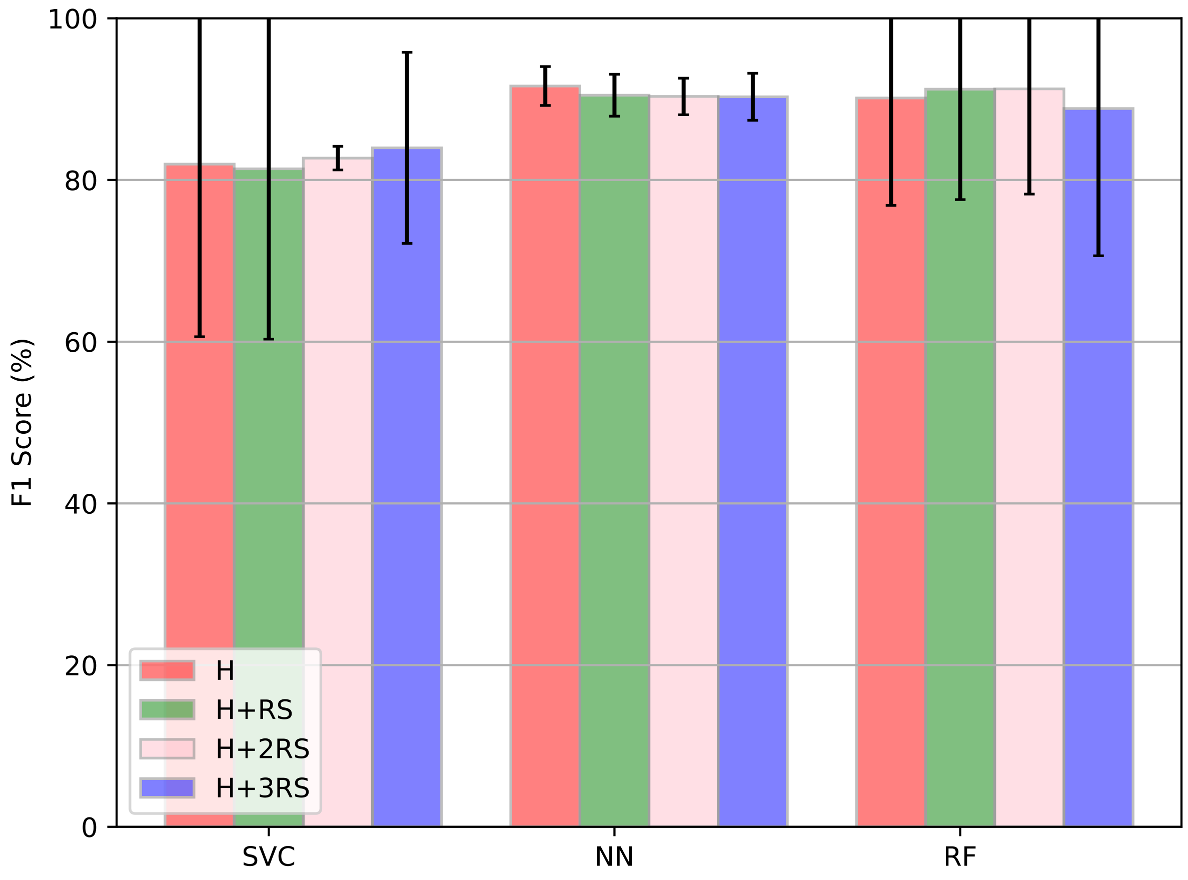


Figure 4.7: Best average F1 Scores for Support Vector Classification (SVC), Neural Network (NN) and Random Forest (RF) for each segment extraction strategy and Entropy+DA as feature on the NapierOne dataset

DAs values and give the resulting vector as input feature for the machine learning algorithms, hoping for a better file classification.

Computationally speaking, it is basically identical to using only the DAs, since the Entropy values are computed anyway, but not used.

Unfortunately, this reasoning did not translate into significant results for the Random Forest and the SVC algorithms, as shown in Figure 4.7.

The RF on average obtain better results than the SVC, however both algorithms exhibit high variance between different runs (aside for H+2RS with SVC which has a low F1 Score anyway) and so can not reliably be used to classify between ransomware encrypted files and legitimate files.

However, this is not the case for the implemented Neural Network. The combination of the

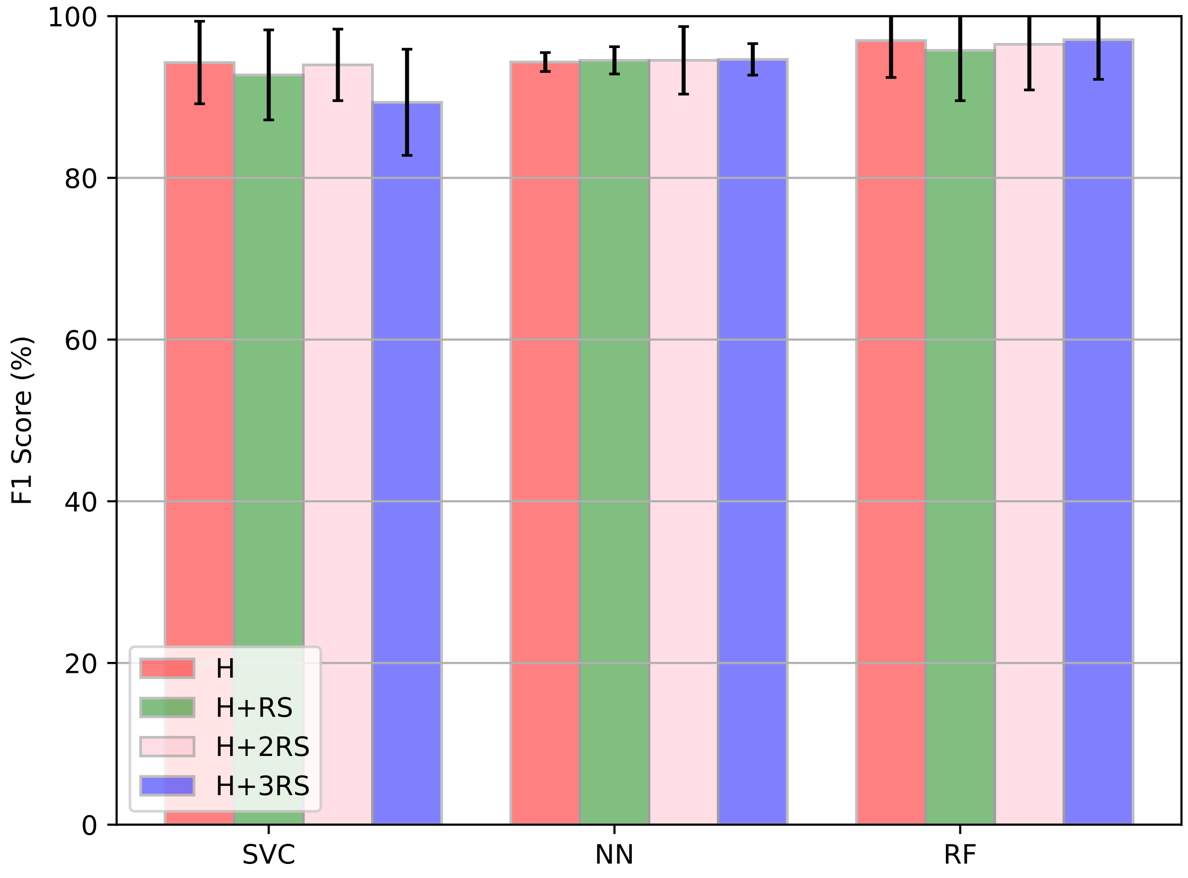


Figure 4.8: Best average F1 Scores for Support Vector Classification (SVC), Neural Network (NN) and Random Forest (RF) for each segment extraction strategy and Entropy+DA as feature on the Attack dataset

Entropy and DA obtain high F1 Scores with low variance values for all the segments' selection strategies.

This means that the algorithm is able to converge to a meaningful model independently of the collected segments, which imply that Entropy+DA features carry useful information regardless of the sampled segments. This is a meaningful result, because it could imply that a defense mechanism build on such a combination could be quick, able to effectively be used in a real world scenario and also generalize to new ransomware that specifically target the selected features.

SVC and RF probably need more hyperparameter tuning to reduce the variance and obtain better performances as NN.

As with the use of Entropy and DA singularly, the combined features are quite effective against the formulated attacks. In particular, the RF has better average performances as shown in Table 4.6 but NN, which follow shortly after in terms of F1 scores for all segments' selection strategies, has lower variance values and therefore can be considered more stable than the other models. Results are shown in Figure 4.8.

5 Conclusion

Ransomware threat has been growing steadily, targeting and damaging both public agencies and also private companies. Unfortunately, given the continuous evolution of ransomware strains, which regularly deploy new and improved techniques to evade defense mechanisms, a lite mitigation technique, able to keep up with the up-to-date ransomware changes has yet to be found.

Many frameworks, based on static analysis, dynamic analysis or a combination of both were proposed, with some of them composed of data collected from different steps. However, sooner or later, such algorithms are destined to fall victims to the evolution of obfuscation techniques and evasion methods alike.

However, given the ransomware nature and scope, independently of how much the strain is convoluted, at one point it will need to encrypt file system data. If a defense mechanism was able to distinguish between the legitimate file on the disk and encrypted files, it could detect the presence of an ongoing ransomware attack and stop it.

Given the scope of an encryption algorithm, which is to create a ciphertext from which the plaintext can not be understood, they output seemingly random values starting from any input. Since legitimate files tend to have repeated bytes due to their type structure, if such randomness could be evaluated through a metric, it could be possible to distinguish between encrypted and normal files. One of the most widely used statistic metric in ransomware classification to evaluate data distribution randomness, is the Shannon Entropy.

The truly new idea of Davies et al. [1] was not to use the Entropy for ransomware classification, which was already explored, but it was to compute the Entropy only on the file header. Such analysis was proposed because files have metadata at the start, which are bytes related to file type and are used to describe general file information. Thus, instead of considering the totality of the file, which would return Entropy values similar between encrypted and compressed files, focusing only on the header classification is faster and lighter than other proposed methods.

The results were promising, but it was not addressed what would have happened to the analysis in the case the attacker was able to tamper with the encrypted files header, and lower the Entropy. Such vulnerability was explored by Lee et al. [6, 7] and Bang et al. [8] for the entire file and by Venturini et al. [3] with a focus on the header. In particular, Venturini et al. [3], also proposed some strengthening strategies to compensate the header Entropy analysis critical flaw. Venturini et al.[3] mitigations were based on the collection of random segments along the files paired with the header so that if an attacker were able to tamper it, it would still be possible to perform the analysis.

This thesis tried to explore the plausibility of implementing a light ransomware detection method based on a machine learning algorithm and the features extracted following similar methodologies as in Davies et al. [1] and Venturini et al. [3]. The algorithms used were SVC, NN and RF. Entropy and DA of the files header were tested and a formulation based on the results was provided.

To achieve this, the correctness of the Entropy and DA computations was verified by rewriting the DAA and comparing the accuracy score with that of Davies et al. [1]. Since the Davies et al. [1] dataset was too large for a quick proof, the Venturini et al. [3] dataset was used instead. The accuracy score obtained was similar to that of Davies et al. [1].

Once this was verified, the DAA was computed on the up-to-date NapierOne dataset and on the Attack dataset. The first was done to prove the usefulness of the Entropy and the DA, used by DAA, to classify known ransomware strains, the second was implemented to prove the DAA vulnerability to neutralization techniques and the need for a new and improved method.

Then the mitigations proposed by Venturini et al. [3], basically a DAA version more resilient against header tempering, ware implemented. They were run ten times, and averaged, to prove the usefulness of random segments sampling strategies against header Entropy neutralization techniques and as a comparison benchmark with the machine learning algorithms.

Ultimately, three algorithms—SVC, RF, and NN—were implemented, and various feature combinations were tested with ten runs each. Specifically, four types of file segment selections were performed, with each segment having a length between 8 and 256 bytes, incrementing by

8 bytes. For the selected segments, the Entropy, DA, or both were computed and used as input features for the algorithms.

The Entropy feature performed poorly in terms of F1 score with all the machine learning algorithms on both datasets, having either lower average values or high variances.

The DA feature also had mediocre results for both datasets, except when it was computed on the header, and on three random file segments, and provided as input for the RF algorithm. This could mean that the DA can potentially carry useful information, and maybe it performed poorly on other algorithms due to how the algorithms were modeled, more than due to the feature itself.

The combination of the Entropy and the DA also performed well on both datasets with only one algorithm, the Neural Network. However, differently from the use of the DA alone with RF, the results remained almost the same with different selection strategies.

In both cases, DA alone and DA combined with Entropy, the results are interesting and show that if a defense mechanism was built using the RF and NN respectively it would be able to quickly and effectively distinguish between ransom encrypted files and legitimate files.

However, independently of the machine learning algorithm, no single feature performed universally better than the others. This could be either due to the limitations of the features, or of the algorithms used.

Future works could explore unsupervised machine learning algorithms, to better understand why certain features are not as good as others, focus on the best performing features, and calibrate the machine learning models to improve the results, try different supervised machine learning models or implement more sophisticated attacks to stress test the algorithms.

A
Appendix

A.1 Advanced Encryption Standard (AES)

Generally speaking, there are two sides in an encrypted communication: the sender, who encrypts the data, and the recipient, who decrypts it. Encryption is divided into two main categories, asymmetric and symmetric encryption. The former, as the name suggests, is different on each side; the sender and the recipient use two different keys. Asymmetric encryption, also known as public key encryption, uses a public key-private key pairing: data encrypted with the public key can only be decrypted with the private key.

In symmetric encryption instead, the same key both encrypts and decrypts data. For symmetric encryption to work, the two or more communicating parties must know what the key is; for it to remain secure, no third party should be able to guess or steal the key. From the data collected during the years of ransomware activity, which can be found in MITRE | ATT&CK catalogue (`https://attack.mitre.org/software/`), it is evident that when ransomware need to encrypt data on the file system they tend to use the Advanced Encryption Standard (AES) or its modern derivatives, thus it could be useful to have a brief summary of such algorithm.

AES is a block cipher based on the substitution-permutation network used for symmetric encryption. It is a specific implementation of the Rijndael block cipher with the block size limited to 128 bit and a key size of 128, 192 or 256 bits.

The AES algorithm is based on the Substitution, Transposition and Linear maps iterated ciphering model, (S,T,L) for short, with n=10,12 or 14 rounds.
The AES algorithm holds a 4 by 4 array of bytes called the state, that is initialized to the input to the cipher (note that the input is 128 bits which is exactly 16 bytes as the block size). The substitution and permutation operations (providing confusion and diffusion) are all applied to the state array. Here are the steps, briefly summarized:

1. **KeyExpansion** – round keys are derived from the cipher key using the AES key schedule. AES requires a separate 128-bit round key block for each round plus one more.
2. Initial round key addition:
   (a) **AddRoundKey** – each byte of the state is combined with a byte of the round key using bitwise XOR.
3. For 9, 11 or 13 rounds:
   (a) **SubBytes** – a non-linear substitution step where each byte is replaced with another according to a lookup table.
   (b) **ShiftRows** – a transposition step where the last three rows of the state are shifted cyclically a certain number of steps.
   (c) **MixColumns** – a linear mixing operation which operates on the columns of the state, combining the four bytes in each column.
   (d) **AddRoundKey**
4. Final round (making 10, 12 or 14 rounds in total):
   (a) **SubBytes**
   (b) **ShiftRows**
   (c) **AddRoundKey**

Figure A.1 is a visual representation of the above described steps. Decryption is performed by the inverse blocks in reverse order. For more information on the AES encryption algorithm, please refer to [47].

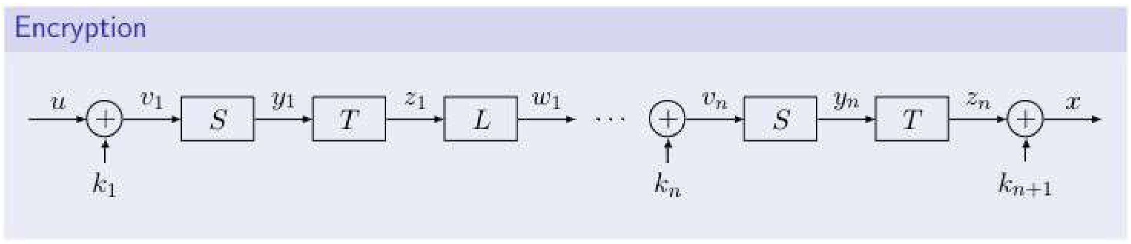


Figure A.1: AES Encryption Model provided by Professor Laurenti Nicola during the Information Security course https://didattica.unipd.it/off/2024/LM/SC/SC2542/000ZZ/SCQ0089463/N0

A.2 Pseudocodes

Algorithm A.1 SVC Training

```
Require: $X_{\text{train}}$: training data features
Require: $y_{\text{train}}$: training data labels
Require: $C$: regularization parameter
Require: kernel: kernel type (linear, polynomial, RBF, etc.)
Require: tol: tolerance for stopping criterion
Require: max_iter: maximum number of iterations (optional)
Ensure: model: trained SVC model
  Begin
  1. Initialize parameters:
    a. Select the kernel function based on the input kernel type.
    b. Initialize Lagrange multipliers ($\alpha$) to zero.
    c. Set up threshold value ($b$) to zero.
  2. Pre-compute the kernel matrix if necessary:
    $K[i,j] = \text{kernel}(X_{\text{train}}[i], X_{\text{train}}[j])$
  3. Optimization (typically using SMO algorithm):
    a. Repeat until convergence or max_iter:
      i. For each sample $i$ in $X_{\text{train}}$:
        1. Compute the decision function:
          $f(i) = \sum(\alpha[j] \cdot y_{\text{train}}[j] \cdot K[i,j]) + b$
        2. Compute the error for sample $i$:
          $E_i = f(i) - y_{\text{train}}[i]$
        3. Check the KKT conditions and select the sample to optimize.
        4. Optimize the chosen sample pair $(i,j)$:
          a. Compute the second error $E_j$ and update $\alpha[i]$ and $\alpha[j]$.
          b. Update the threshold value $b$.
      ii. Check convergence based on tolerance (tol).
  4. Construct the decision function using the optimized $\alpha$ and $b$.
  5. Return the model containing $\alpha$, $b$, and the kernel function.
  End
```

Algorithm A.2 Random Forest Training

```
Require: $X_{\text{train}}$: training data features
Require: $y_{\text{train}}$: training data labels
Require: $n\_trees$: number of trees in the forest
Require: $max\_features$: maximum number of features to consider for each split
Require: $max\_depth$: maximum depth of each tree (optional)
Require: $min\_samples\_split$: minimum number of samples required to split an internal node (optional)
Require: $min\_samples\_leaf$: minimum number of samples required to be at a leaf node (optional)
Ensure: forest: trained RandomForest model
  Begin
  1. Initialize an empty list to hold the individual decision trees.
  2. For each tree $t$ in range(1, $n\_trees$):
     a. Draw a bootstrap sample from the training data:
        $X_{\text{bootstrap}}, y_{\text{bootstrap}} = \text{bootstrap\_sample}(X_{\text{train}}, y_{\text{train}})$
     b. Train a decision tree on the bootstrap sample:
        $tree = \text{train\_decision\_tree}(X_{\text{bootstrap}}, y_{\text{bootstrap}},$
           $max\_features, max\_depth,$
           $min\_samples\_split, min\_samples\_leaf)$
     c. Add the trained tree to the list of trees.
  3. Construct the RandomForest model using the list of decision trees.
  4. Return the RandomForest model.
  End
```

Algorithm A.3 Neural Network Training

```
Require: $X_{\text{train}}$: training data features
Require: $y_{\text{train}}$: training data labels
Require: input_shape: shape of the input data
Require: output_shape: shape of the output data
Require: layers: list of layer configurations (type, units, activation function, etc.)
Require: loss_function: loss function to be used
Require: learning_rate: learning rate for the optimizer
Require: epochs: number of epochs to train the model
Require: batch_size: number of samples per gradient update
Ensure: model: trained Neural Network model
  Begin
  1. Initialize the neural network:
    a. Define the architecture:
      - Input layer: match the input_shape
      - Hidden layers: as per the layers configuration (units, activation function)
      - Output layer: match the output_shape, typically with a softmax or sigmoid activation
  2. Initialize weights and biases for each layer:
    For each layer $l$ in the network:
      a. Initialize weights $W[l]$ with small random values
      b. Initialize biases $b[l]$ with zeros
  3. For each epoch in range(epochs):
    a. Shuffle the training data
    b. For each batch in the training data:
      i. Extract batch data $X_{\text{batch}}$ and $y_{\text{batch}}$
      ii. Forward propagation:
        - For each layer $l$:
          $z[l] = W[l] \cdot a[l-1] + b[l] \quad (a[0] = X_{\text{batch}})$
          $a[l] = \text{activation\_function}(z[l])$
      iii. Compute loss:
        $\text{loss} = \text{loss\_function}(y_{\text{batch}}, a[\text{output\_layer}])$
      iv. Backward propagation:
        - Initialize gradients for the output layer
        - For each layer $l$ from output to input:
          Compute gradient of loss with respect to $a[l]$, $W[l]$, and $b[l]$
          Update weights and biases:
            $W[l] = W[l] - \text{learning\_rate} \cdot dW[l]$
            $b[l] = b[l] - \text{learning\_rate} \cdot db[l]$
  4. Return the trained model with updated weights and biases
  End
```

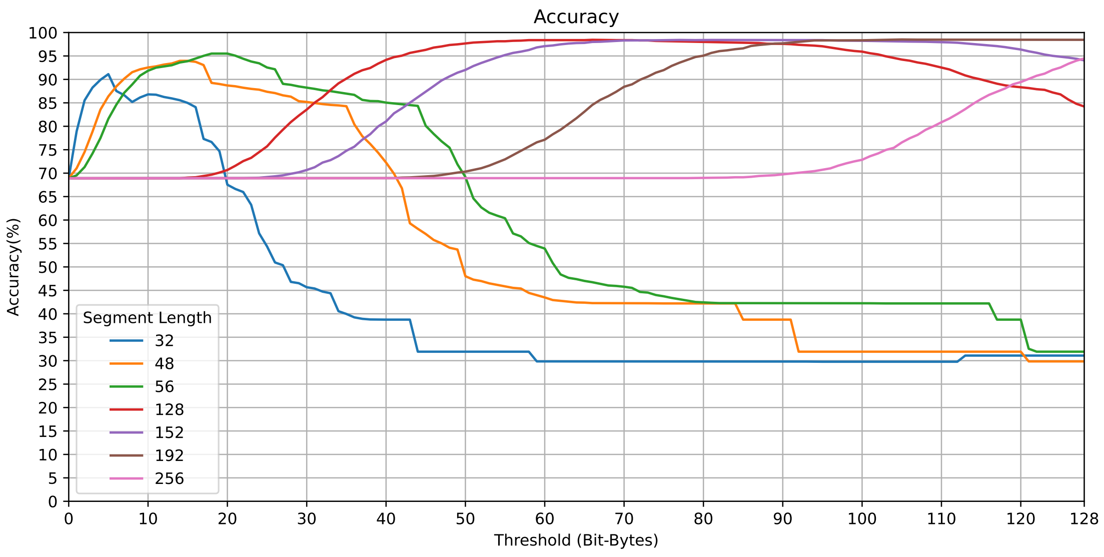


Figure A.2: Accuracy of the rewritten DAA on the OldNapierOne Dataset

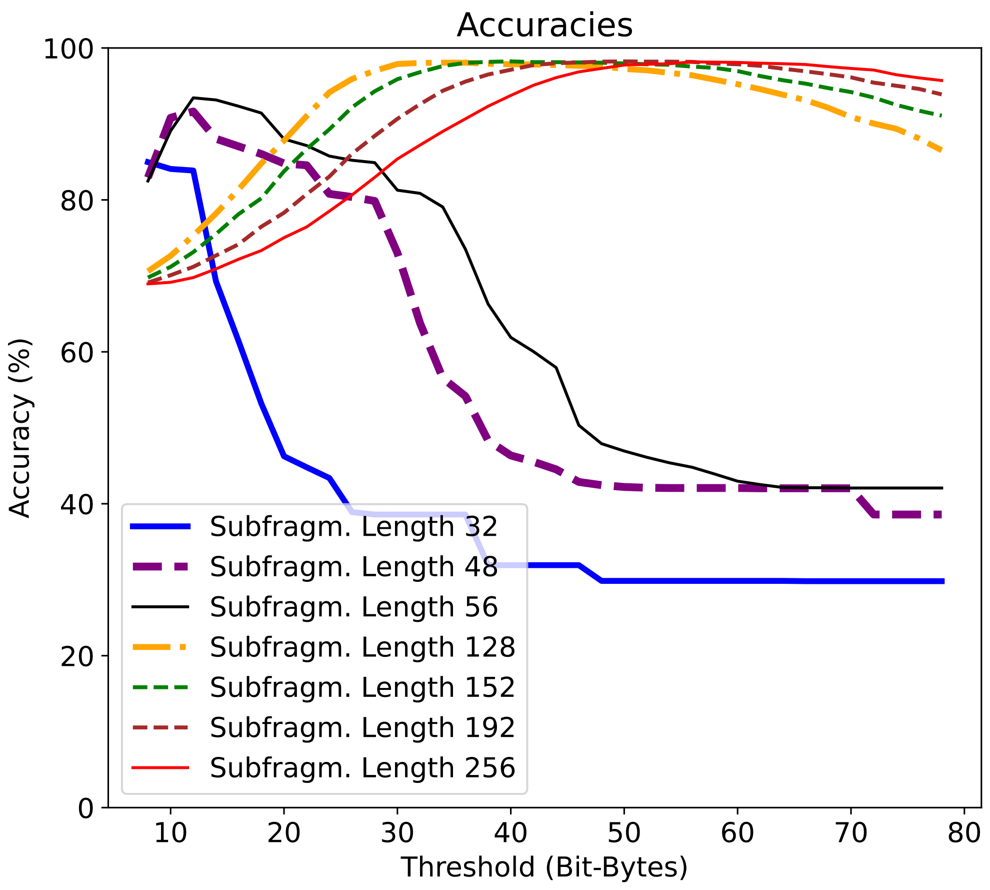


Figure A.3: Accuracy of the DAA by Venturini et al. on the OldNapierOne Dataset

A.3 DAA

A.4 Mitigations

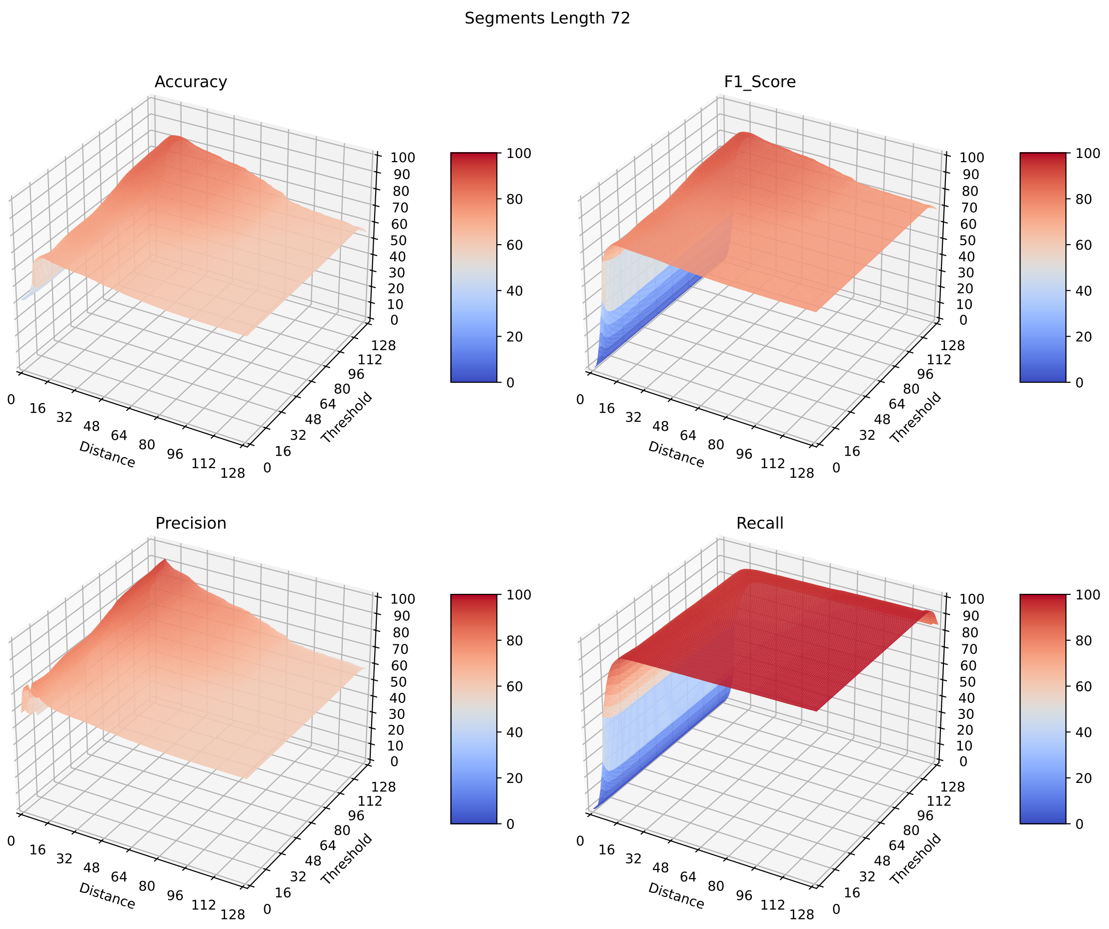


Figure A.4: Best 2F mitigation metrics values for all distance-threshold pairs on the Attack dataset

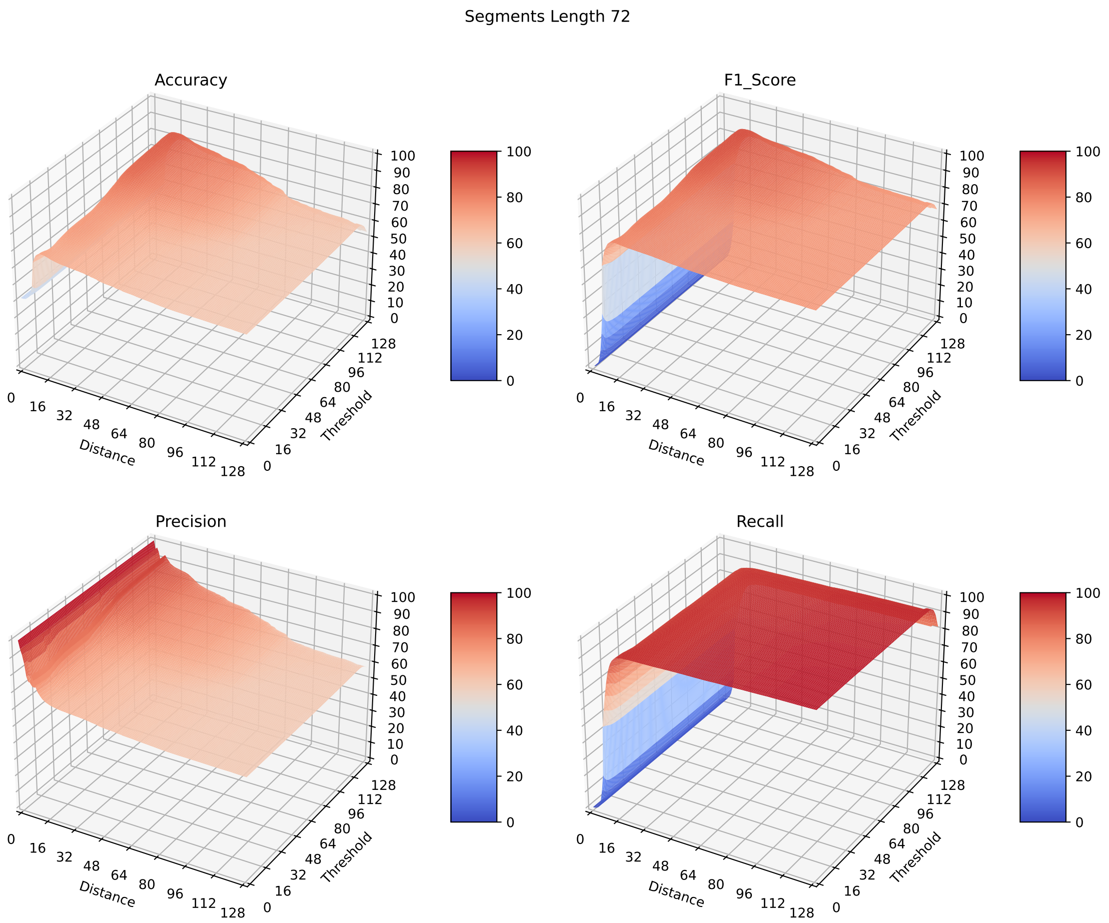


Figure A.5: Best 3F mitigation metrics values for all distance-threshold pairs on the Attack dataset

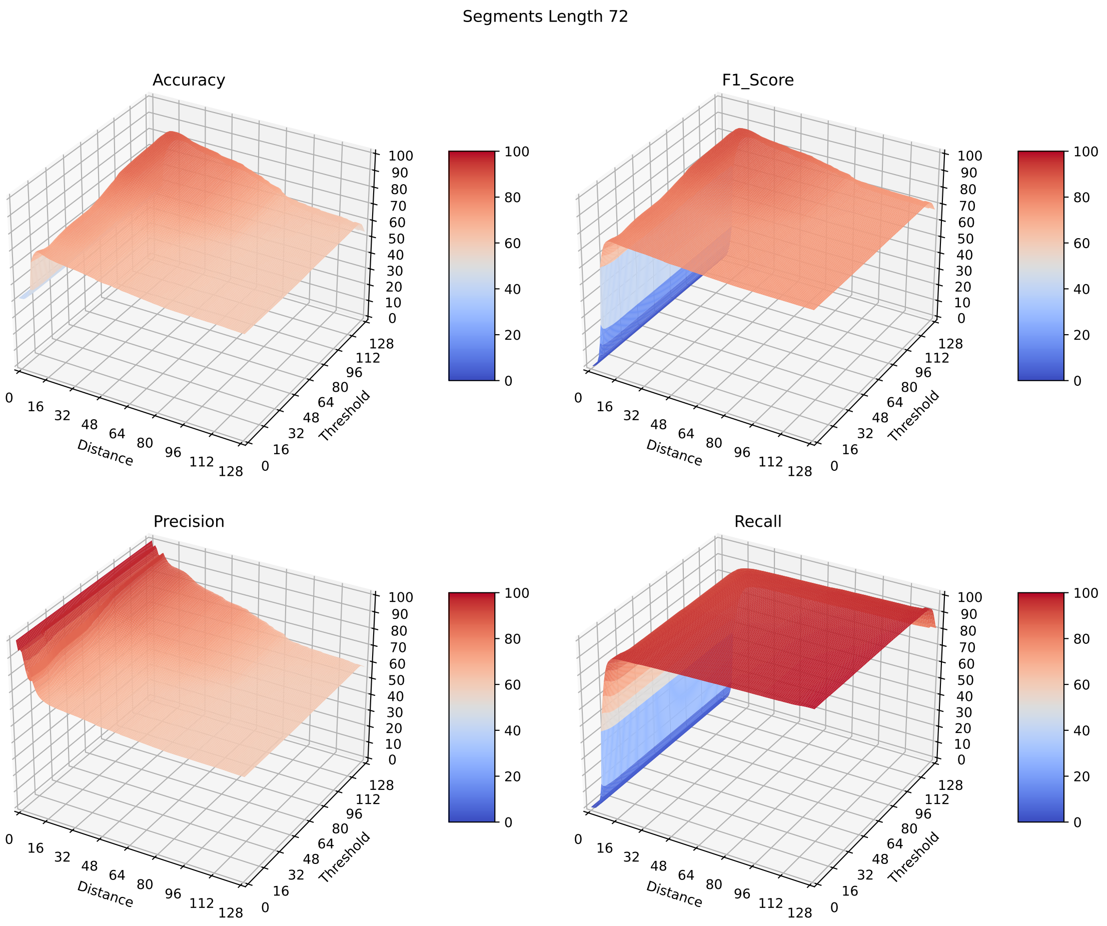


Figure A.6: Best 4F mitigation metrics values for all distance-threshold pairs on the Attack dataset

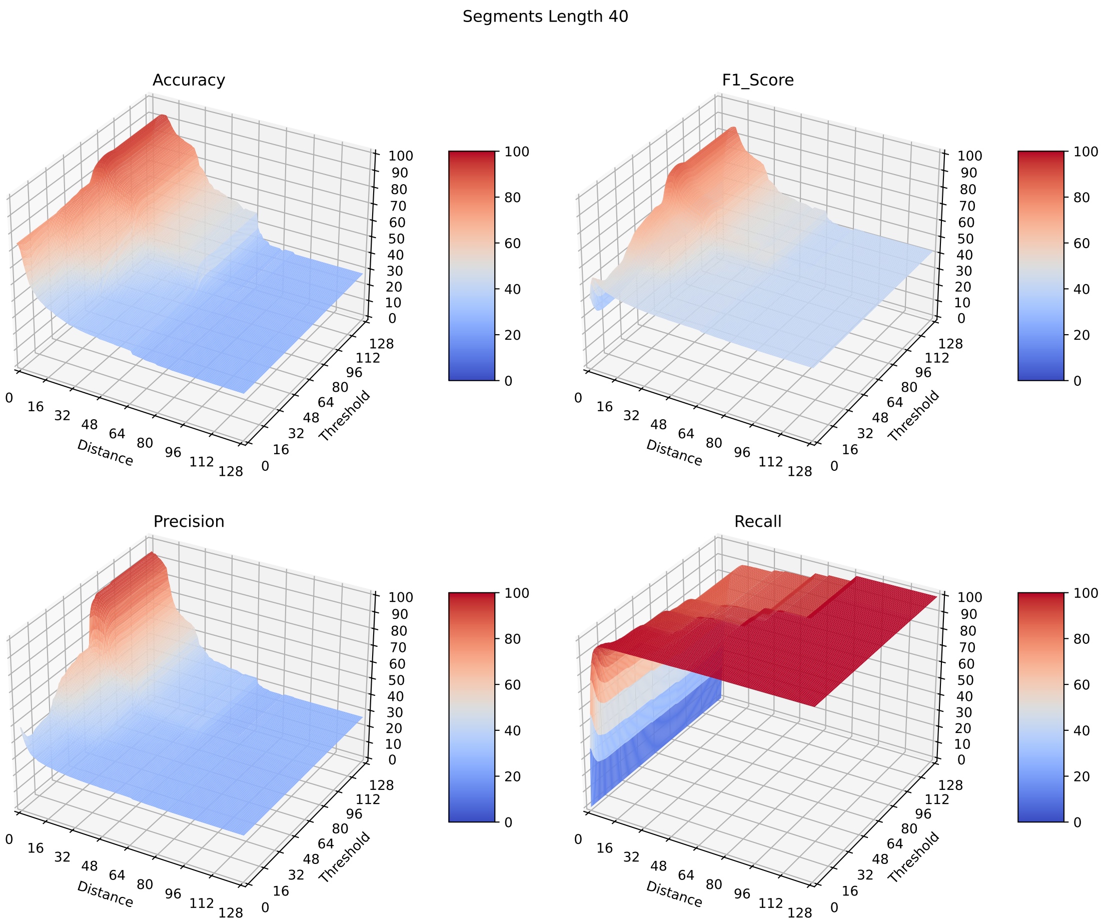


Figure A.7: Best 2F mitigation metrics values for all distance-threshold pairs on the NapierOne dataset

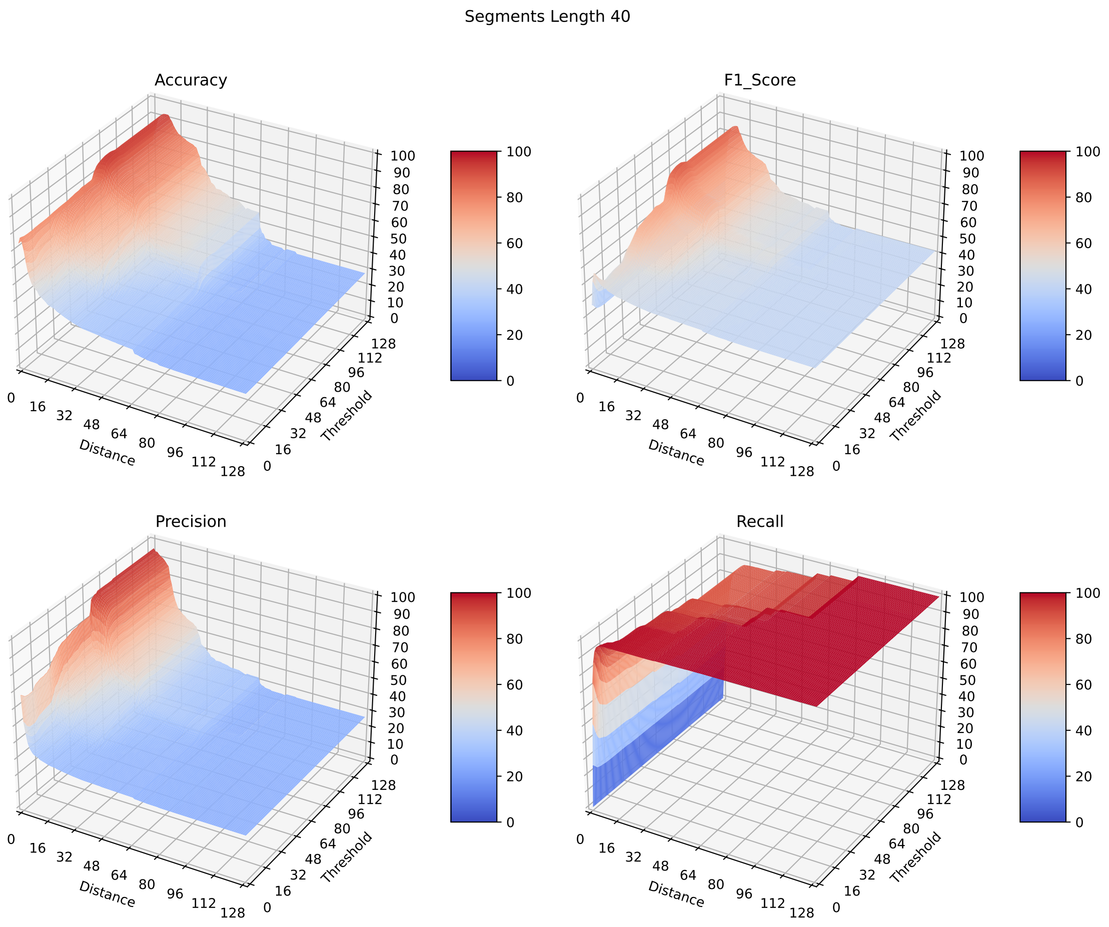


Figure A.8: Best 4F mitigation metrics values for all distance-threshold pairs on the NapierOne dataset

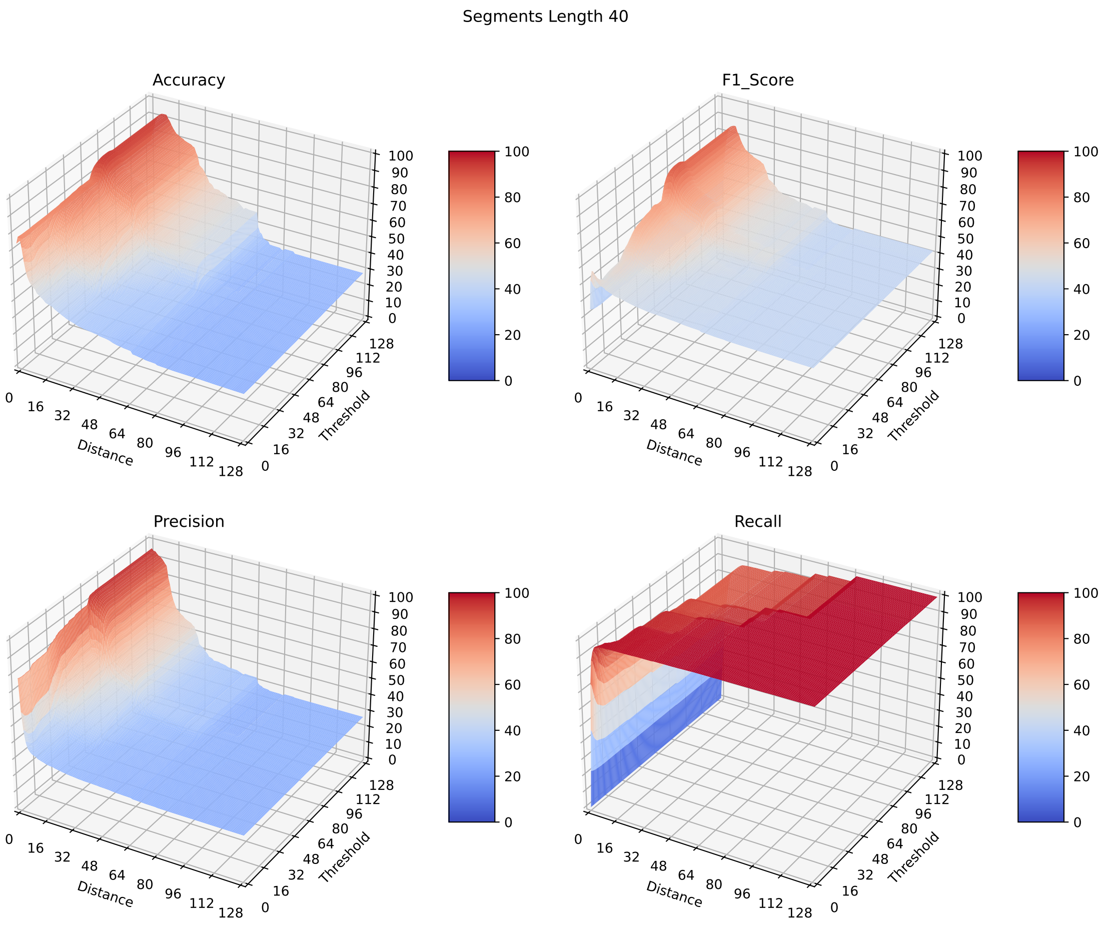


Figure A.9: Best 4F mitigation metrics values for all distance-threshold pairs on the NapierOne dataset

A.5 Machine Learning

A.5.1 Mean Runs

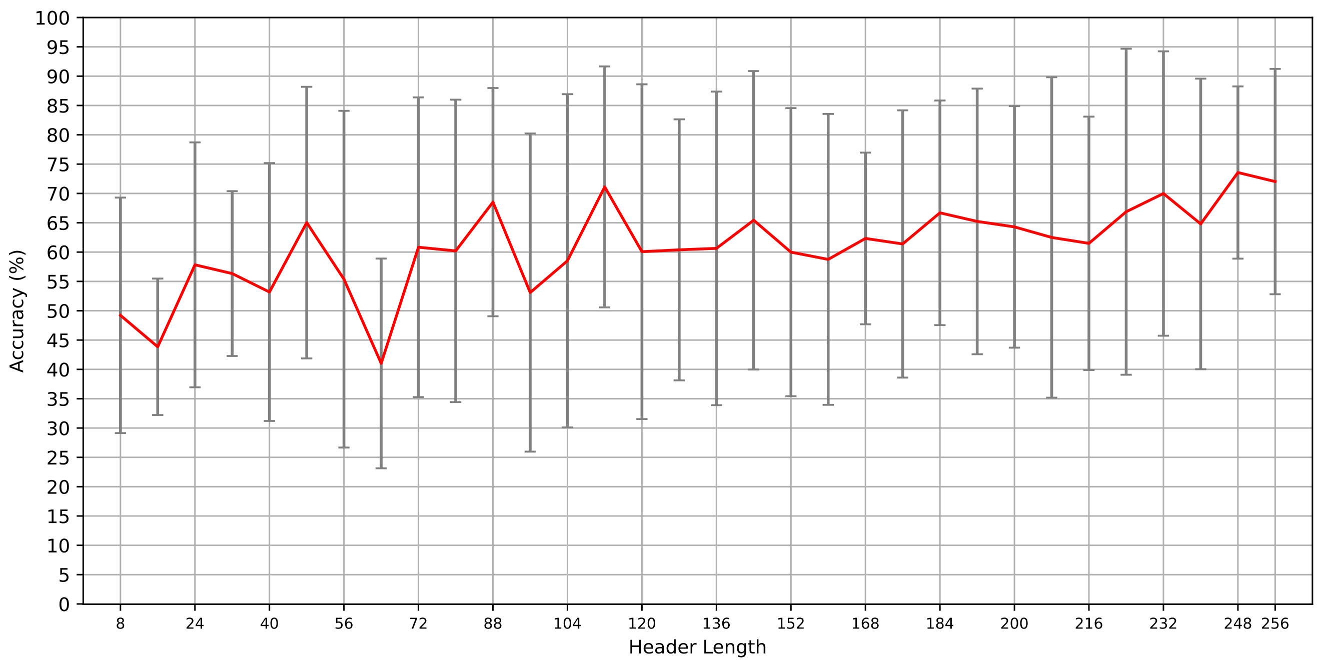


Figure A.10: Accuracy obtained by Entropy of header as feature for Neural Network on the NapierOne Dataset. Each lying point is the average results obtained from ten runs. The vertical lines represent the standard deviation

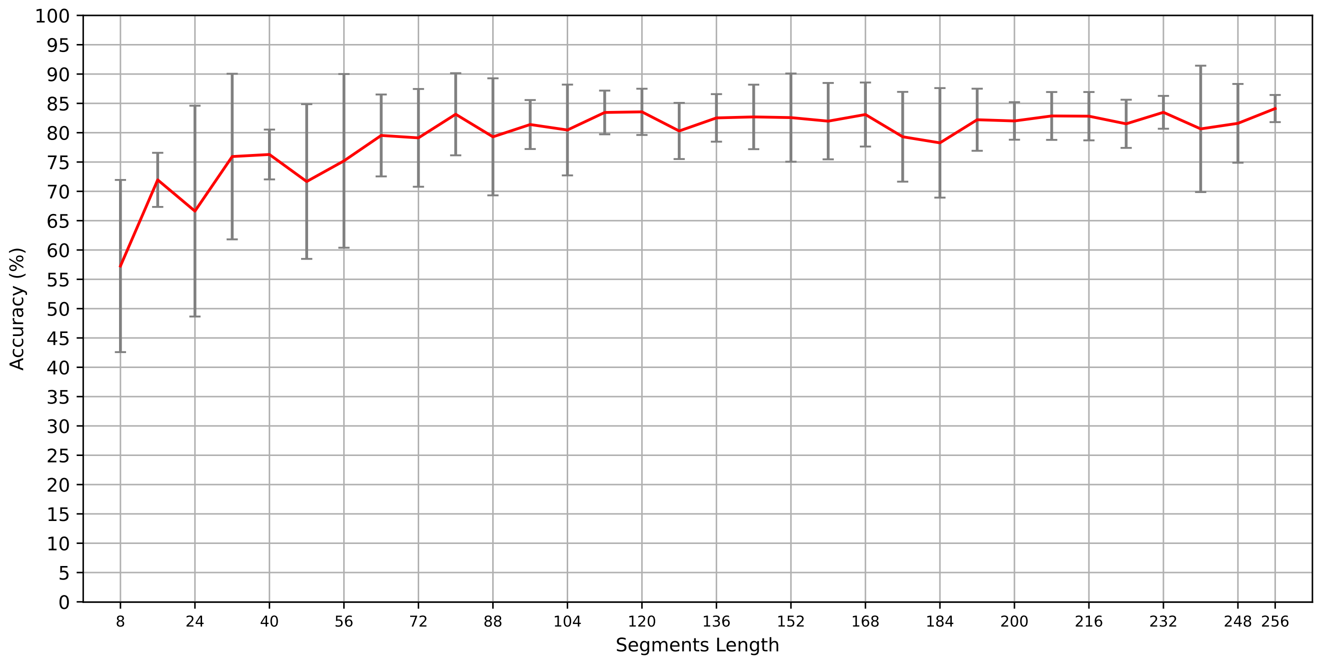


Figure A.11: Accuracy obtained by Entropy of 2F as feature for Neural Network on the NapierOne Dataset. Each lying point is the average results obtained from ten runs. The vertical lines represent the standard deviation

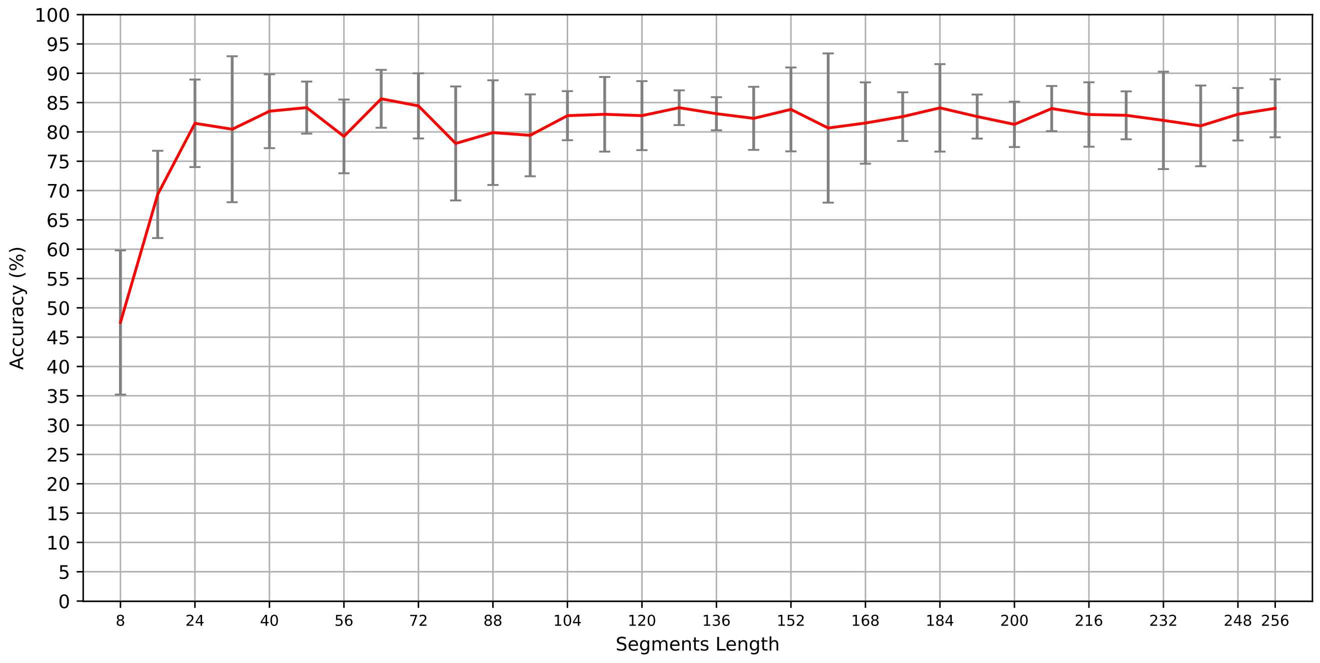


Figure A.12: Accuracy obtained by Entropy of 3F as feature for Neural Network on the NapierOne Dataset. Each lying point is the average results obtained from ten runs. The vertical lines represent the standard deviation

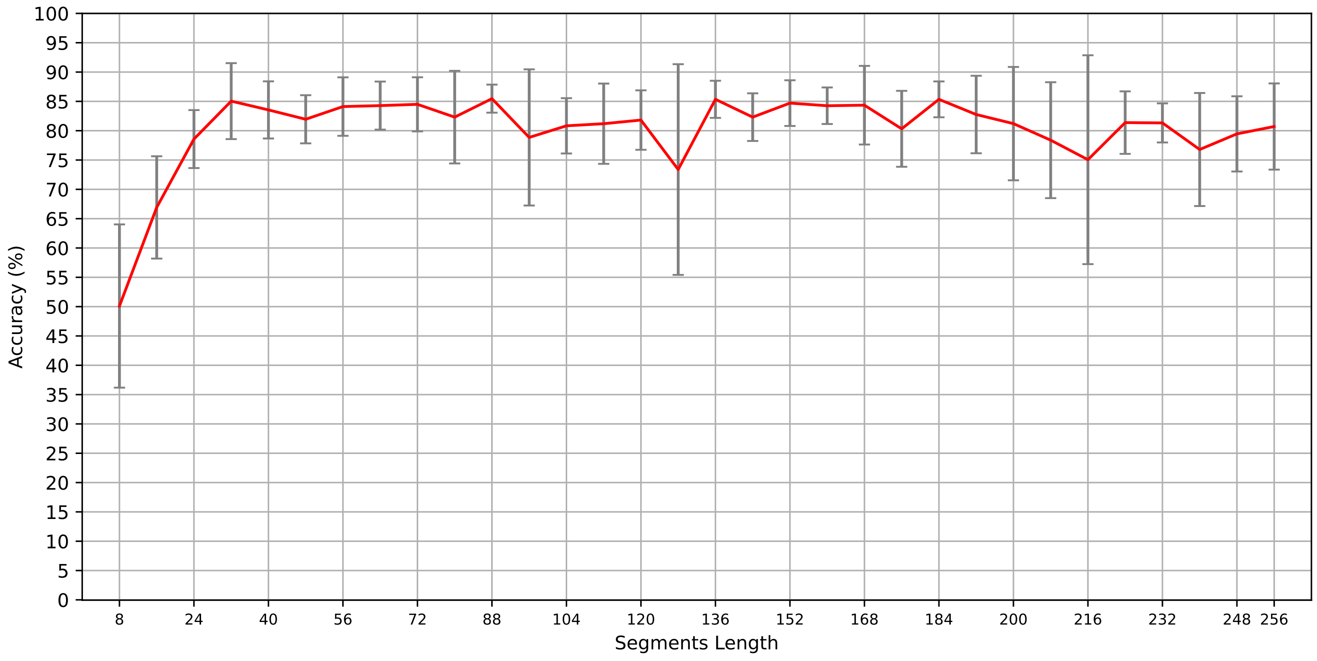


Figure A.13: Accuracy obtained by Entropy of 4F as feature for Neural Network on the NapierOne Dataset. Each lying point is the average results obtained from ten runs. The vertical lines represent the standard deviation

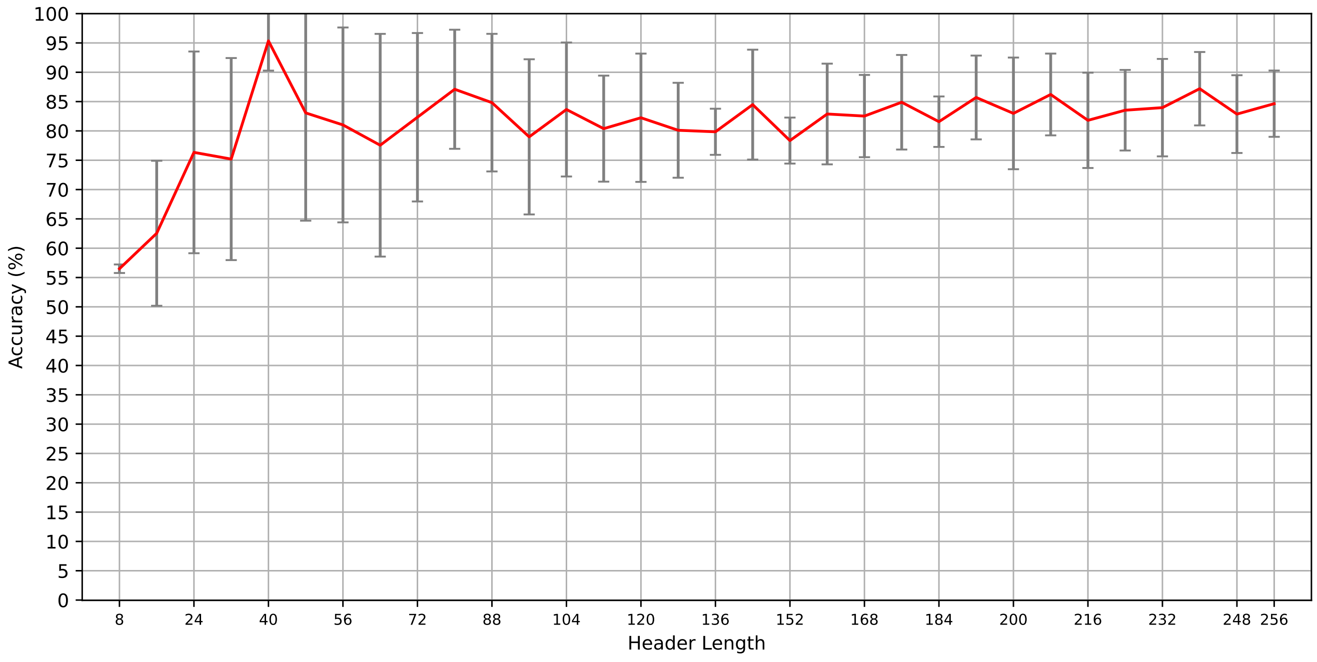


Figure A.14: Accuracy obtained by Entropy of header as feature for Neural Network on the Attack Dataset. Each lying point is the average results obtained from ten runs. The vertical lines represent the standard deviation

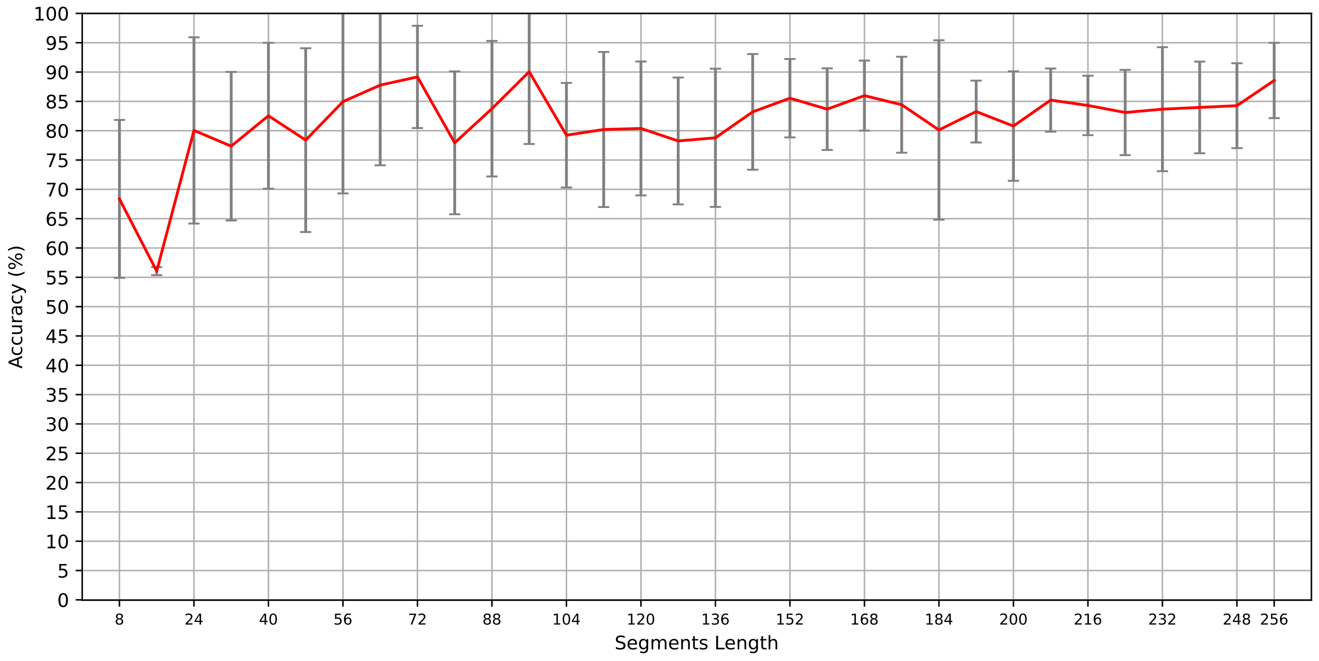


Figure A.15: Accuracy obtained by Entropy of 2F as feature for Neural Network on the Attack Dataset. Each lying point is the average results obtained from ten runs. The vertical lines represent the standard deviation

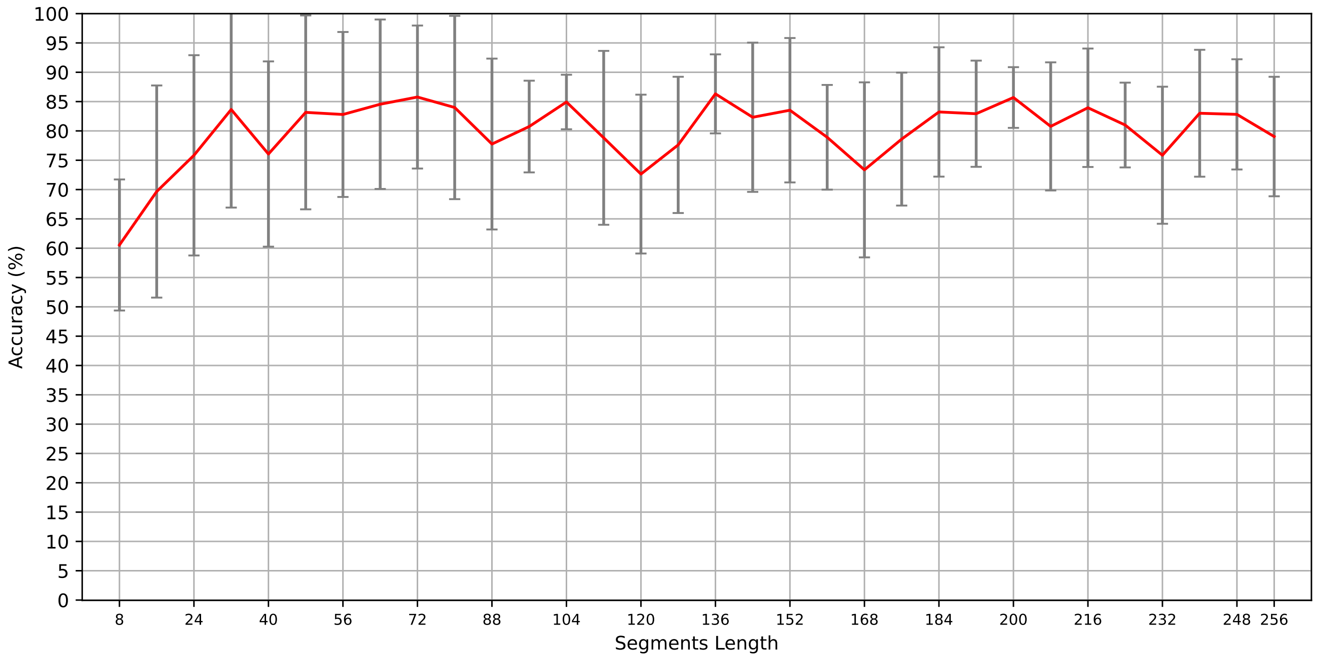


Figure A.16: Accuracy obtained by Entropy of 3F as feature for Neural Network on the Attack Dataset. Each lying point is the average results obtained from ten runs. The vertical lines represent the standard deviation

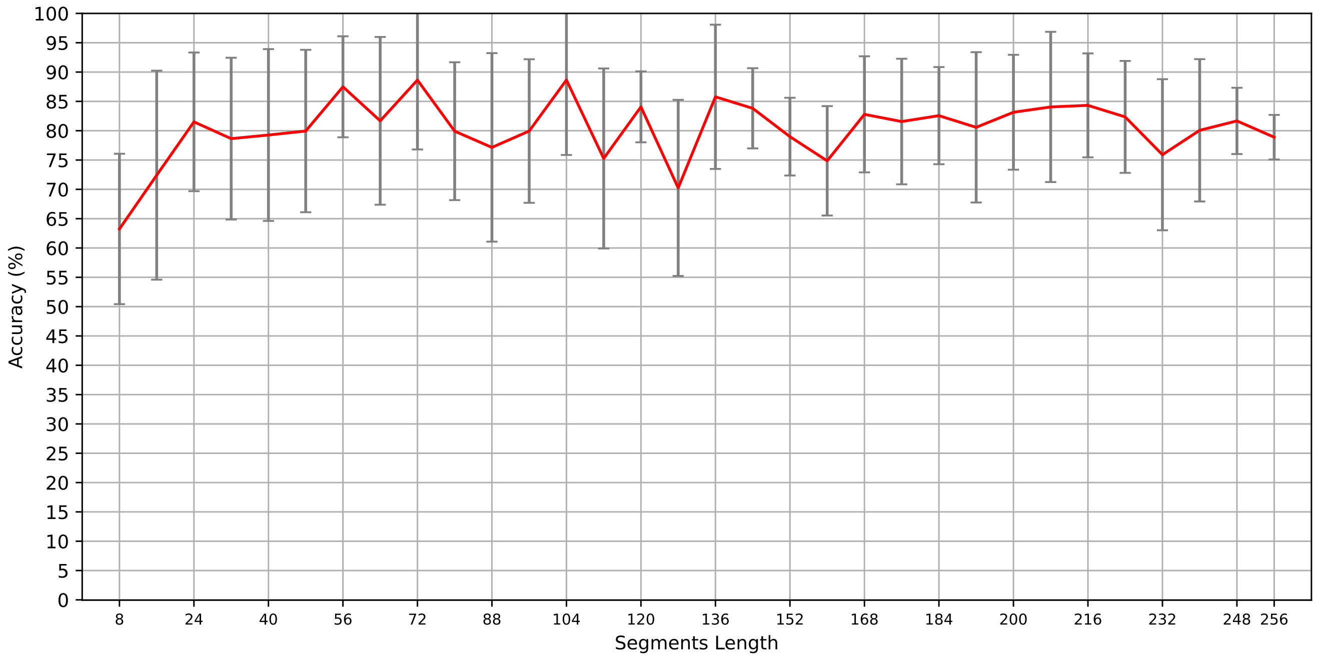


Figure A.17: Accuracy obtained by Entropy of 4F as feature for Neural Network on the Attack Dataset. Each lying point is the average results obtained from ten runs. The vertical lines represent the standard deviation

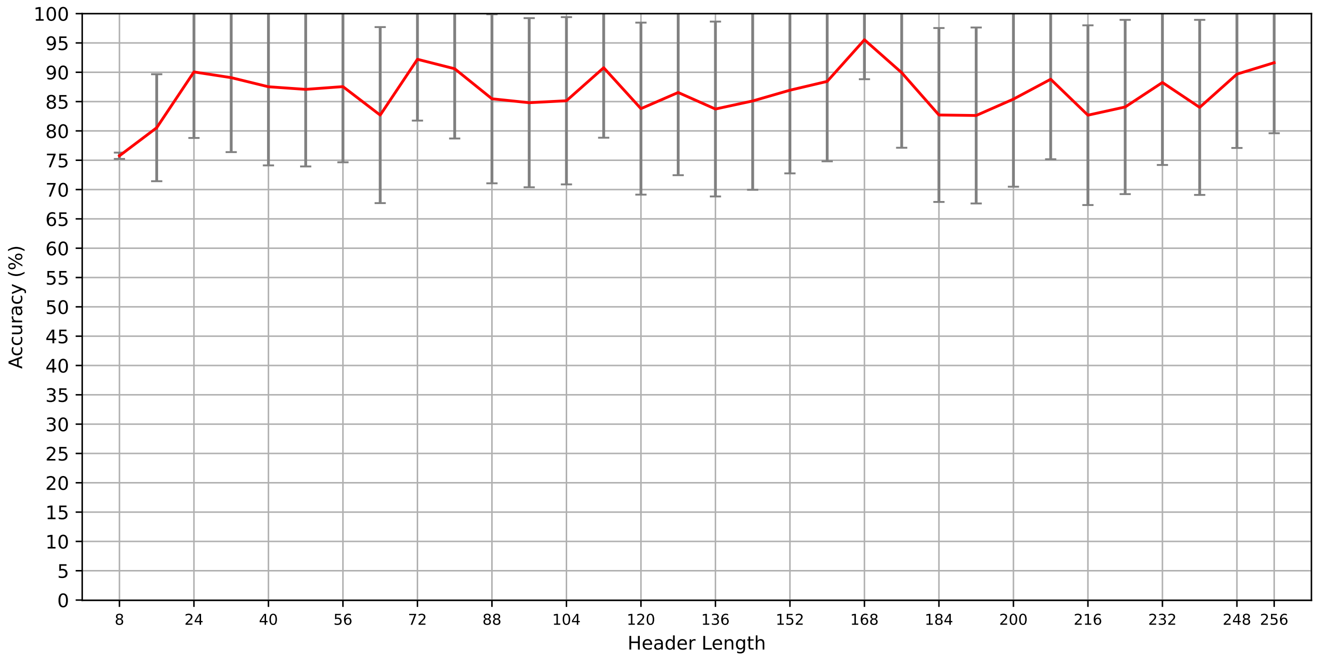


Figure A.18: Accuracy obtained by Entropy of header as feature for Random Forest on the NapierOne Dataset. Each lying point is the average results obtained from ten runs. The vertical lines represent the standard deviation

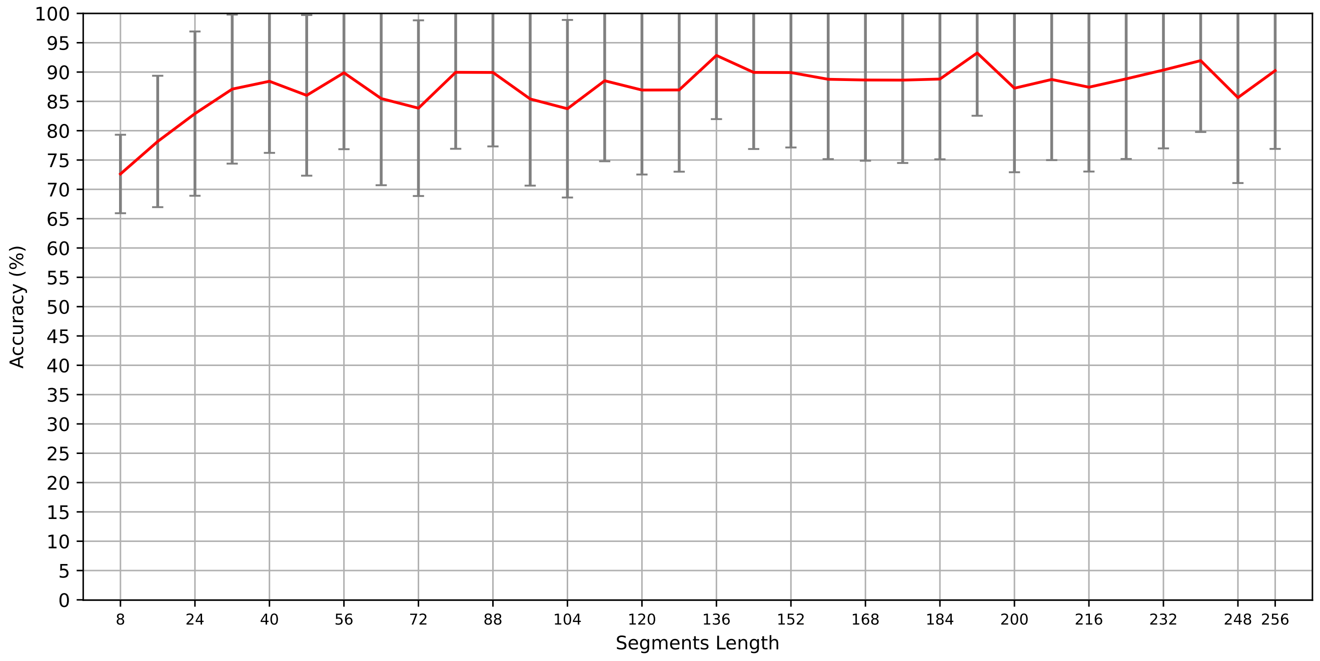


Figure A.19: Accuracy obtained by Entropy of 2F as feature for Random Forest on the NapierOne Dataset. Each lying point is the average results obtained from ten runs. The vertical lines represent the standard deviation

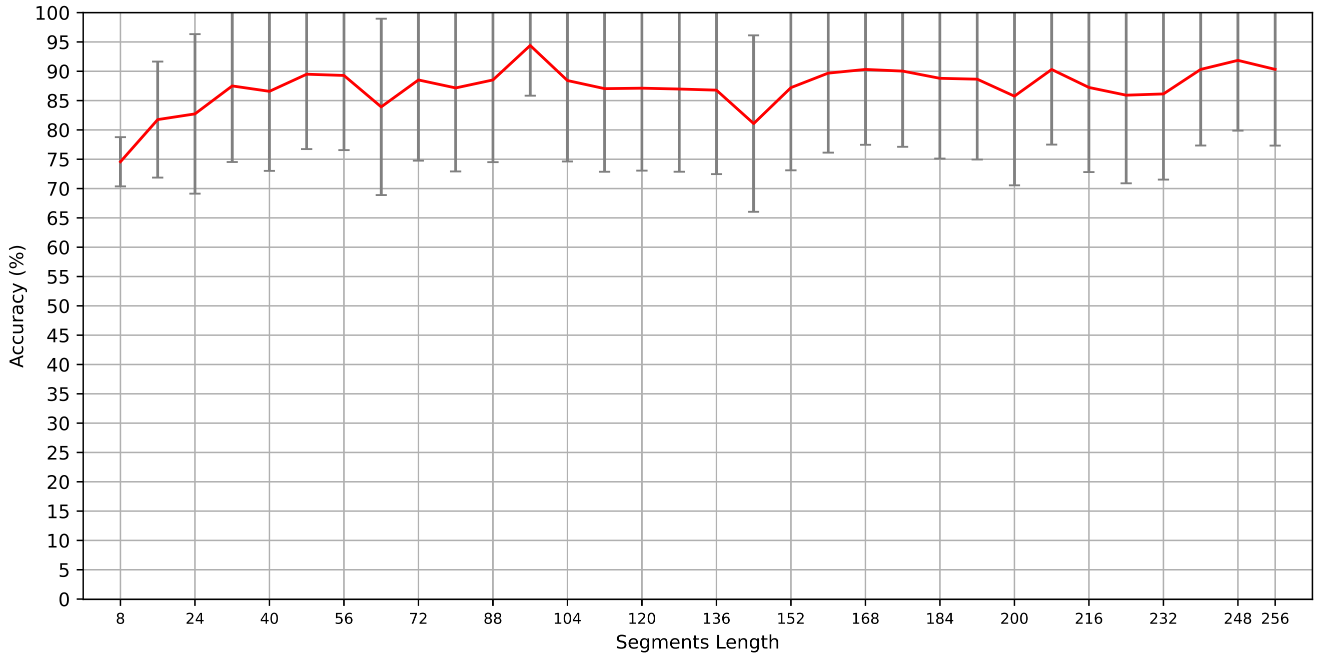


Figure A.20: Accuracy obtained by Entropy of 3F as feature for Random Forest on the NapierOne Dataset. Each lying point is the average results obtained from ten runs. The vertical lines represent the standard deviation

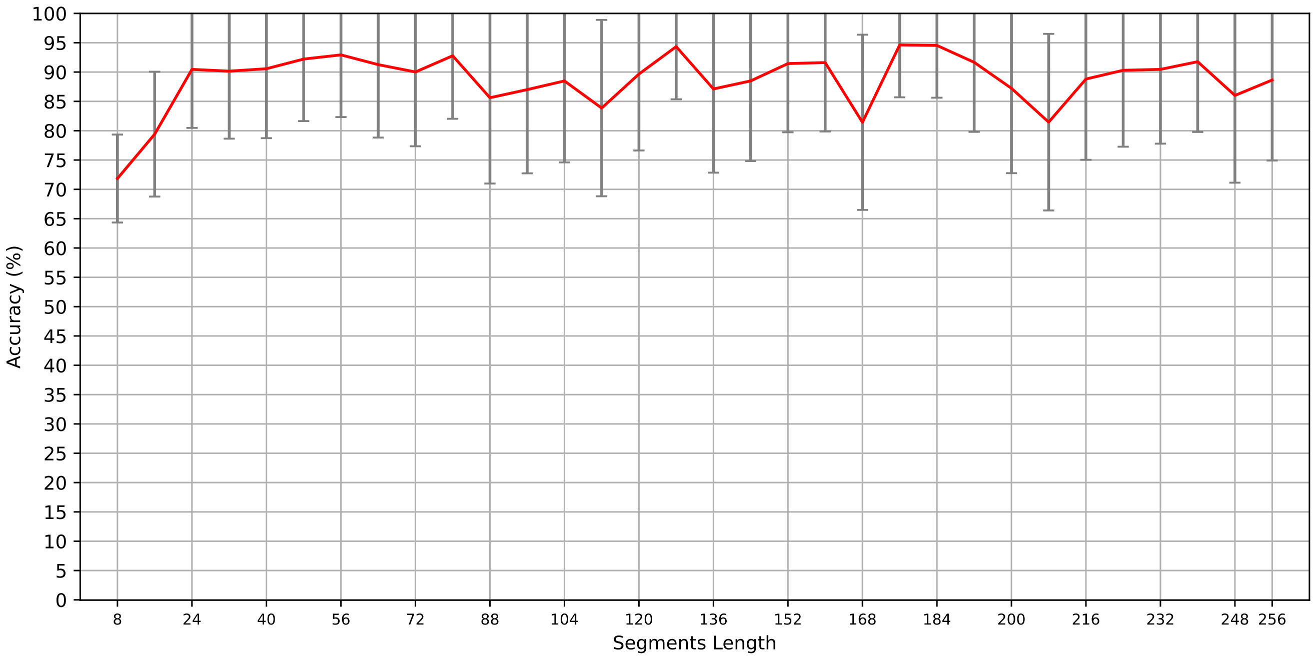


Figure A.21: Accuracy obtained by Entropy of 4F as feature for Random Forest on the NapierOne Dataset. Each lying point is the average results obtained from ten runs. The vertical lines represent the standard deviation

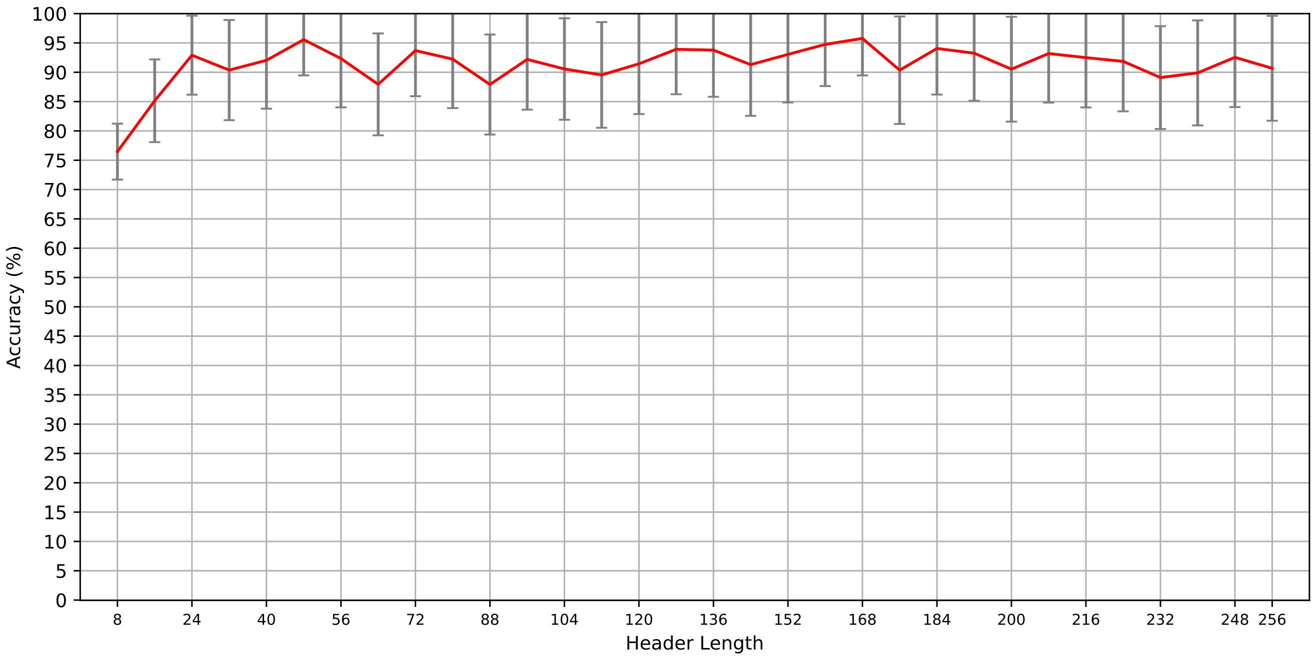


Figure A.22: Accuracy obtained by Entropy of header as feature for Random Forest on the Attack Dataset. Each lying point is the average results obtained from ten runs. The vertical lines represent the standard deviation

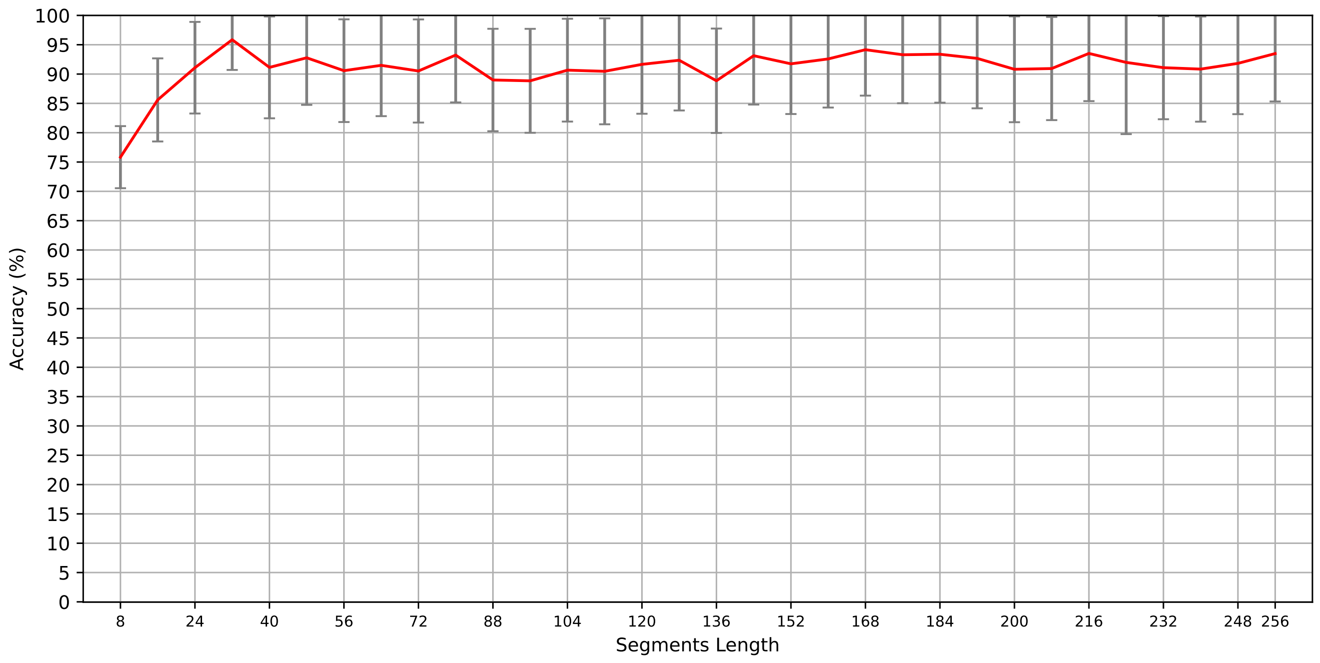


Figure A.23: Accuracy obtained by Entropy of 2F as feature for Random Forest on the Attack Dataset. Each lying point is the average results obtained from ten runs. The vertical lines represent the standard deviation

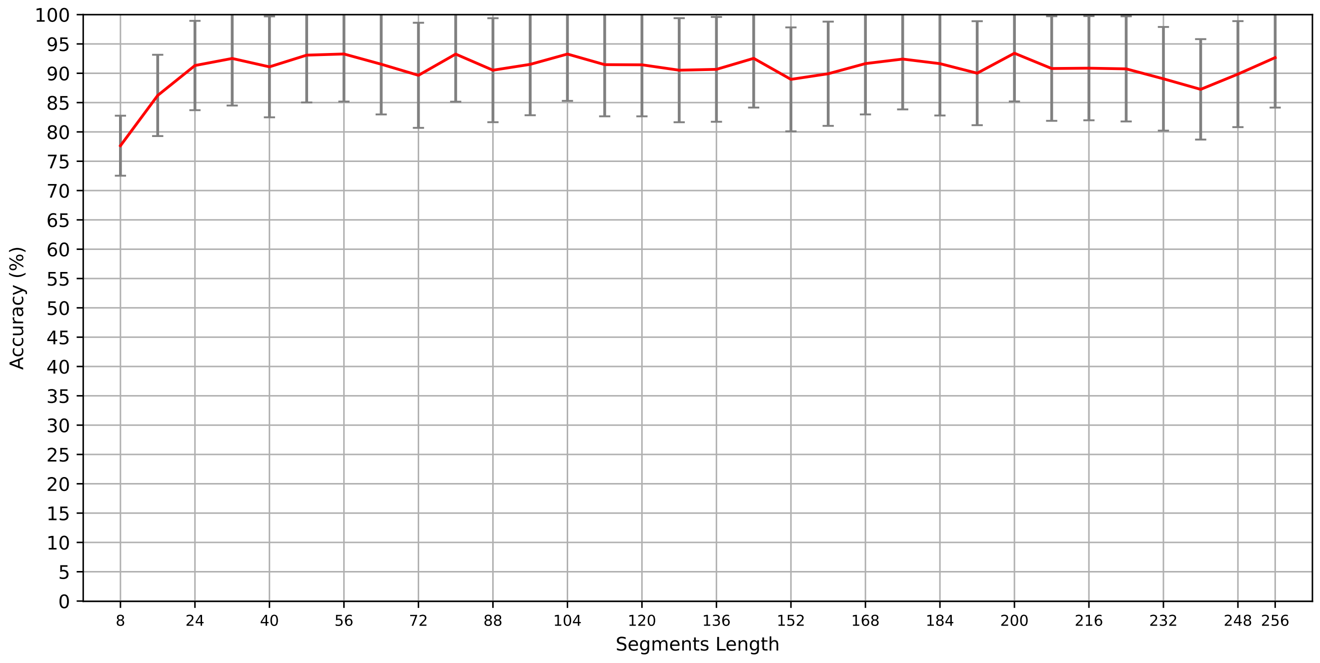


Figure A.24: Accuracy obtained by Entropy of 3F as feature for Random Forest on the Attack Dataset. Each lying point is the average results obtained from ten runs. The vertical lines represent the standard deviation

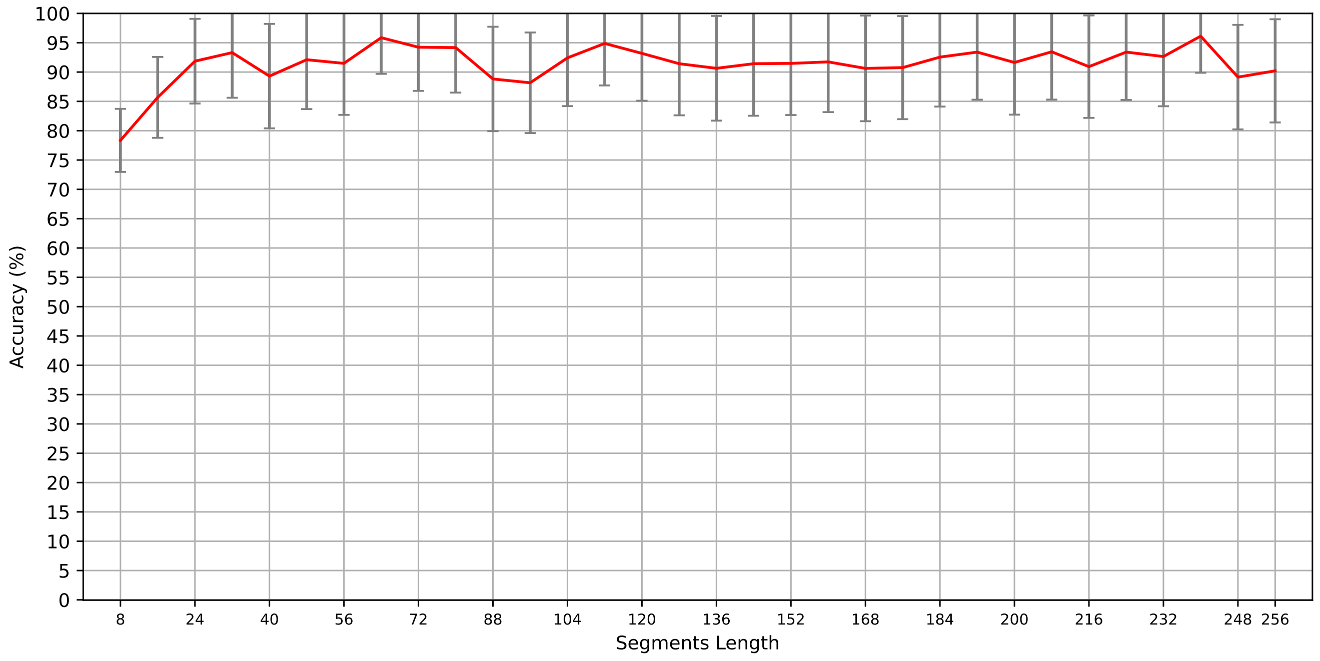


Figure A.25: Accuracy obtained by Entropy of 4F as feature for Random Forest on the Attack Dataset. Each lying point is the average results obtained from ten runs. The vertical lines represent the standard deviation

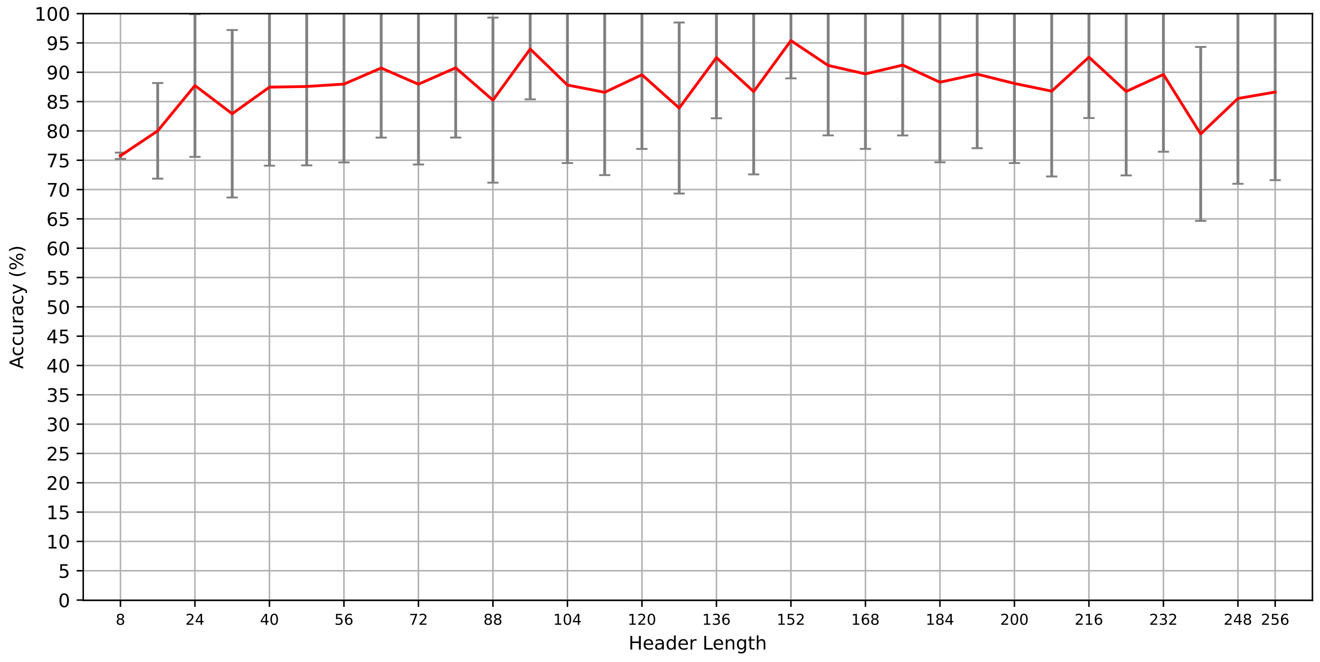


Figure A.26: Accuracy obtained by Entropy of header as feature for SVC on the NapierOne Dataset. Each lying point is the average results obtained from ten runs. The vertical lines represent the standard deviation

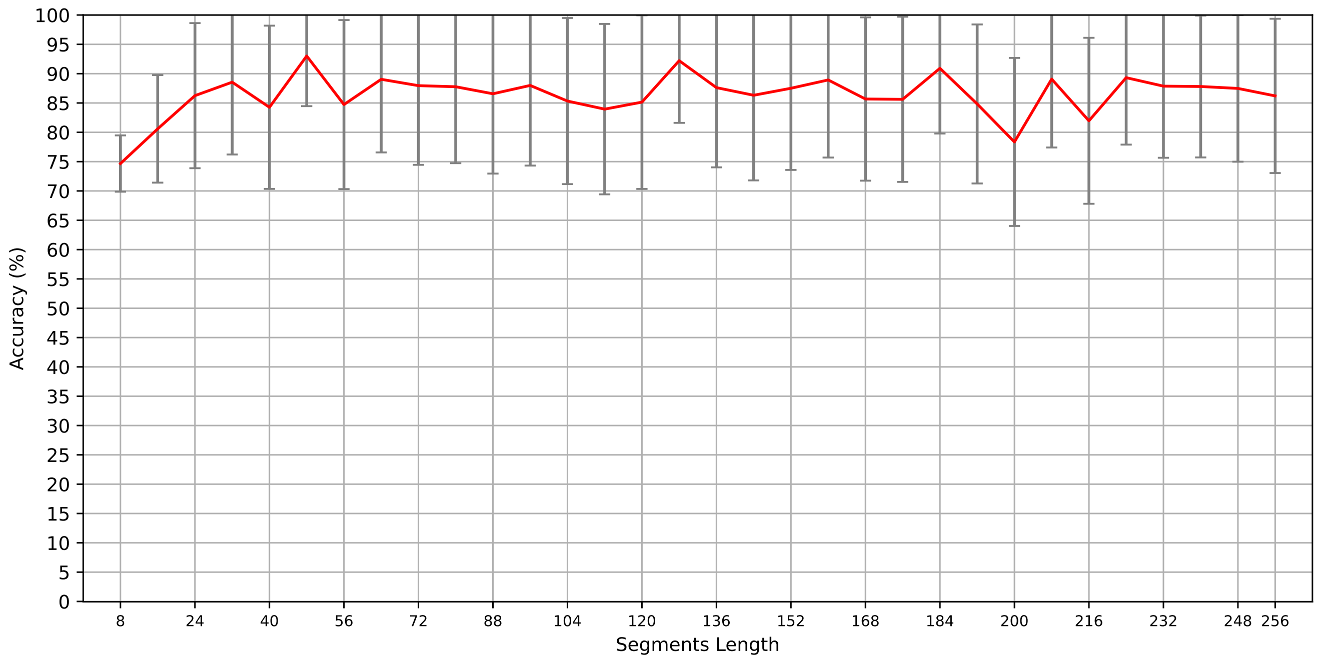


Figure A.27: Accuracy obtained by Entropy of 2F as feature for SVC on the NapierOne Dataset. Each lying point is the average results obtained from ten runs. The vertical lines represent the standard deviation

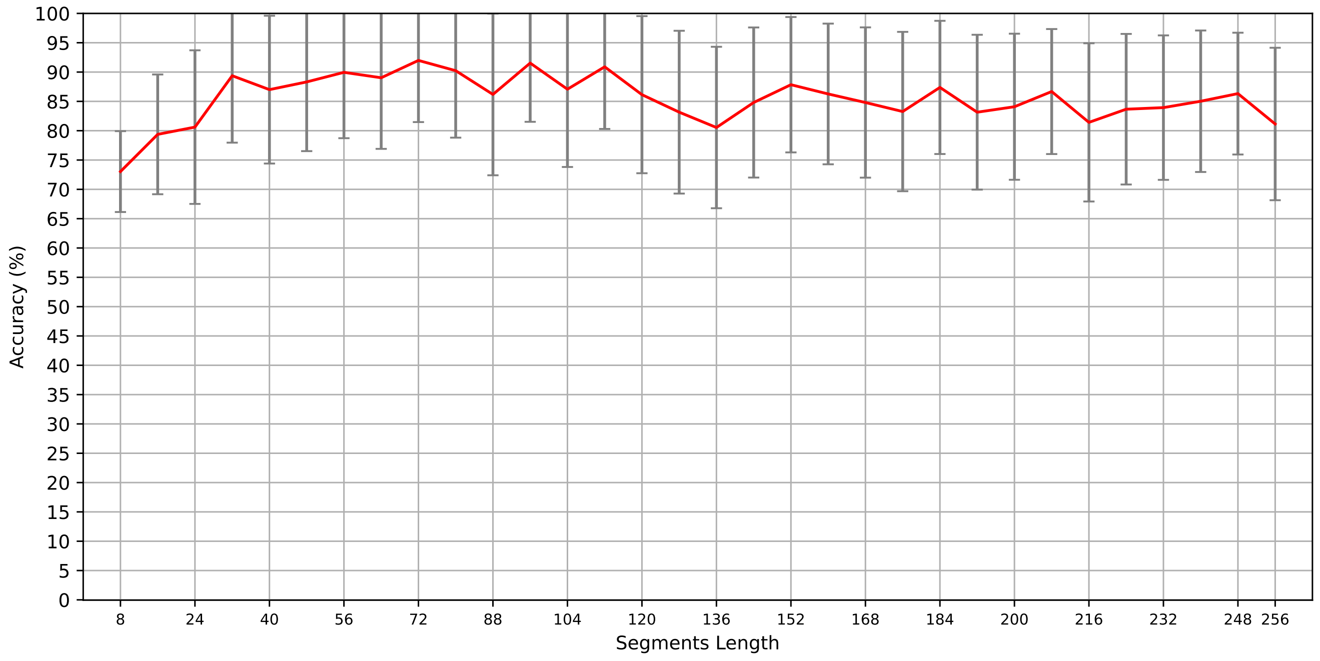


Figure A.28: Accuracy obtained by Entropy of 3F as feature for SVC on the NapierOne Dataset. Each lying point is the average results obtained from ten runs. The vertical lines represent the standard deviation

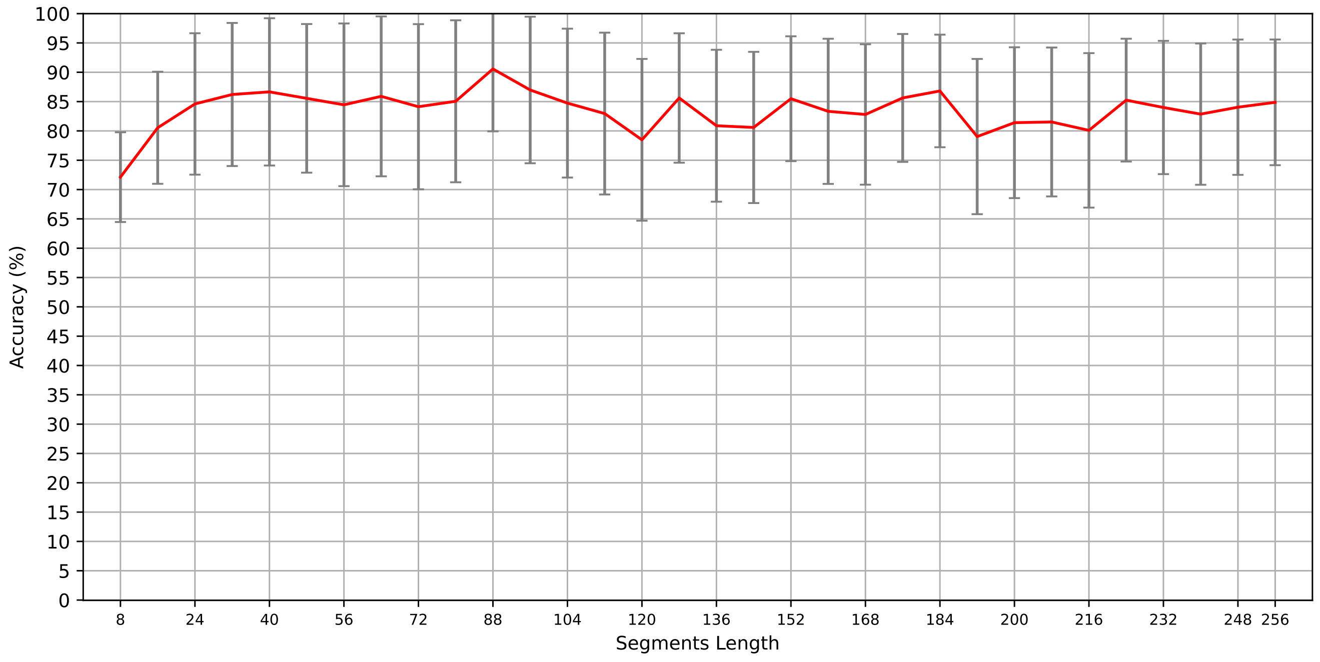


Figure A.29: Accuracy obtained by Entropy of 4F as feature for SVC on the NapierOne Dataset. Each lying point is the average results obtained from ten runs. The vertical lines represent the standard deviation

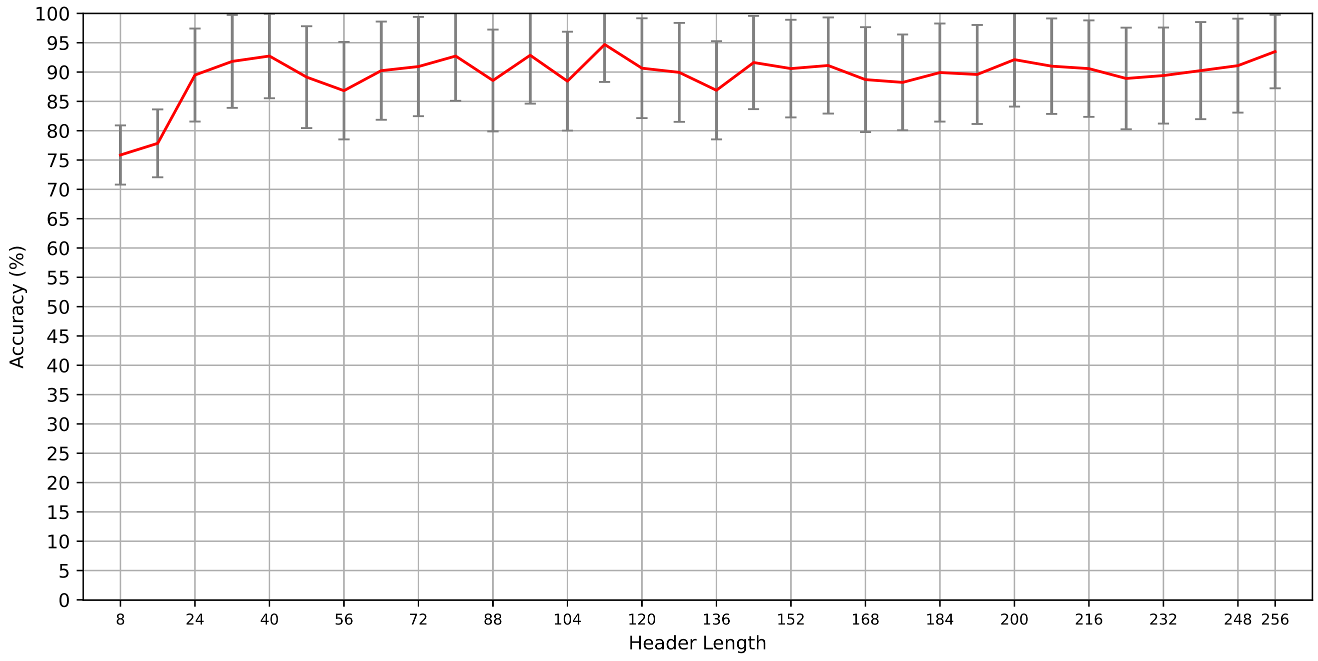


Figure A.30: Accuracy obtained by Entropy of header as feature for SVC on the Attack Dataset. Each lying point is the average results obtained from ten runs. The vertical lines represent the standard deviation

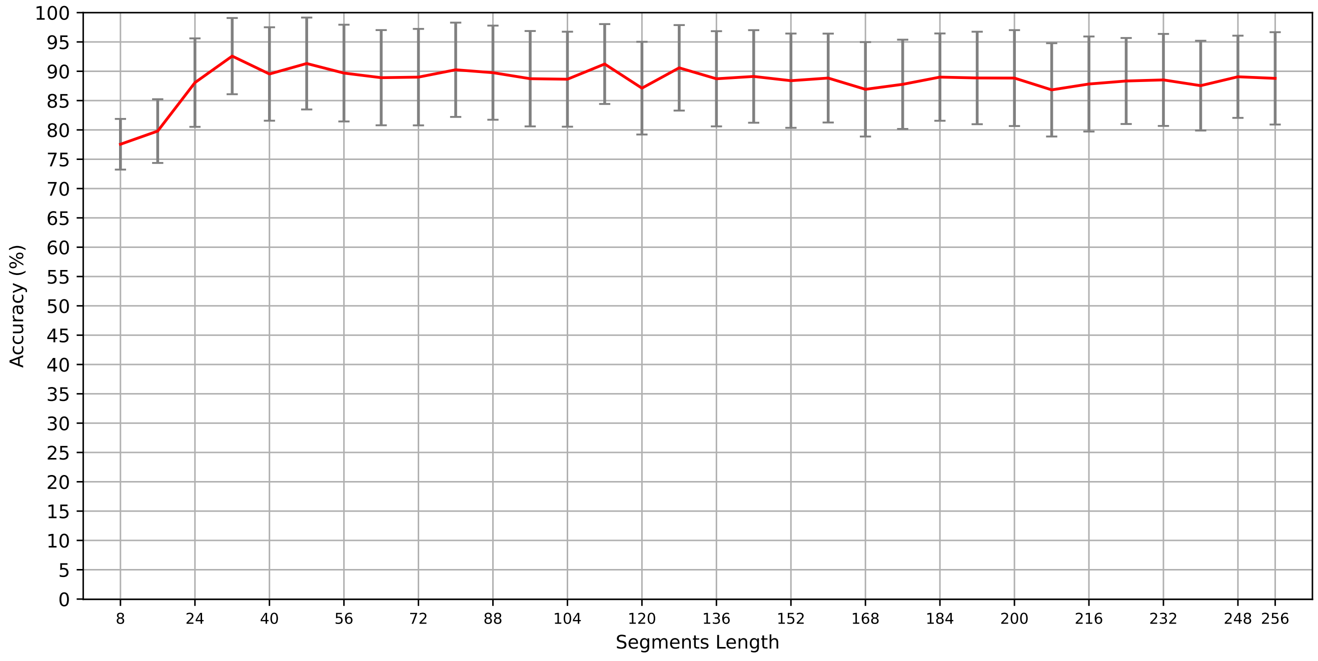


Figure A.31: Accuracy obtained by Entropy of 2F as feature for SVC on the Attack Dataset. Each lying point is the average results obtained from ten runs. The vertical lines represent the standard deviation

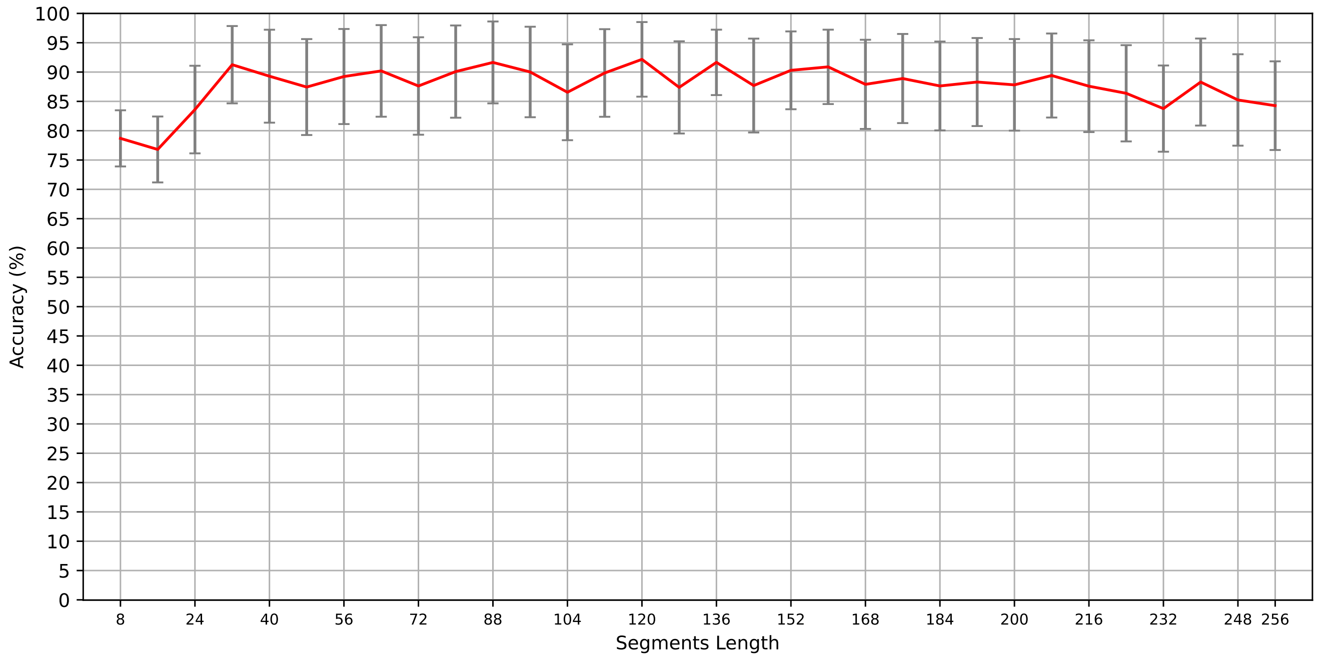


Figure A.32: Accuracy obtained by Entropy of 3F as feature for SVC on the Attack Dataset. Each lying point is the average results obtained from ten runs. The vertical lines represent the standard deviation

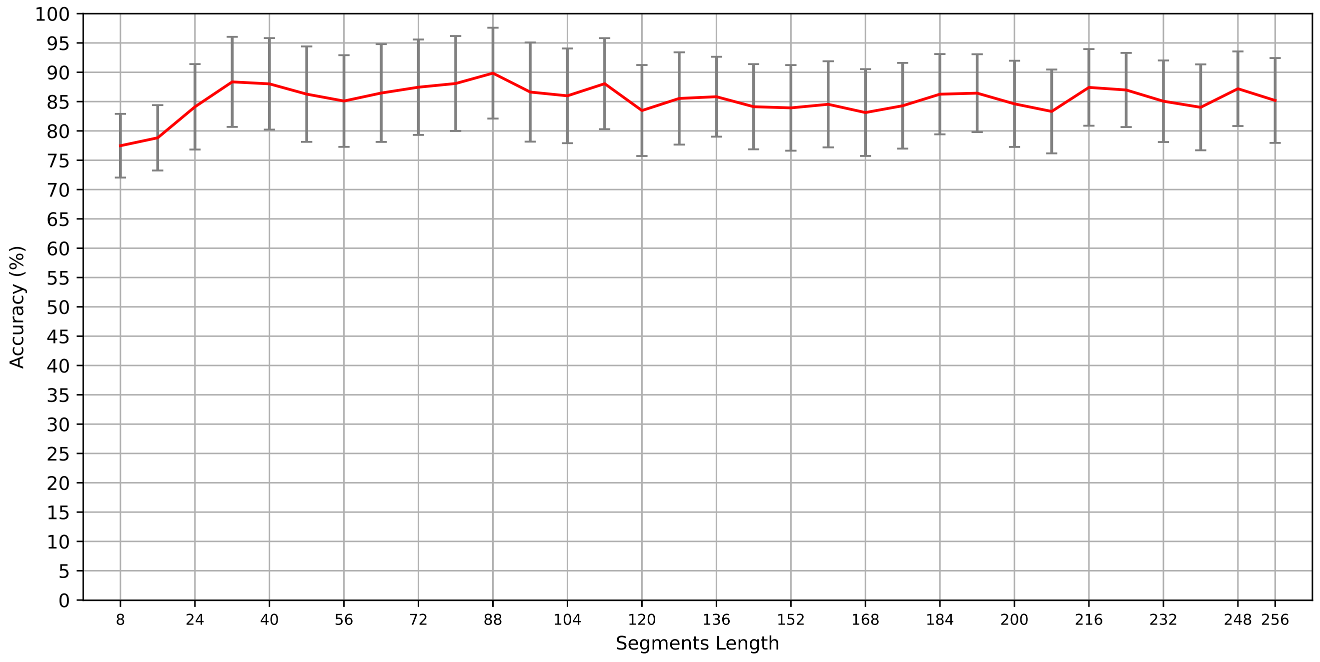


Figure A.33: Accuracy obtained by Entropy of 4F as feature for SVC on the Attack Dataset. Each lying point is the average results obtained from ten runs. The vertical lines represent the standard deviation

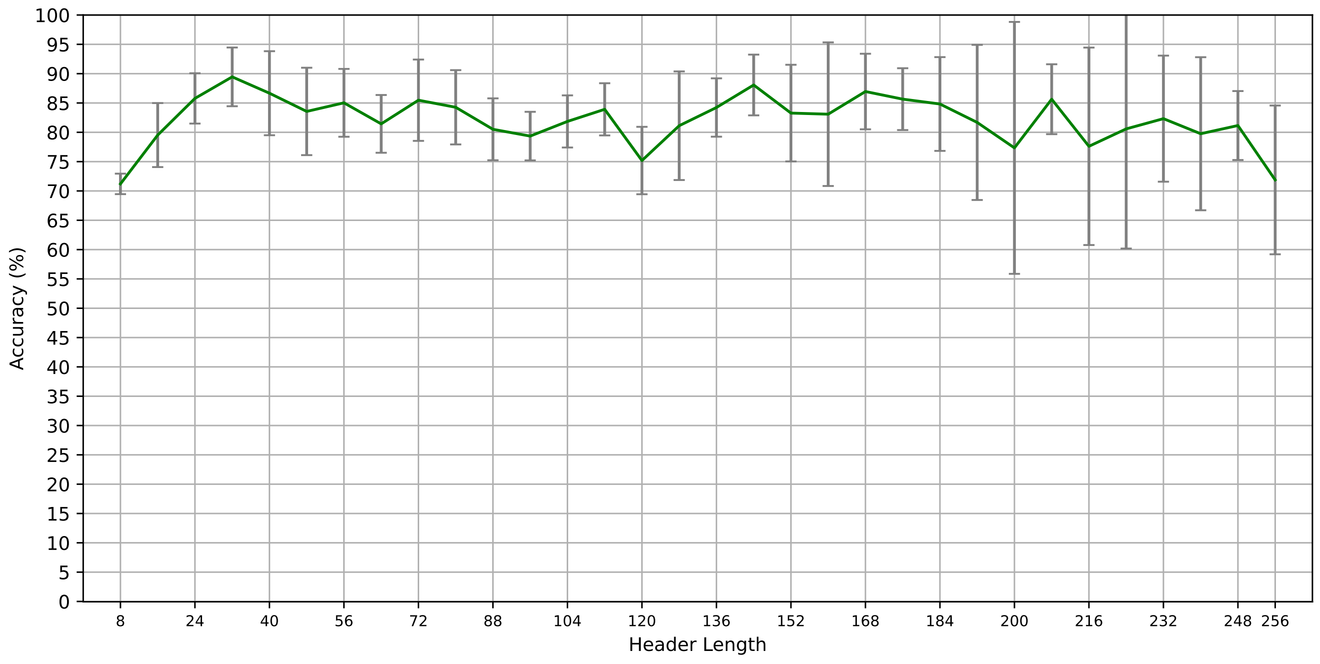


Figure A.34: Accuracy obtained by DA of header as feature for Neural Network on the NapierOne Dataset. Each lying point is the average results obtained from ten runs. The vertical lines represent the standard deviation

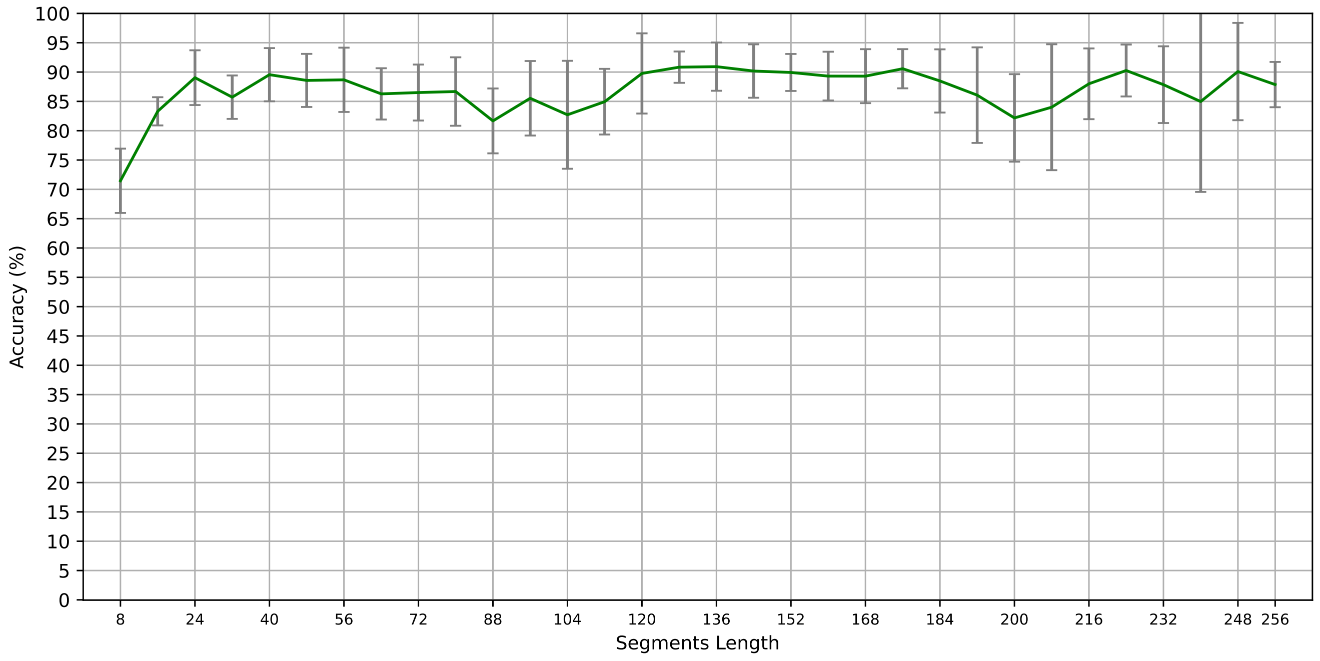


Figure A.35: Accuracy obtained by DA of 2F as feature for Neural Network on the NapierOne Dataset. Each lying point is the average results obtained from ten runs. The vertical lines represent the standard deviation

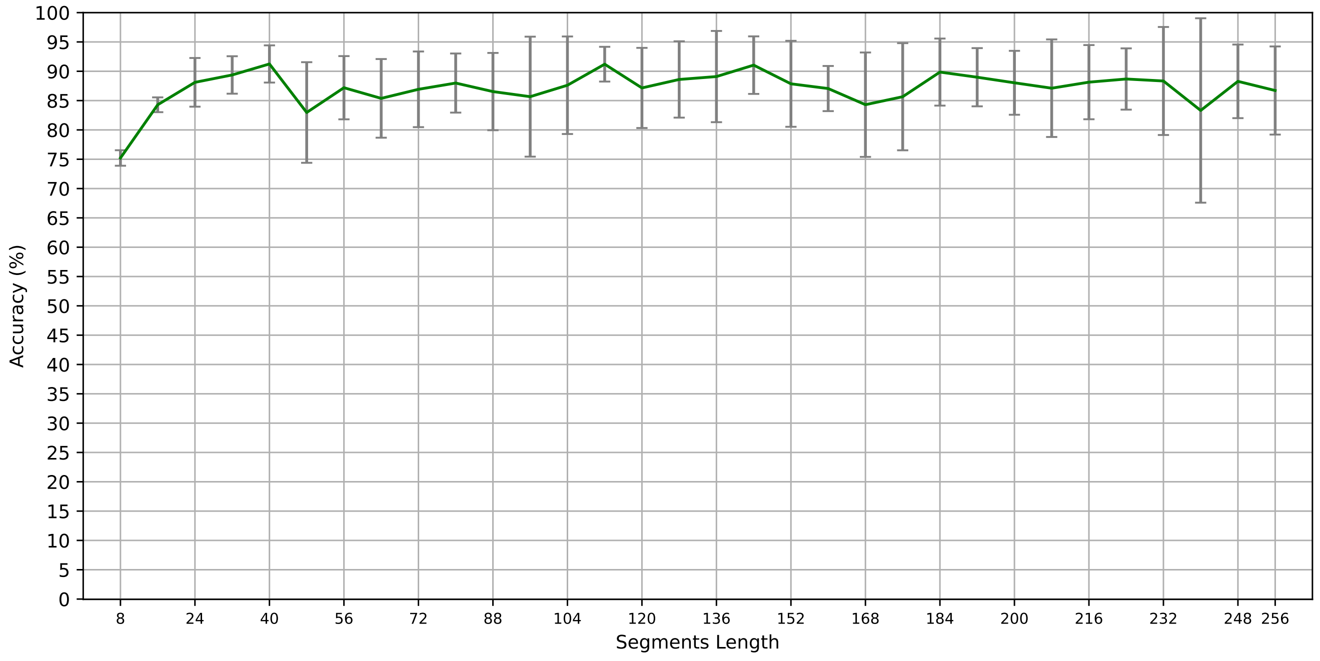


Figure A.36: Accuracy obtained by DA of 3F as feature for Neural Network on the NapierOne Dataset. Each lying point is the average results obtained from ten runs. The vertical lines represent the standard deviation

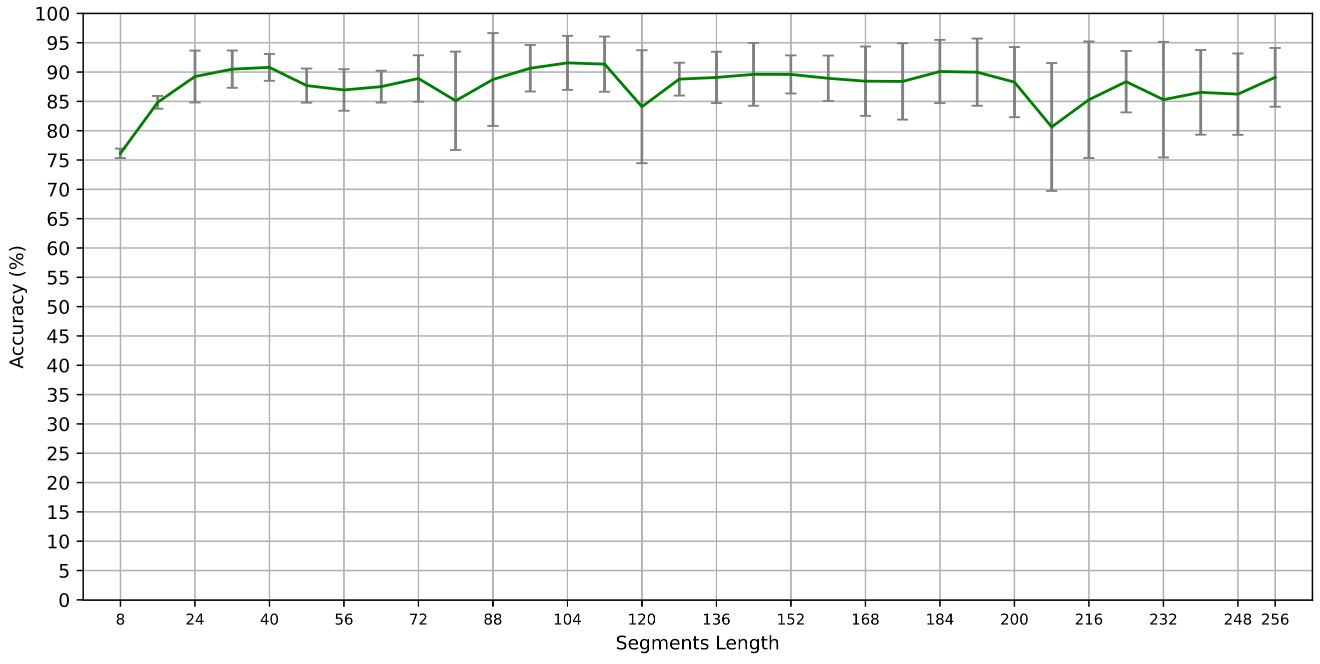


Figure A.37: Accuracy obtained by DA of 4F as feature for Neural Network on the NapierOne Dataset. Each lying point is the average results obtained from ten runs. The vertical lines represent the standard deviation

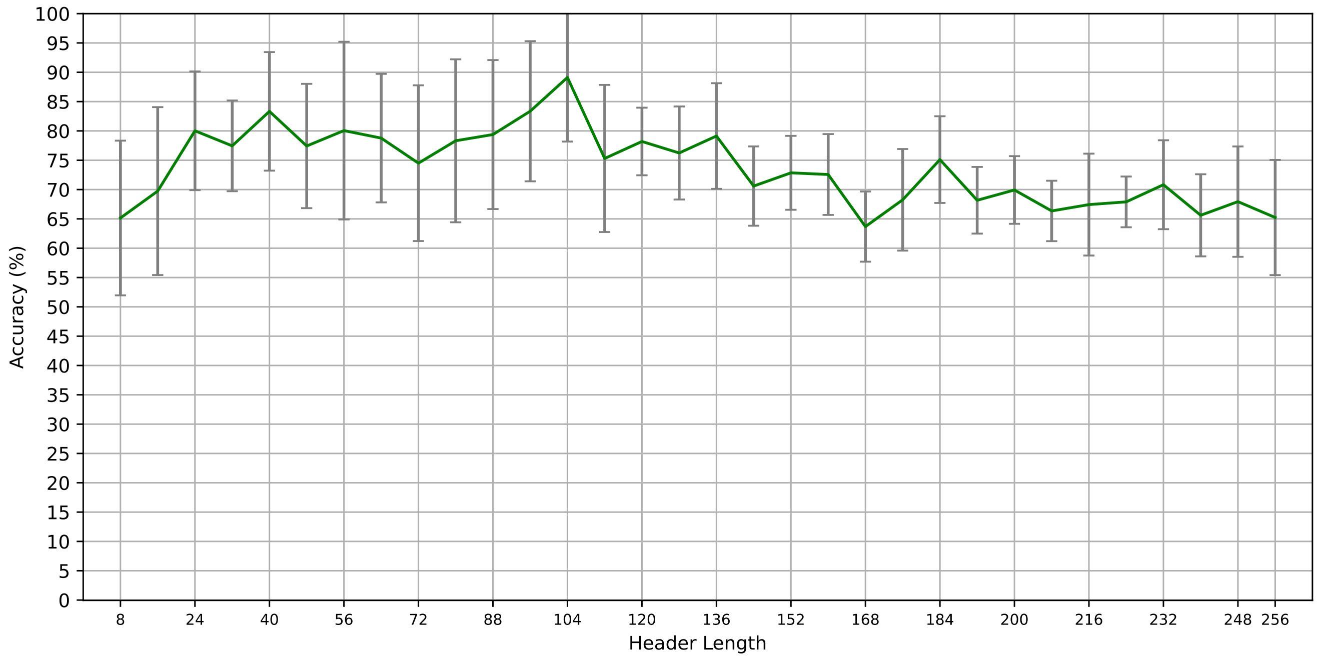


Figure A.38: Accuracy obtained by DA of header as feature for Neural Network on the Attack Dataset. Each lying point is the average results obtained from ten runs. The vertical lines represent the standard deviation

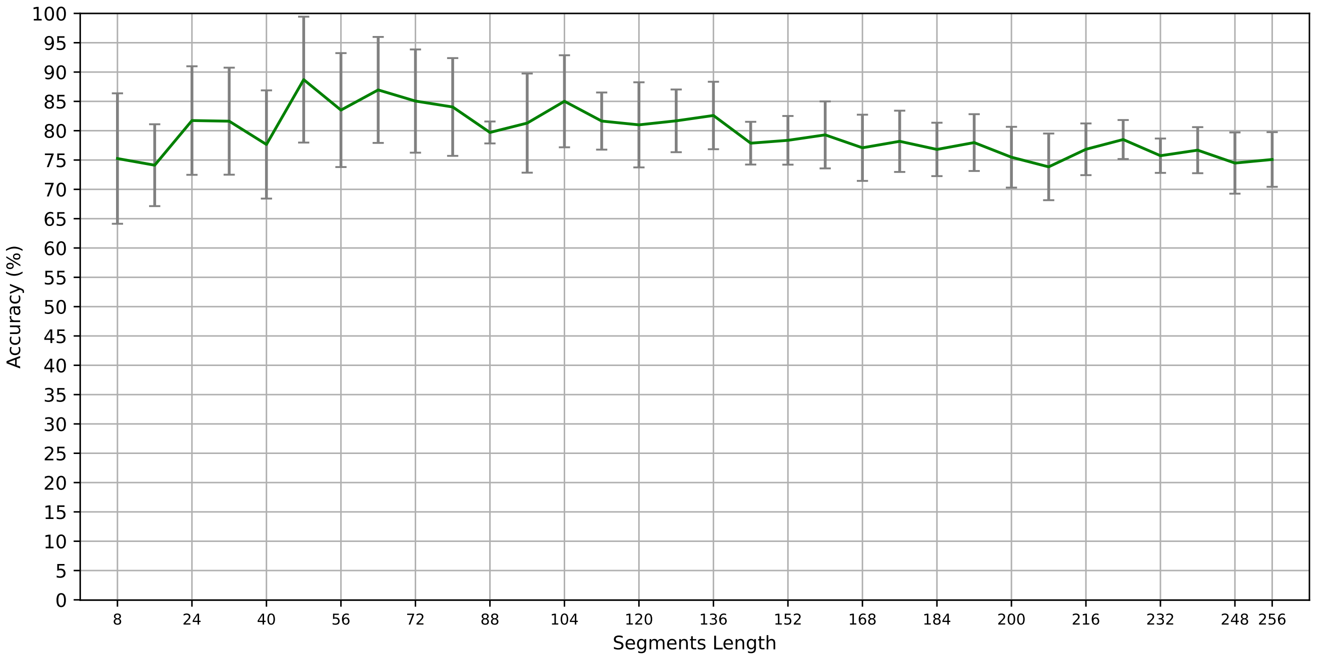


Figure A.39: Accuracy obtained by DA of 2F as feature for Neural Network on the Attack Dataset. Each lying point is the average results obtained from ten runs. The vertical lines represent the standard deviation

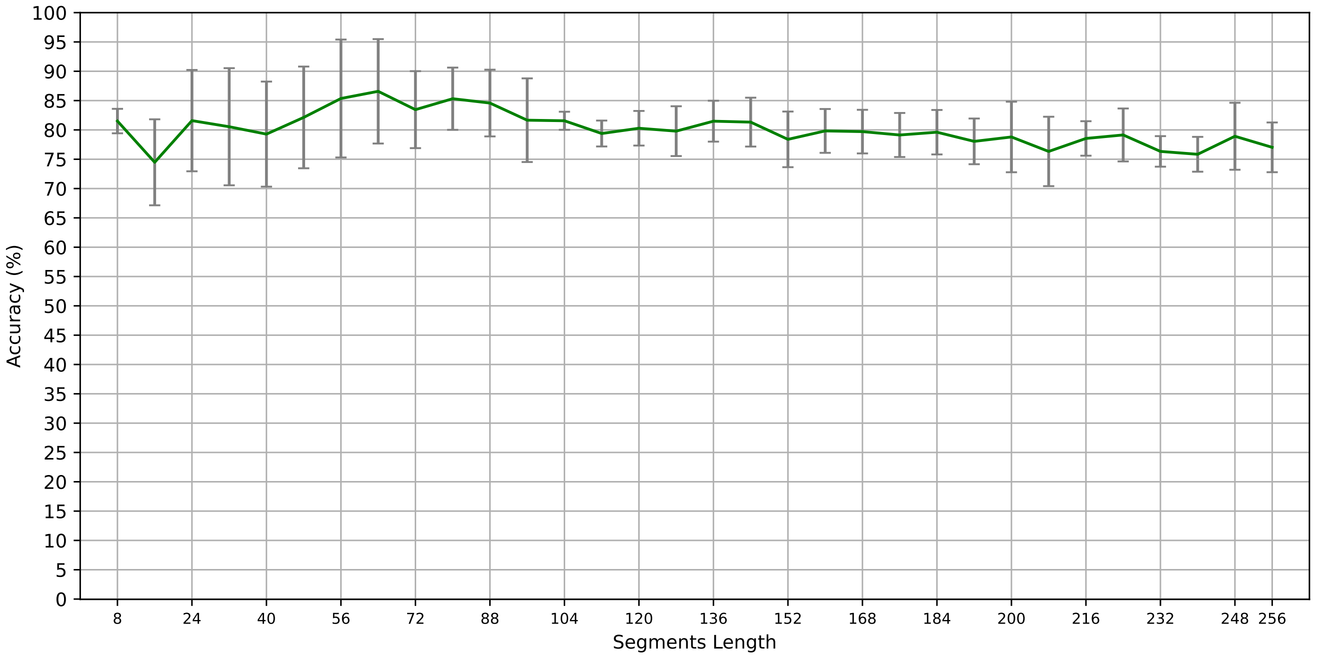


Figure A.40: Accuracy obtained by DA of 3F as feature for Neural Network on the Attack Dataset. Each lying point is the average results obtained from ten runs. The vertical lines represent the standard deviation

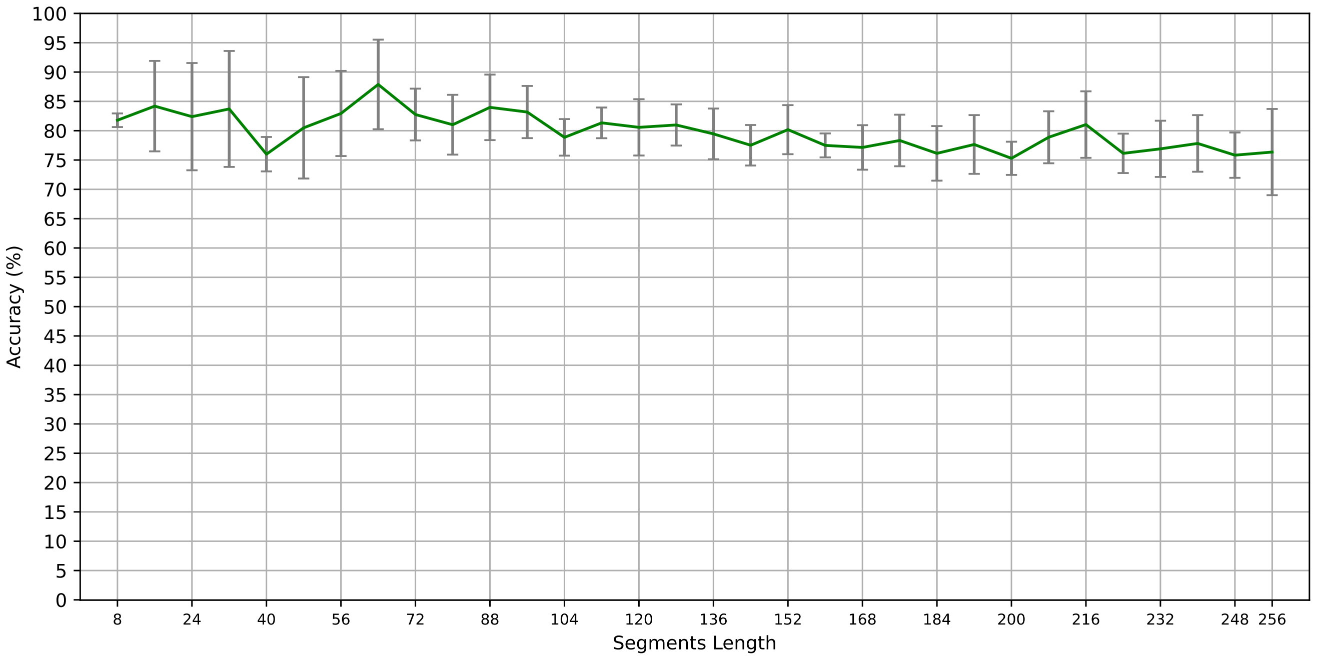


Figure A.41: Accuracy obtained by DA of 4F as feature for Neural Network on the Attack Dataset. Each lying point is the average results obtained from ten runs. The vertical lines represent the standard deviation

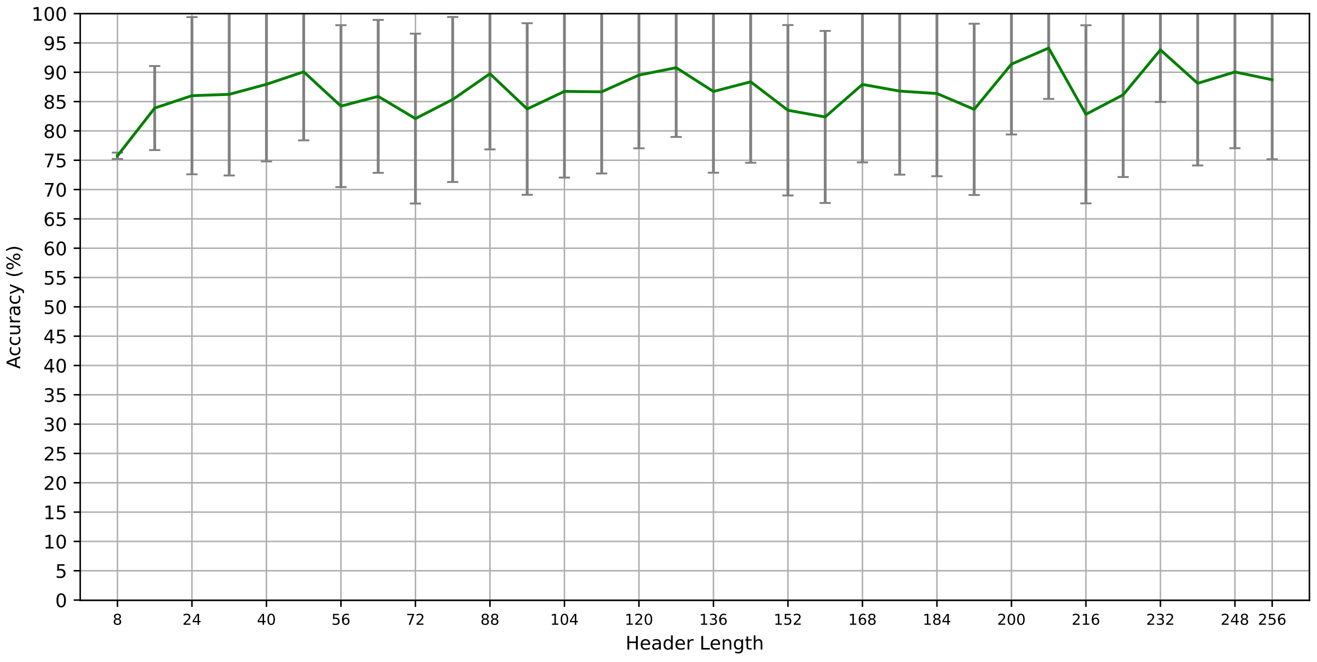


Figure A.42: Accuracy obtained by DA of header as feature for Random Forest on the NapierOne Dataset. Each lying point is the average results obtained from ten runs. The vertical lines represent the standard deviation

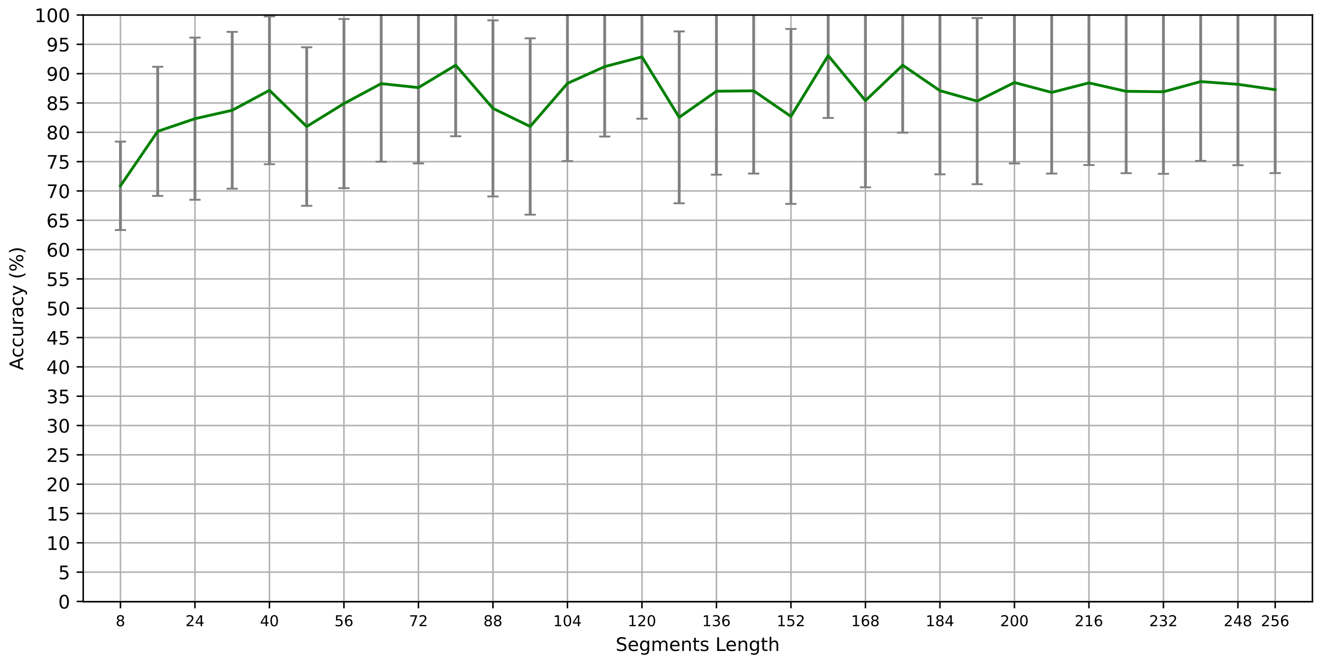


Figure A.43: Accuracy obtained by DA of 2F as feature for Random Forest on the NapierOne Dataset. Each lying point is the average results obtained from ten runs. The vertical lines represent the standard deviation

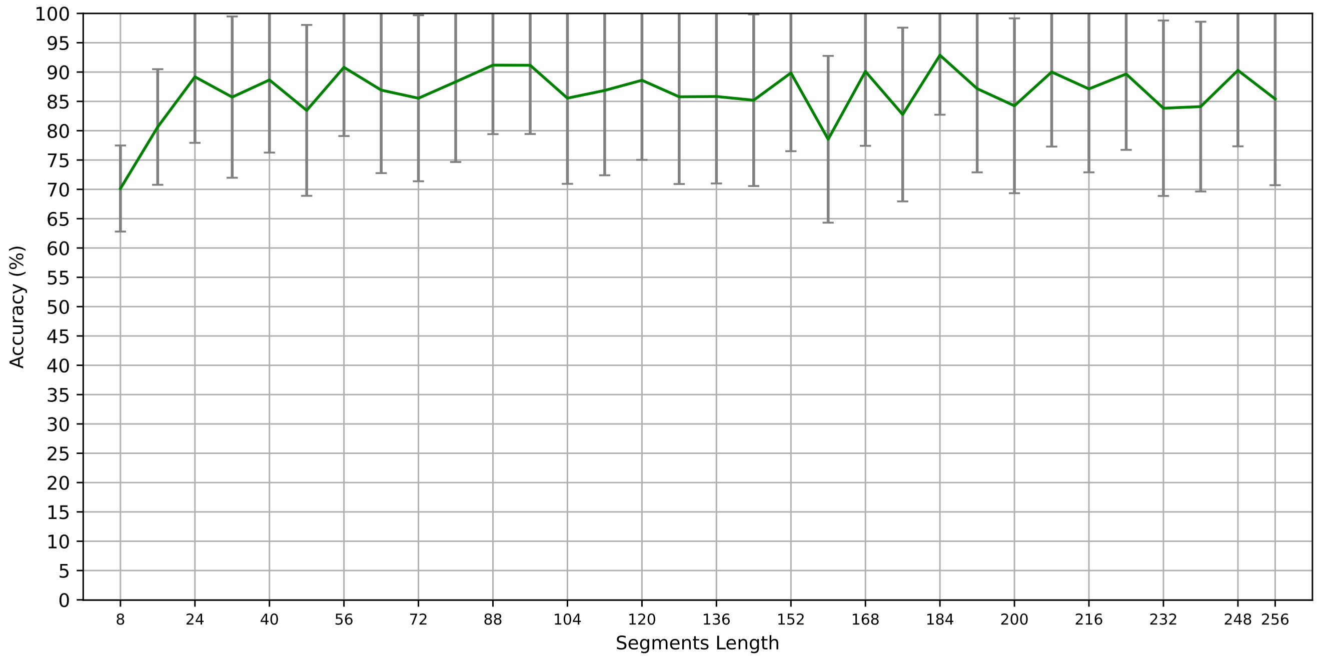


Figure A.44: Accuracy obtained by DA of 3F as feature for Random Forest on the NapierOne Dataset. Each lying point is the average results obtained from ten runs. The vertical lines represent the standard deviation

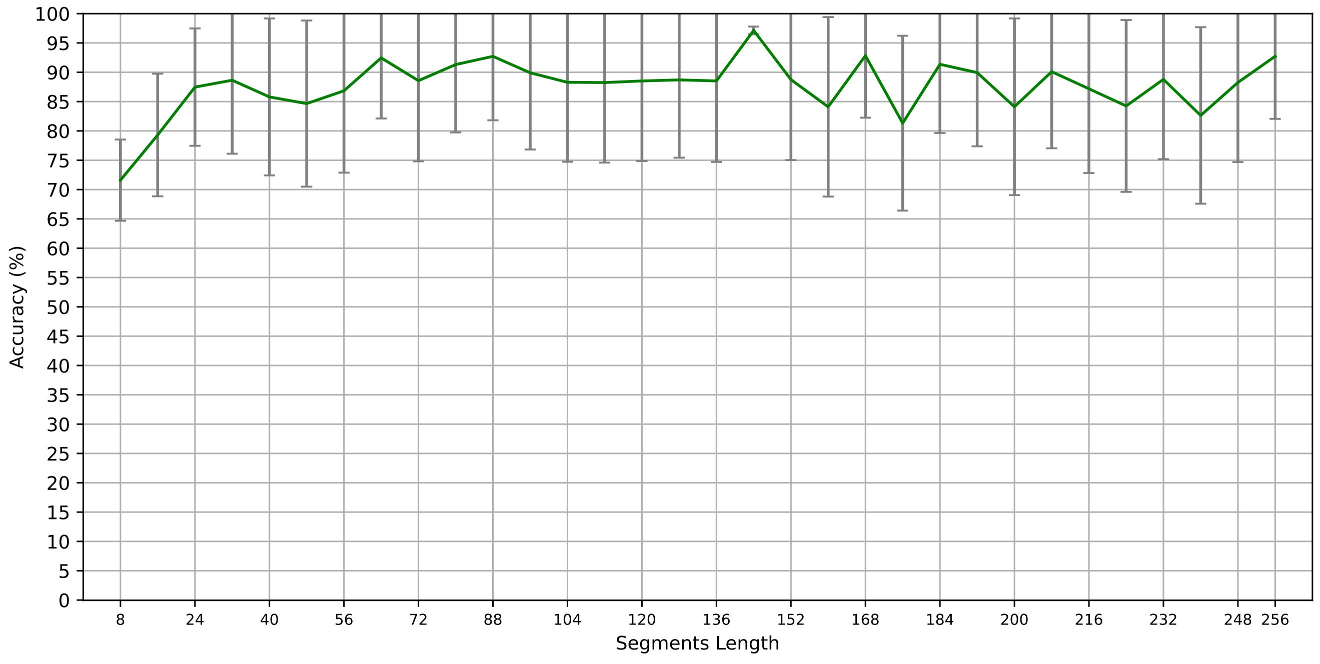


Figure A.45: Accuracy obtained by DA of 4F as feature for Random Forest on the NapierOne Dataset. Each lying point is the average results obtained from ten runs. The vertical lines represent the standard deviation

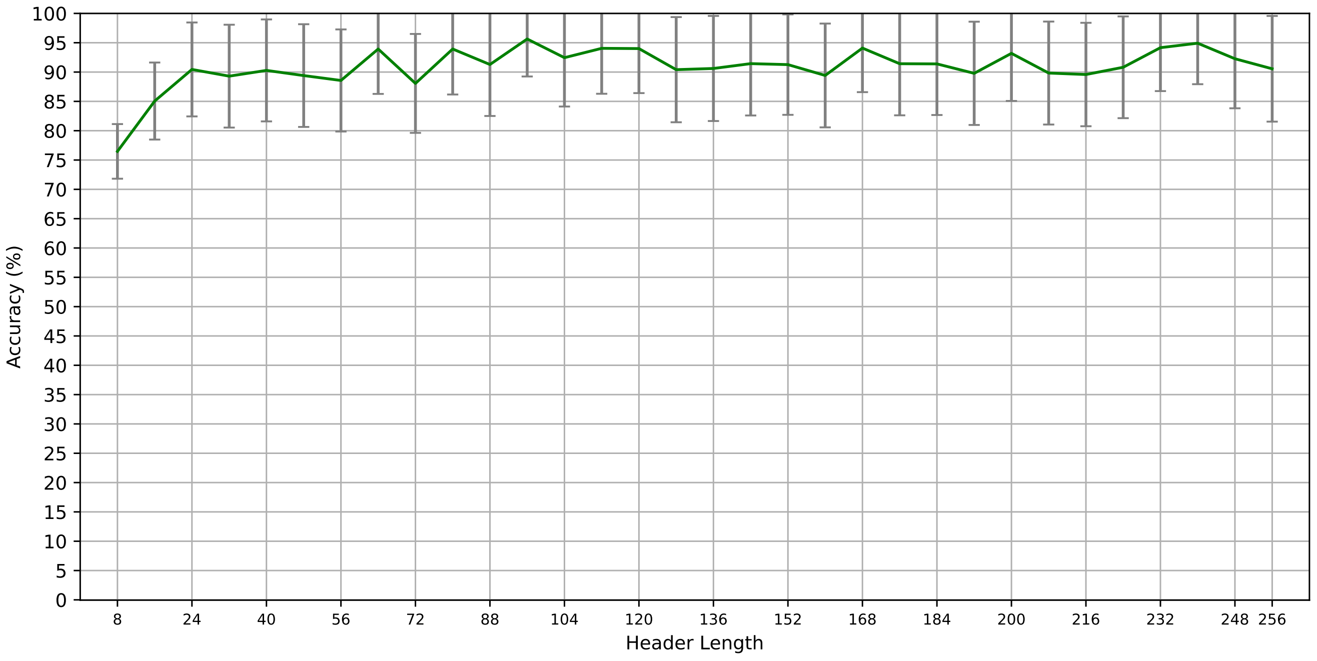


Figure A.46: Accuracy obtained by DA of header as feature for Random Forest on the Attack Dataset. Each lying point is the average results obtained from ten runs. The vertical lines represent the standard deviation

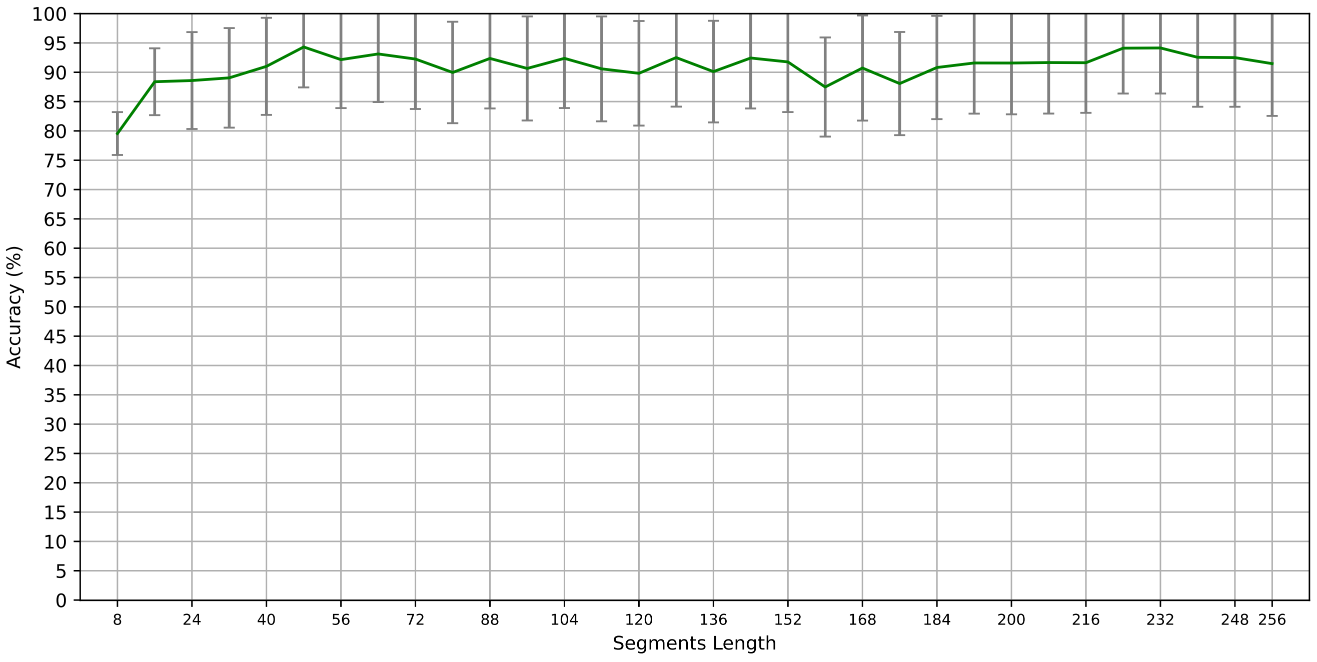


Figure A.47: Accuracy obtained by DA of 2F as feature for Random Forest on the Attack Dataset. Each lying point is the average results obtained from ten runs. The vertical lines represent the standard deviation

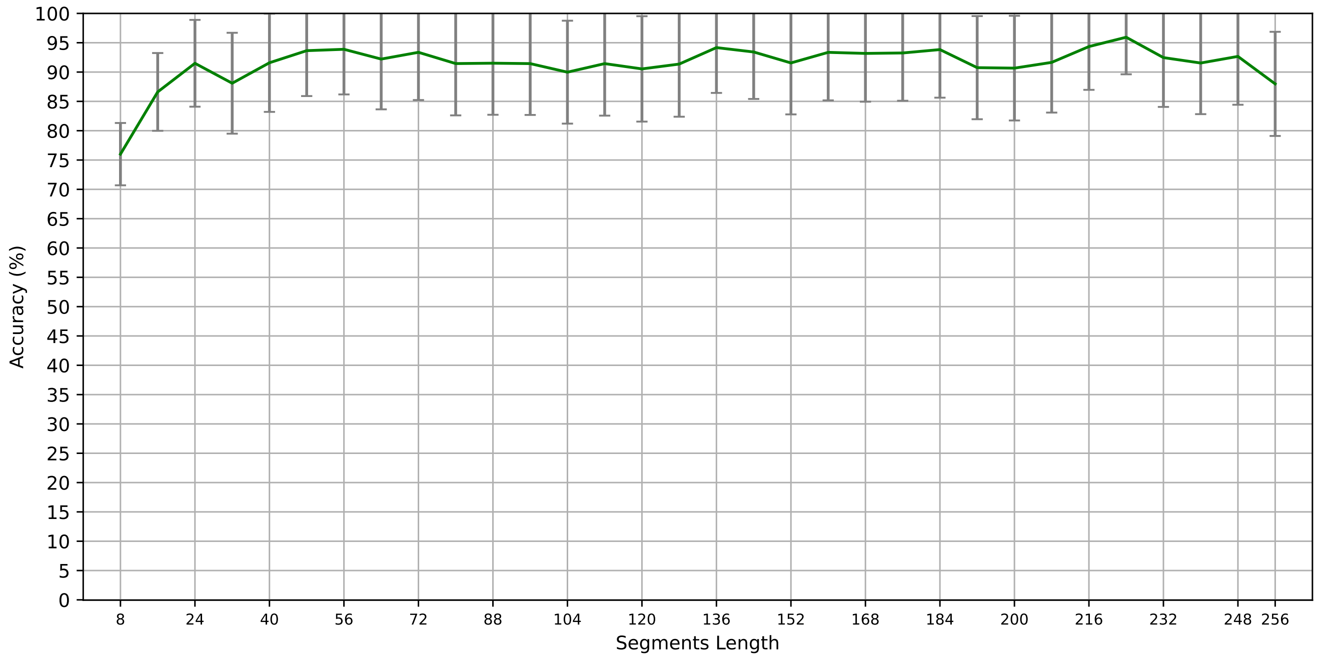


Figure A.48: Accuracy obtained by DA of 3F as feature for Random Forest on the Attack Dataset. Each lying point is the average results obtained from ten runs. The vertical lines represent the standard deviation

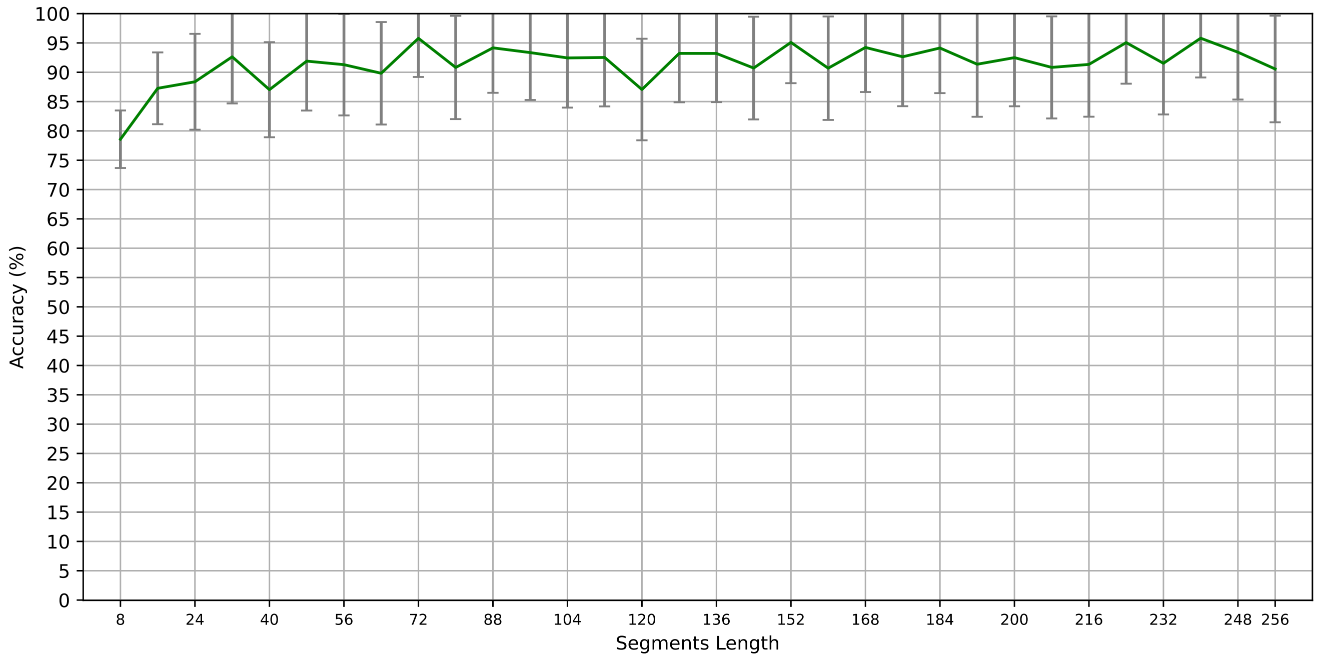


Figure A.49: Accuracy obtained by DA of 4F as feature for Random Forest on the Attack Dataset. Each lying point is the average results obtained from ten runs. The vertical lines represent the standard deviation

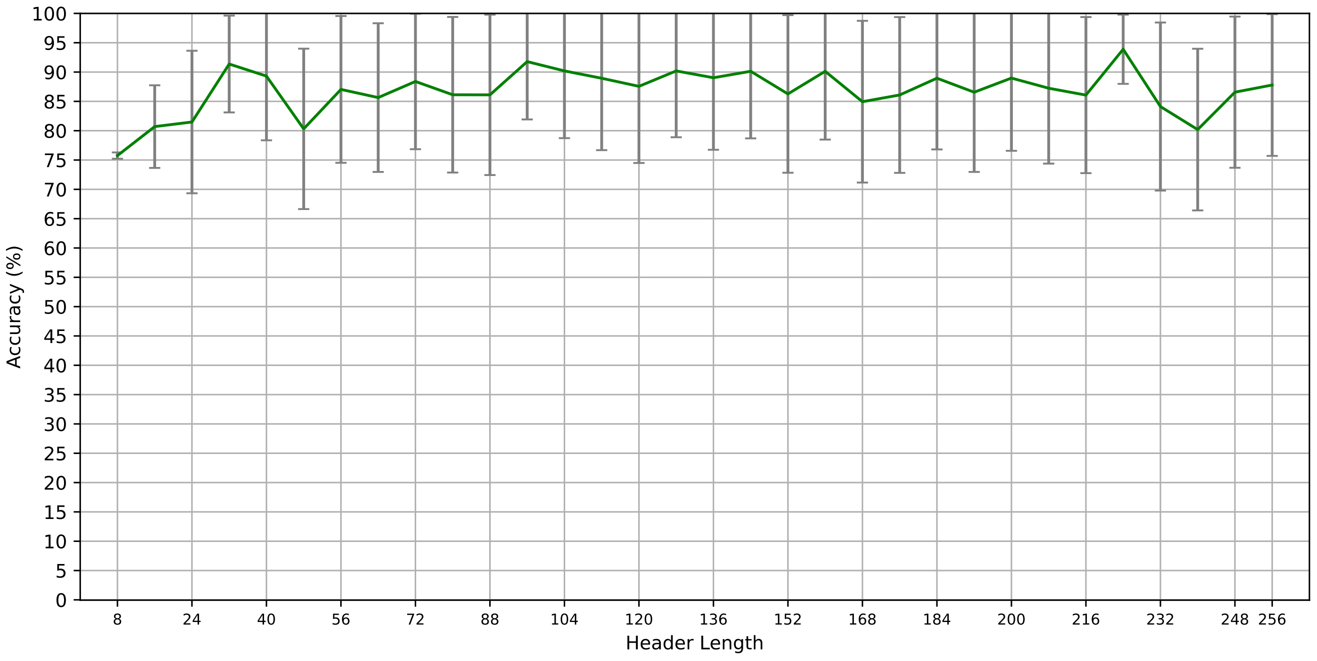


Figure A.50: Accuracy obtained by DA of header as feature for SVC on the NapierOne Dataset. Each lying point is the average results obtained from ten runs. The vertical lines represent the standard deviation

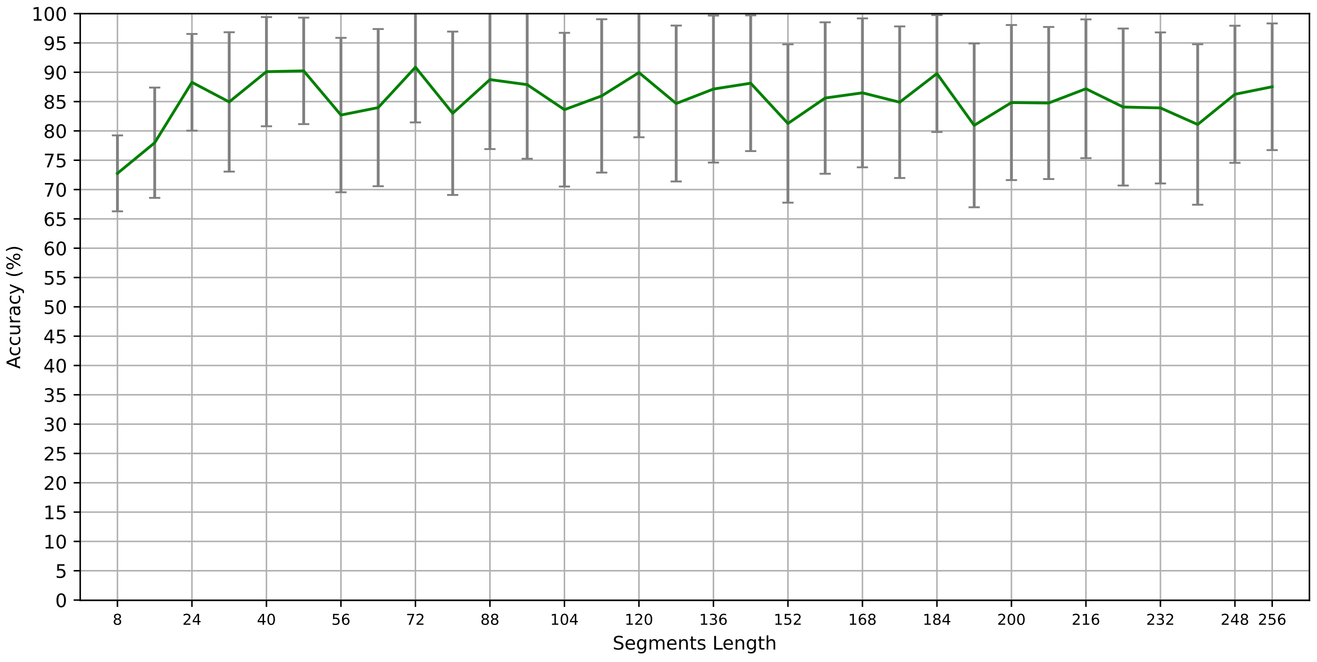


Figure A.51: Accuracy obtained by DA of 2F as feature for SVC on the NapierOne Dataset. Each lying point is the average results obtained from ten runs. The vertical lines represent the standard deviation

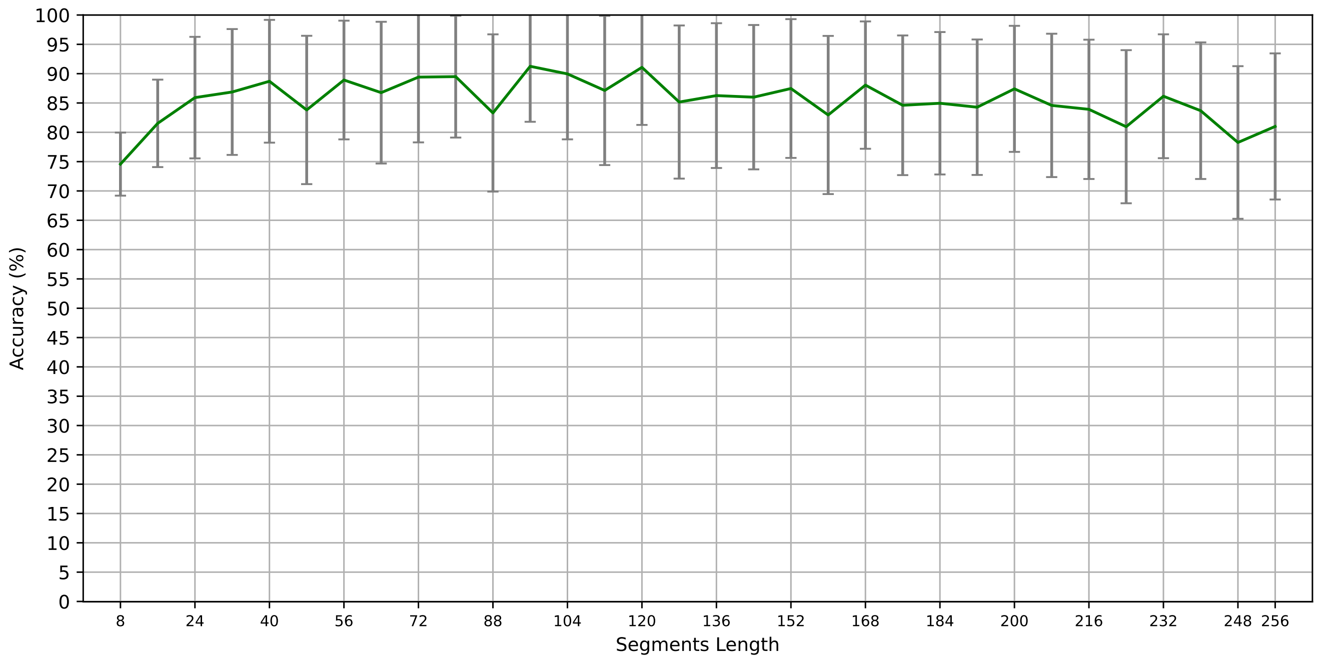


Figure A.52: Accuracy obtained by DA of 3F as feature for SVC on the NapierOne Dataset. Each lying point is the average results obtained from ten runs. The vertical lines represent the standard deviation

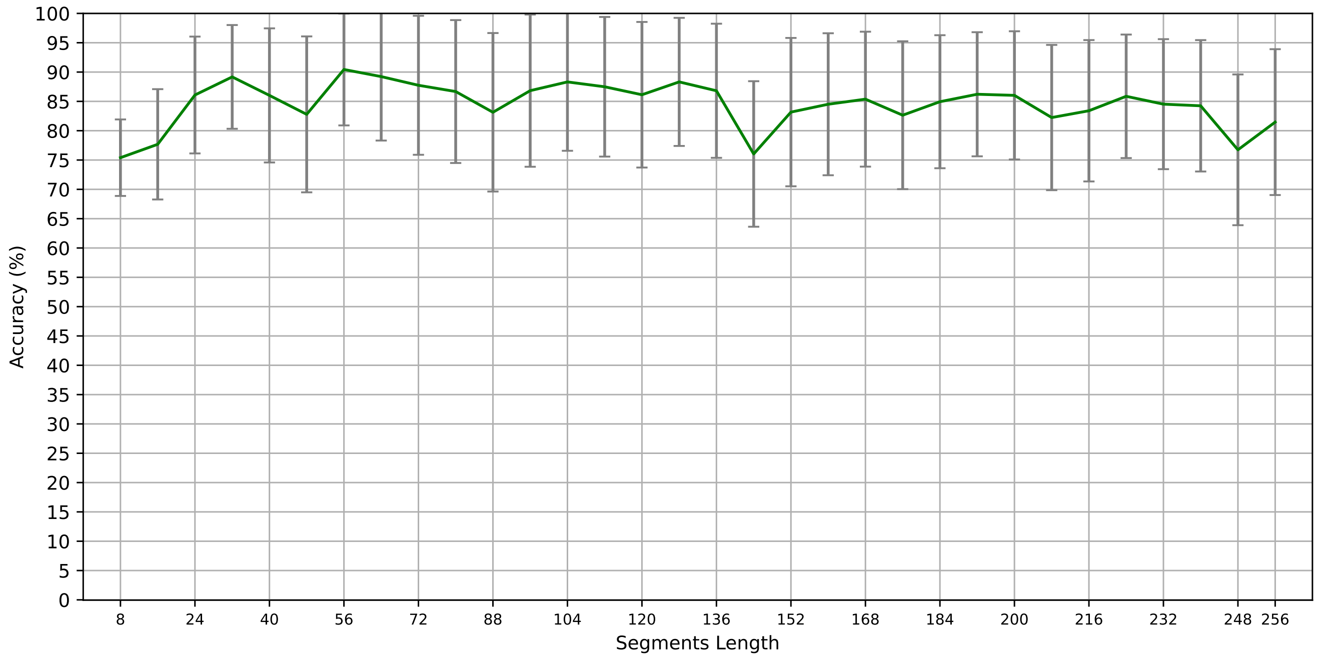


Figure A.53: Accuracy obtained by DA of 4F as feature for SVC on the NapierOne Dataset. Each lying point is the average results obtained from ten runs. The vertical lines represent the standard deviation

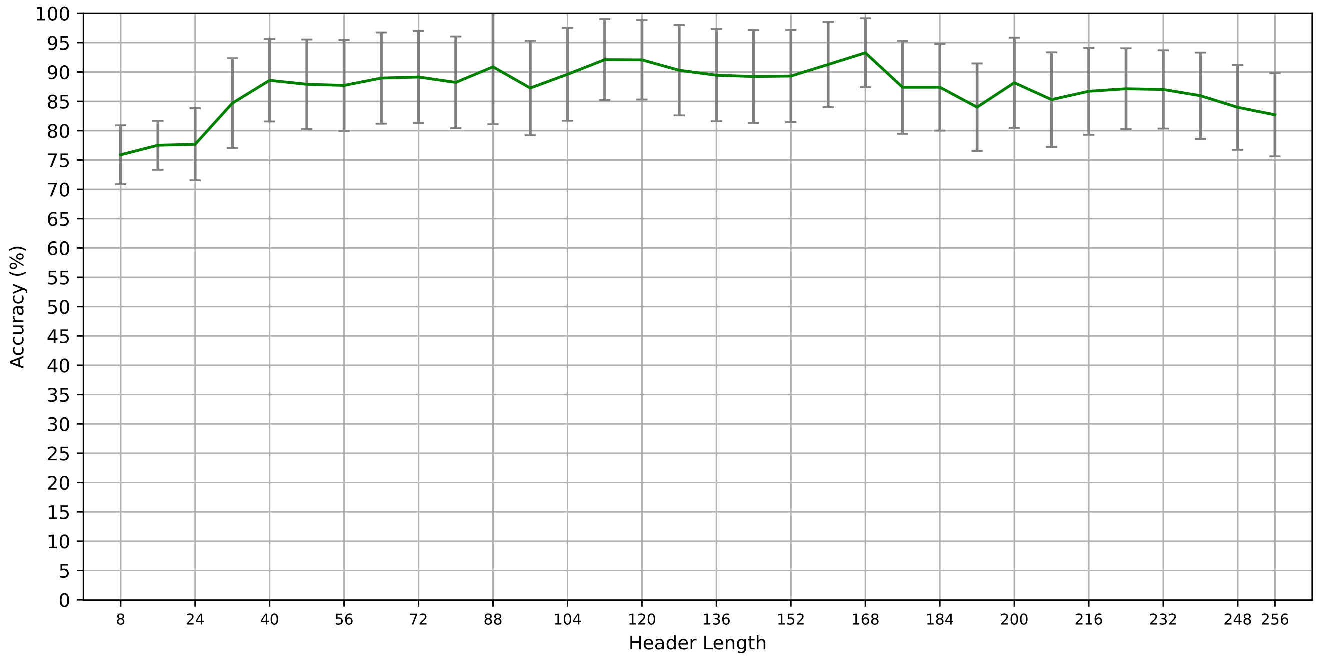


Figure A.54: Accuracy obtained by DA of header as feature for SVC on the Attack Dataset. Each lying point is the average results obtained from ten runs. The vertical lines represent the standard deviation

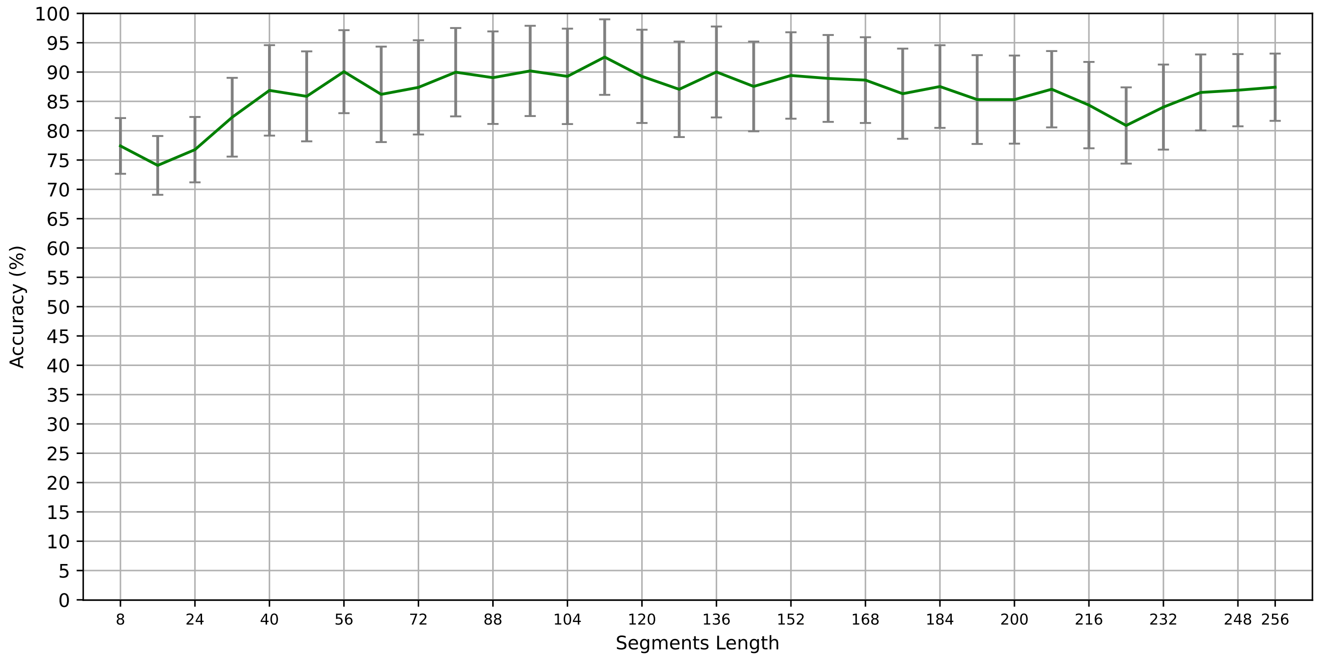


Figure A.55: Accuracy obtained by DA of 2F as feature for SVC on the Attack Dataset. Each lying point is the average results obtained from ten runs. The vertical lines represent the standard deviation

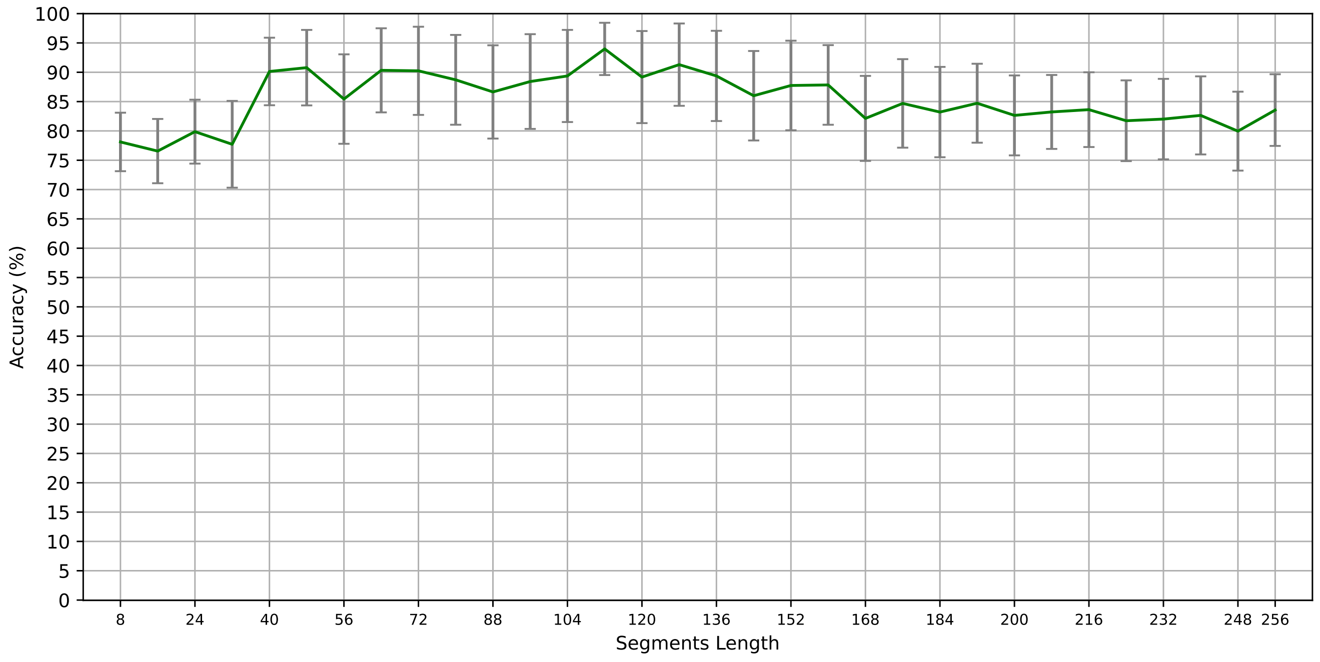


Figure A.56: Accuracy obtained by DA of 3F as feature for SVC on the Attack Dataset. Each lying point is the average results obtained from ten runs. The vertical lines represent the standard deviation

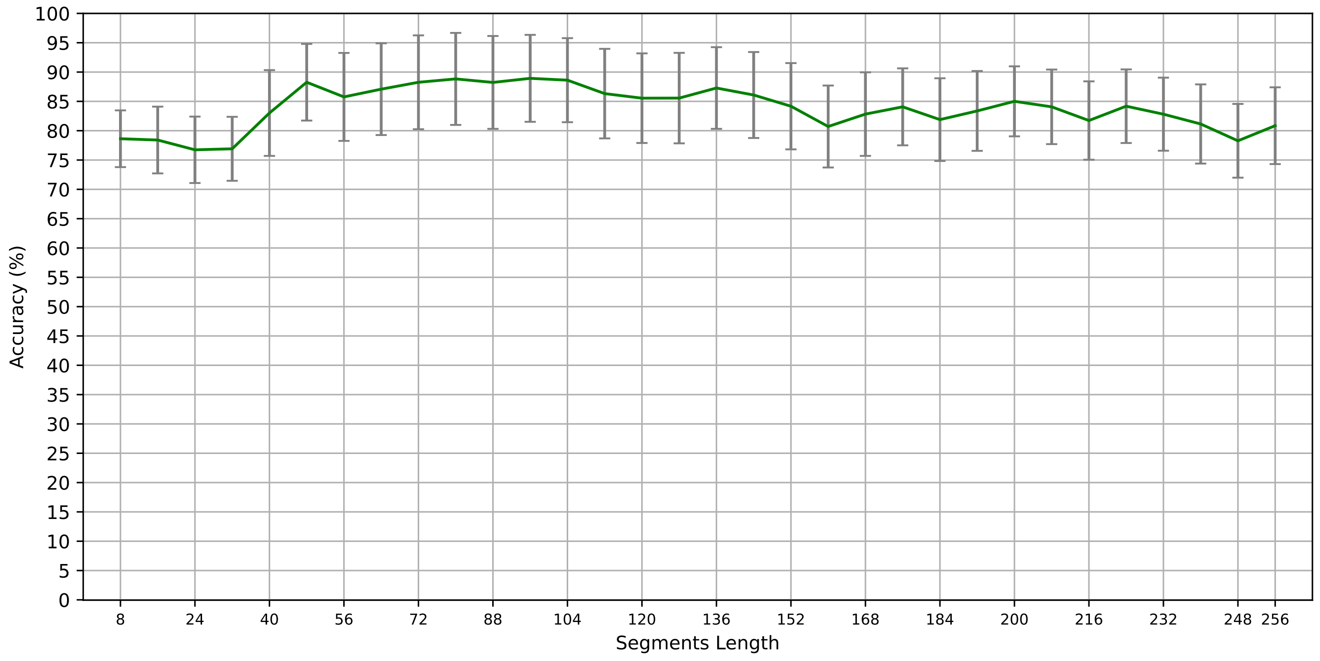


Figure A.57: Accuracy obtained by DA of 4F as feature for SVC on the NapierOne Dataset. Each lying point is the average results obtained from ten runs. The vertical lines represent the standard deviation

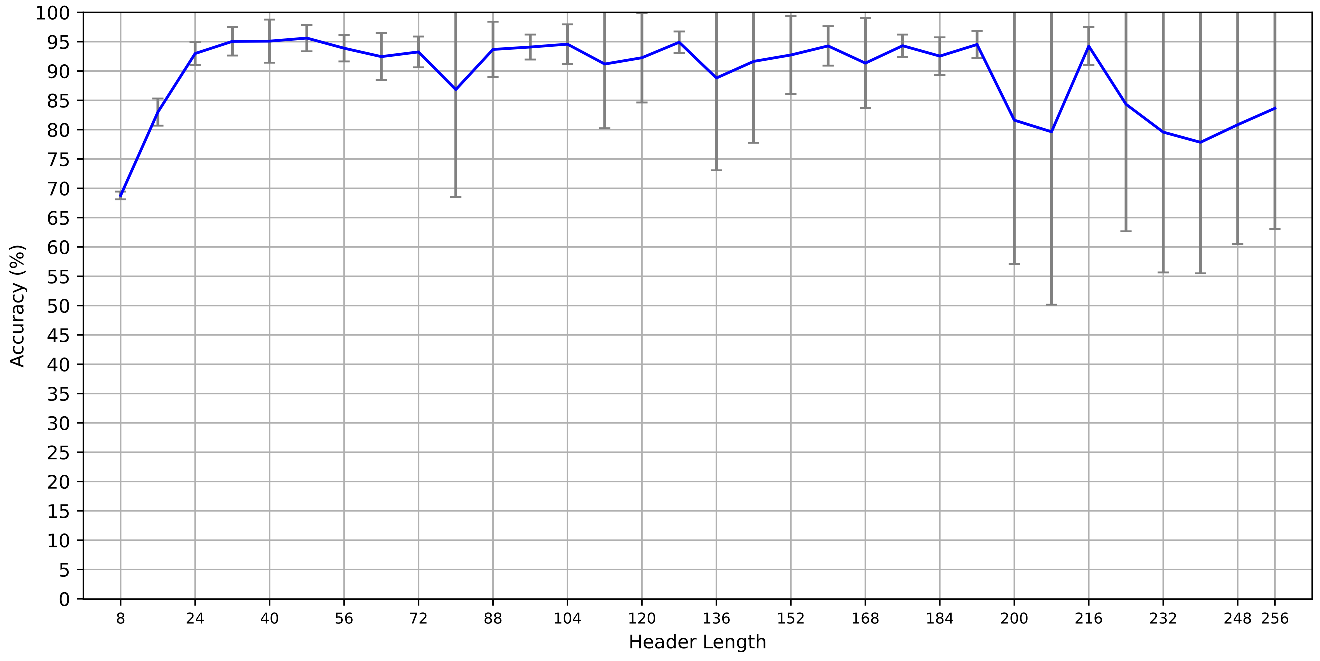


Figure A.58: Accuracy obtained by Entropy and DA of header as features for Neural Network on the NapierOne Dataset. Each lying point is the average results obtained from ten runs. The vertical lines represent the standard deviation

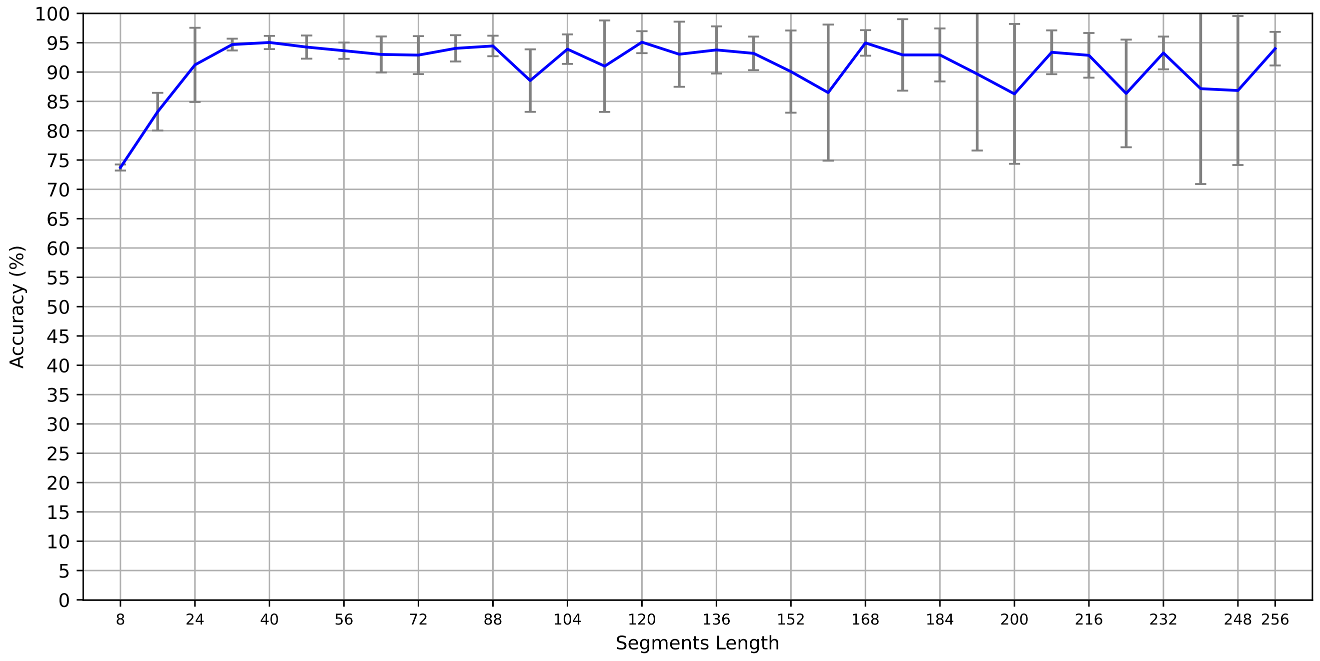


Figure A.59: Accuracy obtained by Entropy and DA of 2F as features for Neural Network on the NapierOne Dataset. Each lying point is the average results obtained from ten runs. The vertical lines represent the standard deviation

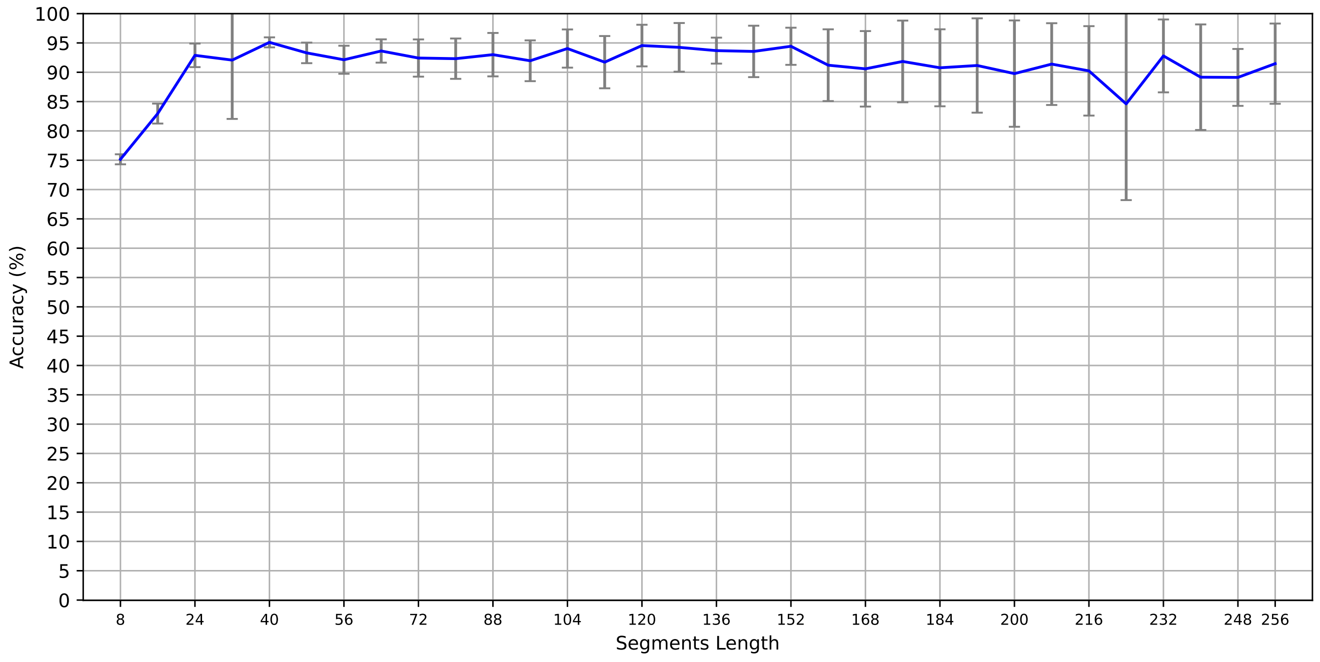


Figure A.60: Accuracy obtained by Entropy and DA of 3F as features for Neural Network on the NapierOne Dataset. Each lying point is the average results obtained from ten runs. The vertical lines represent the standard deviation

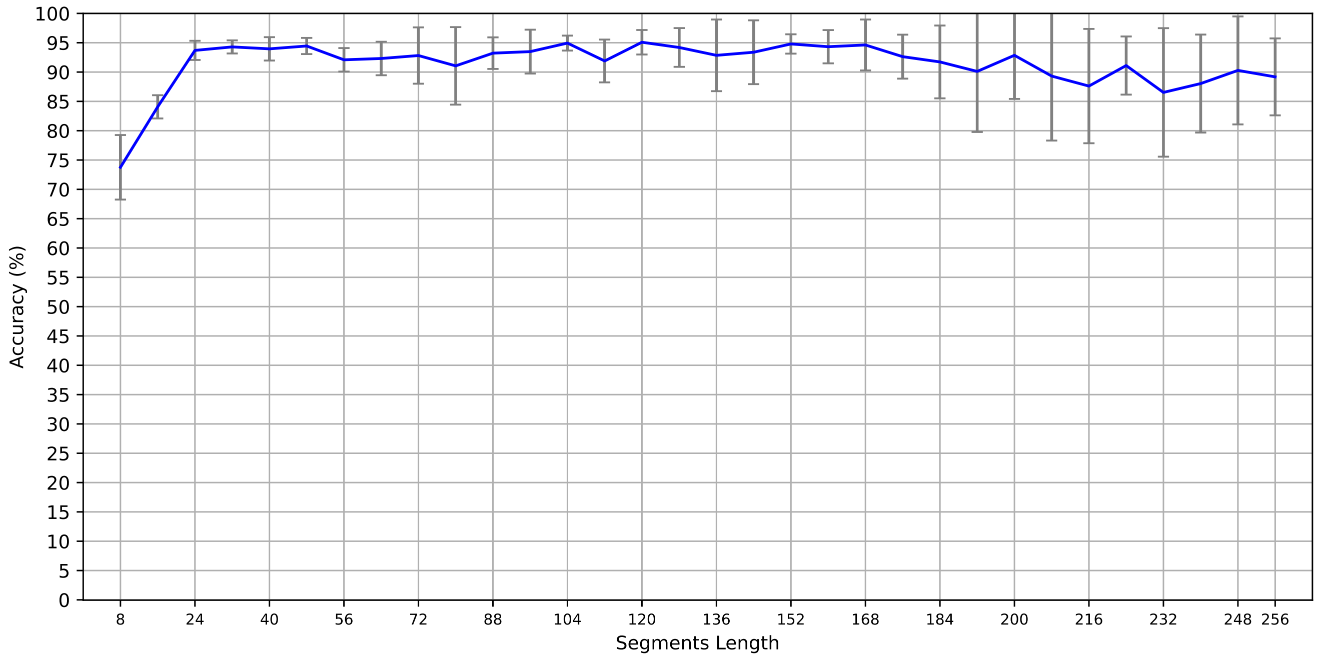


Figure A.61: Accuracy obtained by Entropy and DA of 4F as features for Neural Network on the NapierOne Dataset. Each lying point is the average results obtained from ten runs. The vertical lines represent the standard deviation

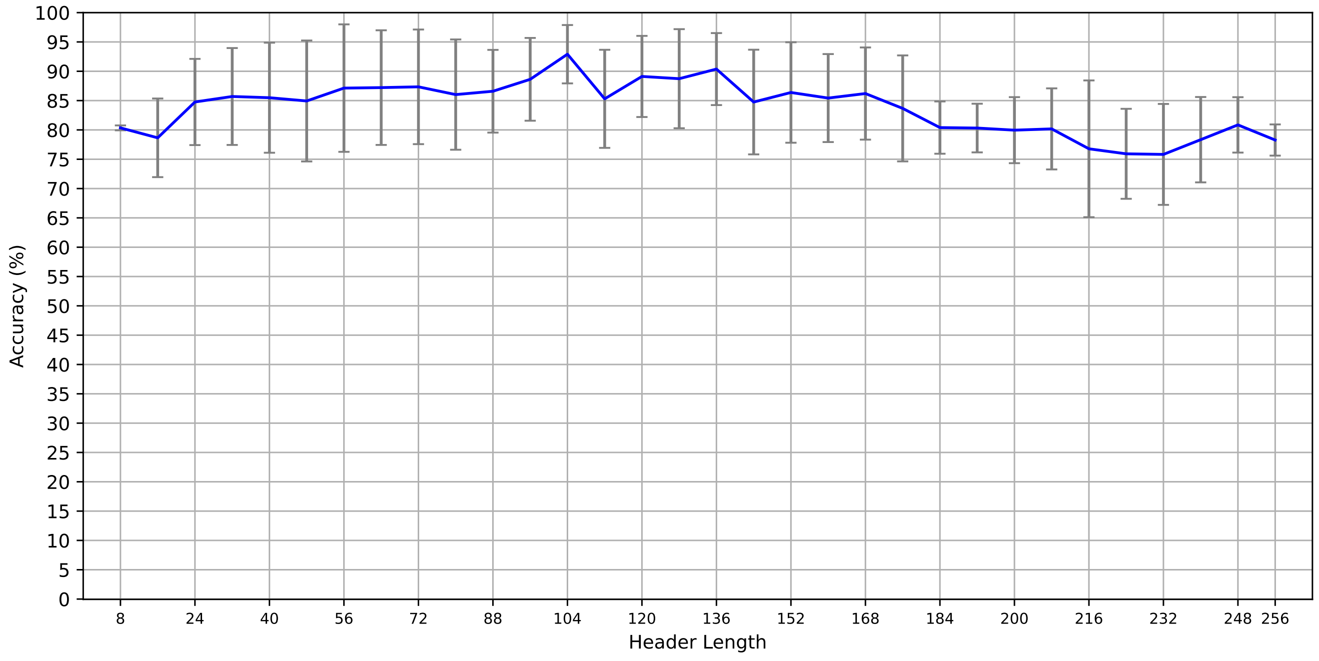


Figure A.62: Accuracy obtained by Entropy and DA of header as features for Neural Network on the Attack Dataset. Each lying point is the average results obtained from ten runs. The vertical lines represent the standard deviation

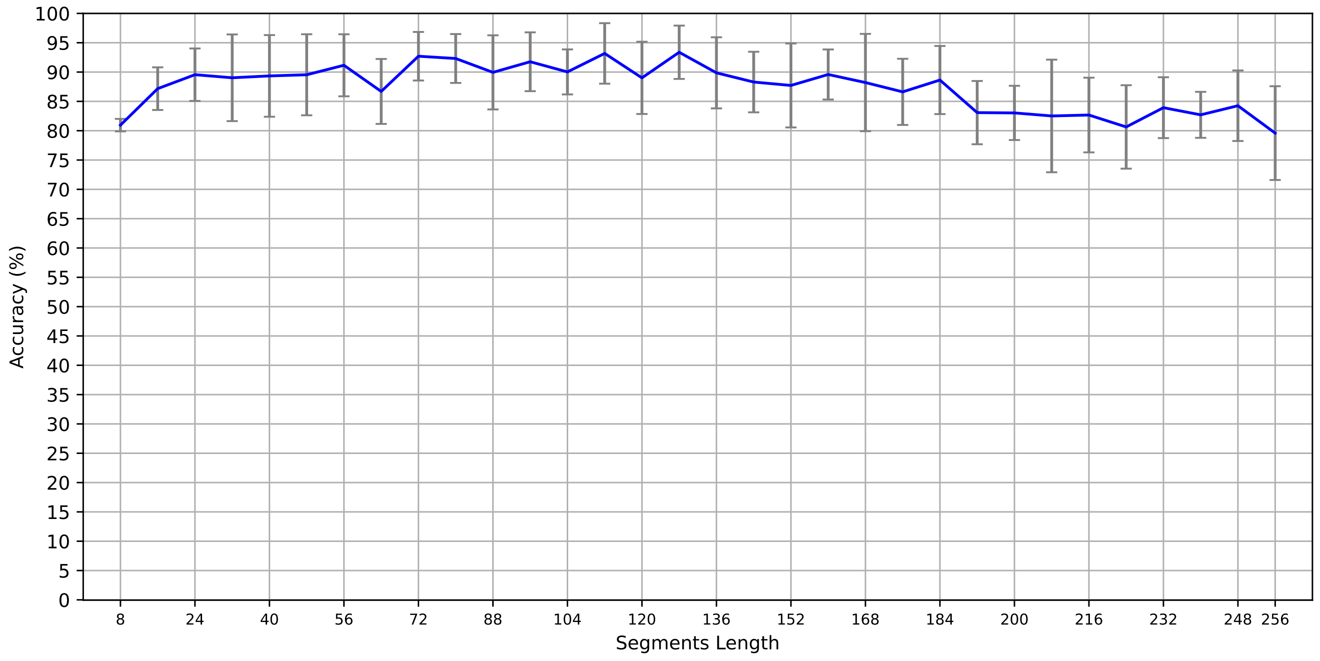


Figure A.63: Accuracy obtained by Entropy and DA of 2F as features for Neural Network on the Attack Dataset. Each lying point is the average results obtained from ten runs. The vertical lines represent the standard deviation

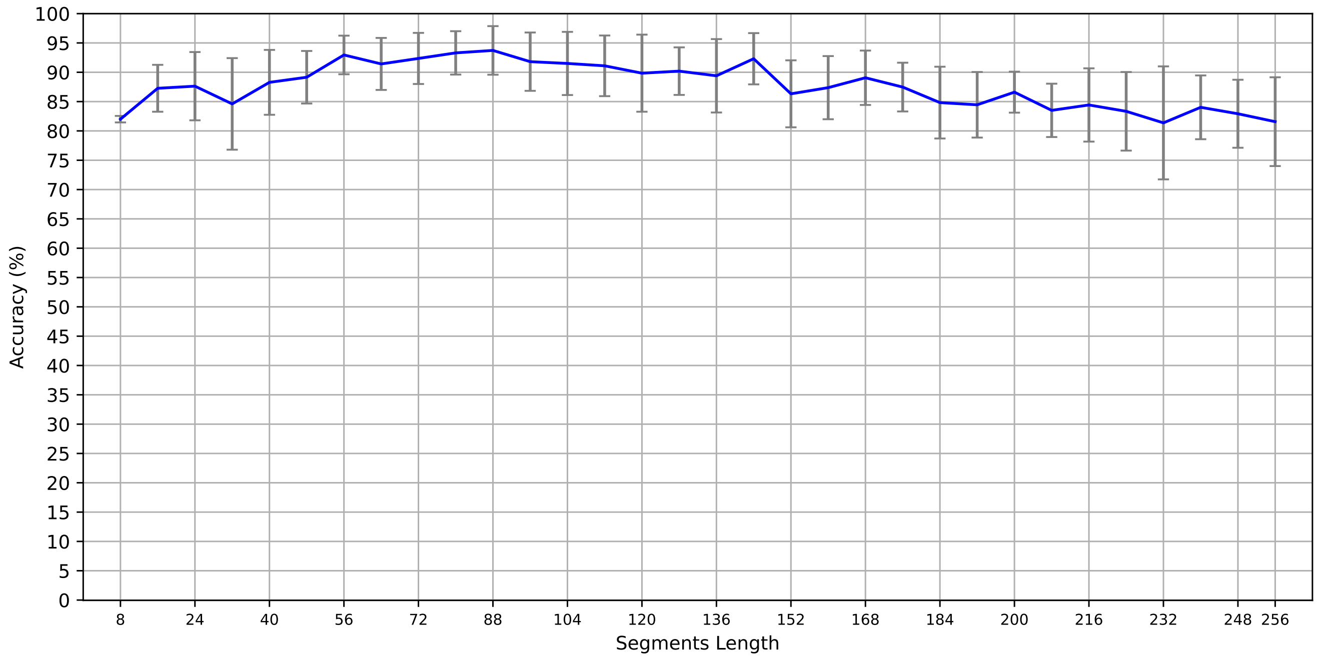


Figure A.64: Accuracy obtained by Entropy and DA of 3F as features for Neural Network on the Attack Dataset. Each lying point is the average results obtained from ten runs. The vertical lines represent the standard deviation

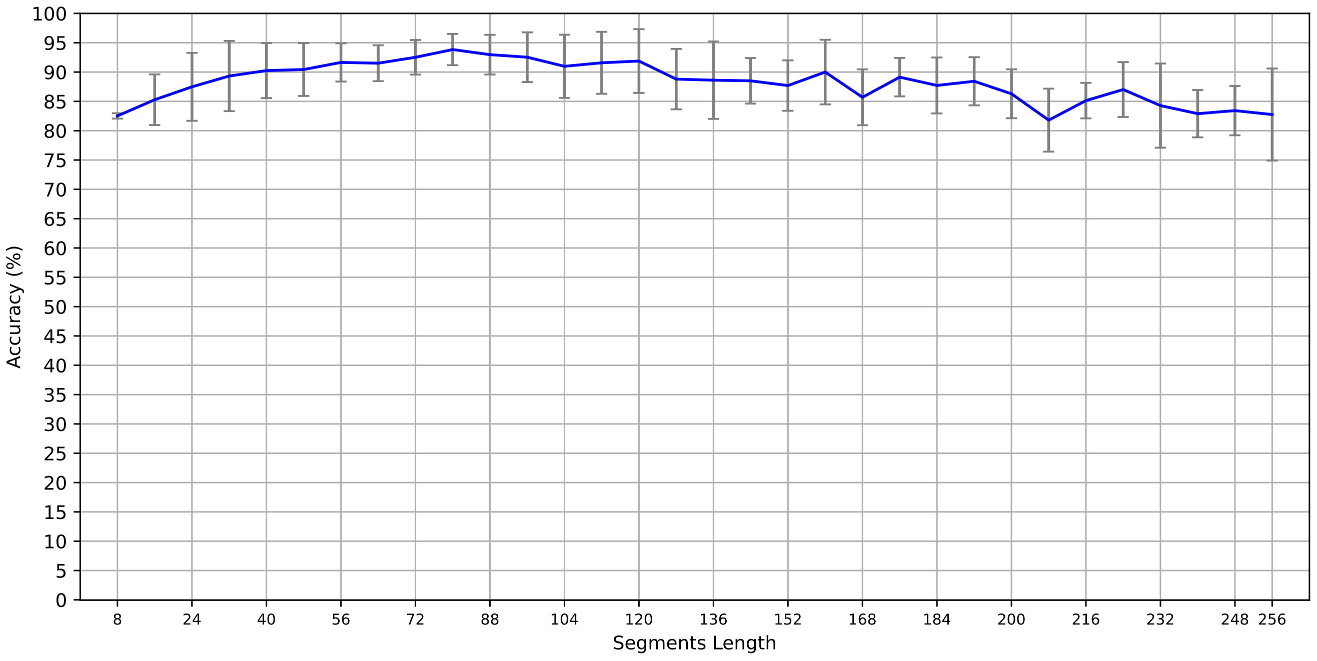


Figure A.65: Accuracy obtained by Entropy and DA of 4F as features for Neural Network on the Attack Dataset. Each lying point is the average results obtained from ten runs. The vertical lines represent the standard deviation

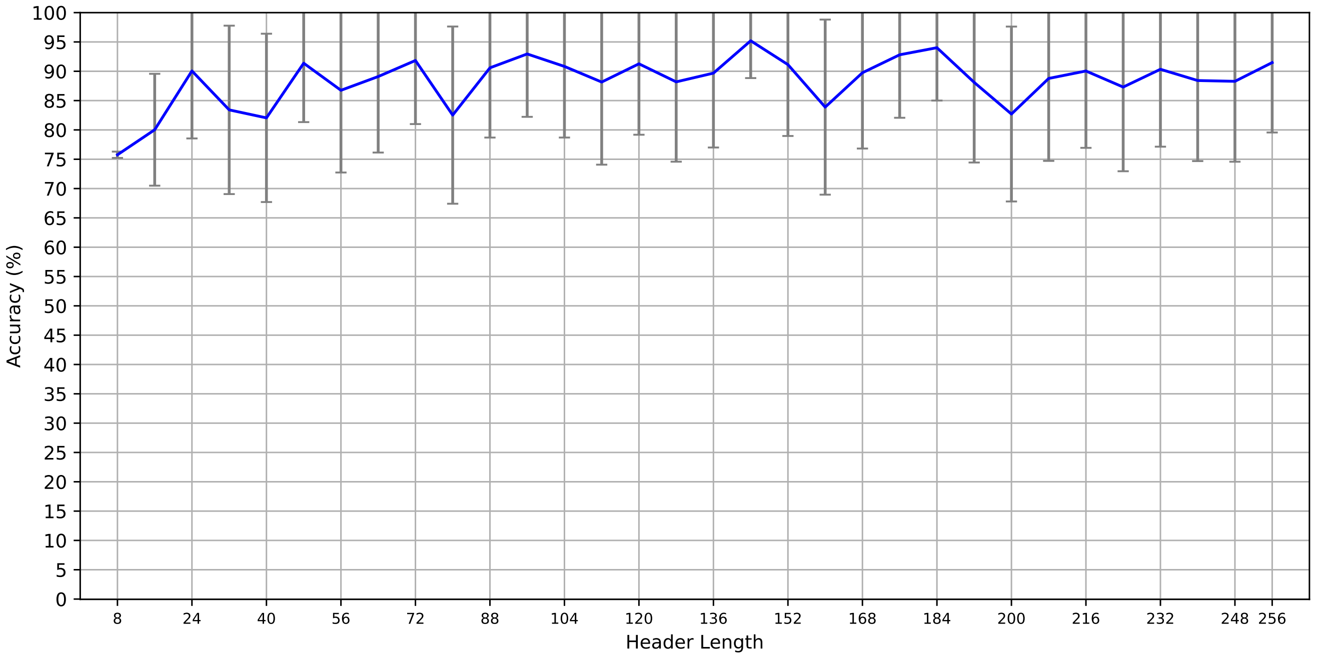


Figure A.66: Accuracy obtained by Entropy and DA of header as features for Random Forest on the NapierOne Dataset. Each lying point is the average results obtained from ten runs. The vertical lines represent the standard deviation

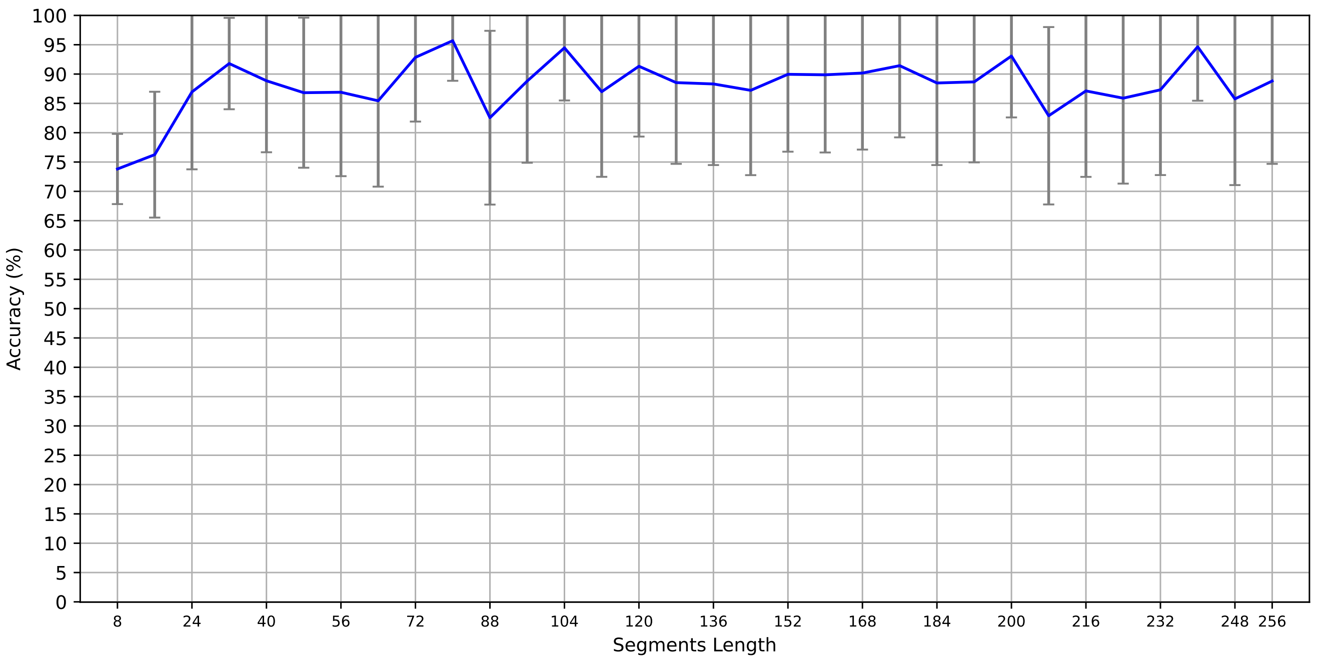


Figure A.67: Accuracy obtained by Entropy and DA of 2F as features for Random Forest on the NapierOne Dataset. Each lying point is the average results obtained from ten runs. The vertical lines represent the standard deviation

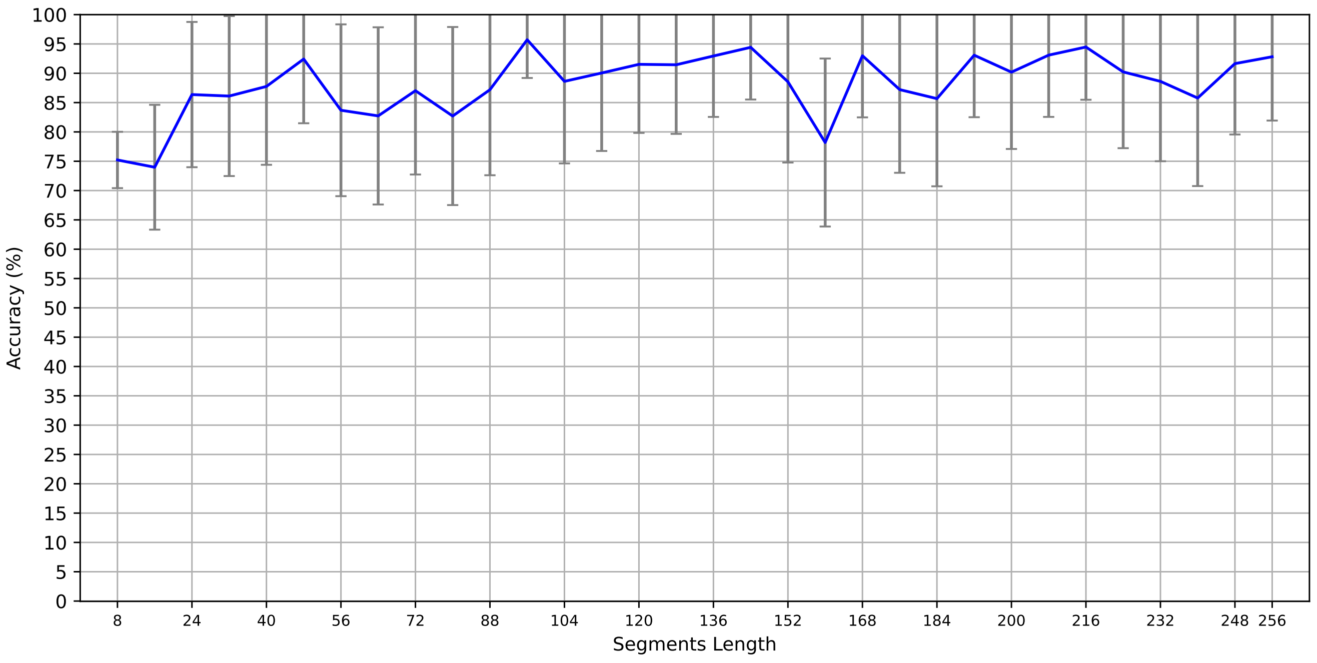


Figure A.68: Accuracy obtained by Entropy and DA of 3F as features for Random Forest on the NapierOne Dataset. Each lying point is the average results obtained from ten runs. The vertical lines represent the standard deviation

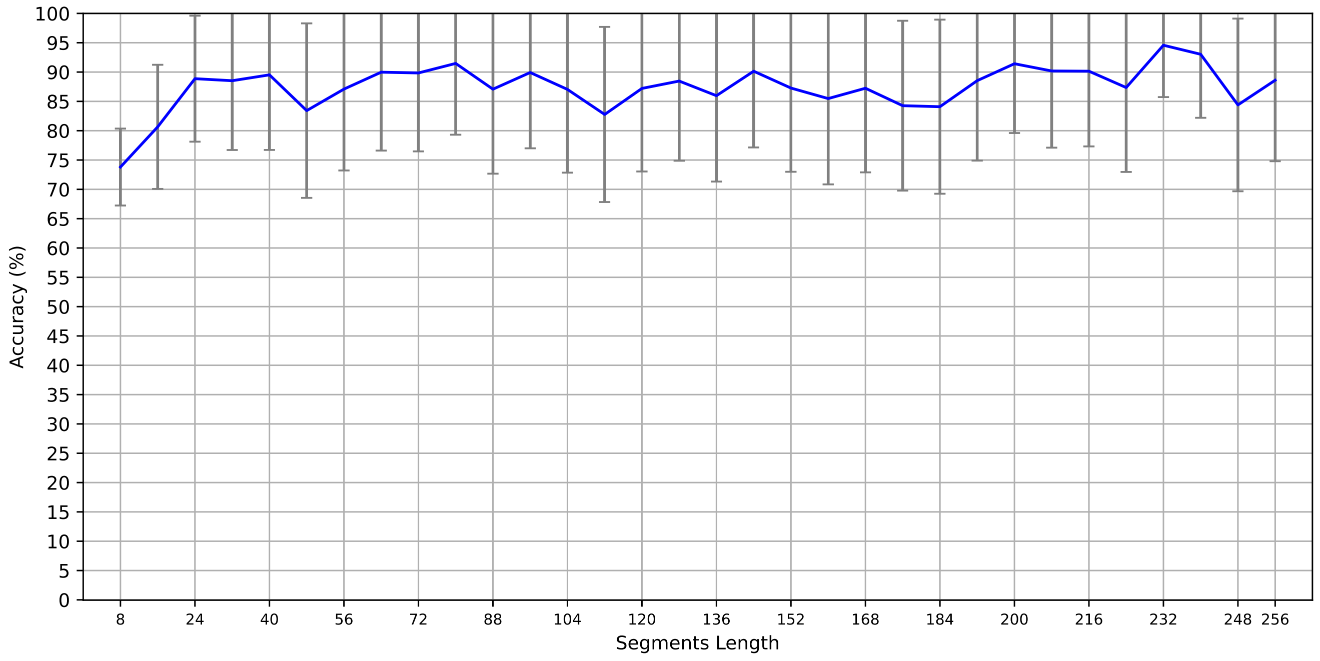


Figure A.69: Accuracy obtained by Entropy and DA of 4F as features for Random Forest on the NapierOne Dataset. Each lying point is the average results obtained from ten runs. The vertical lines represent the standard deviation

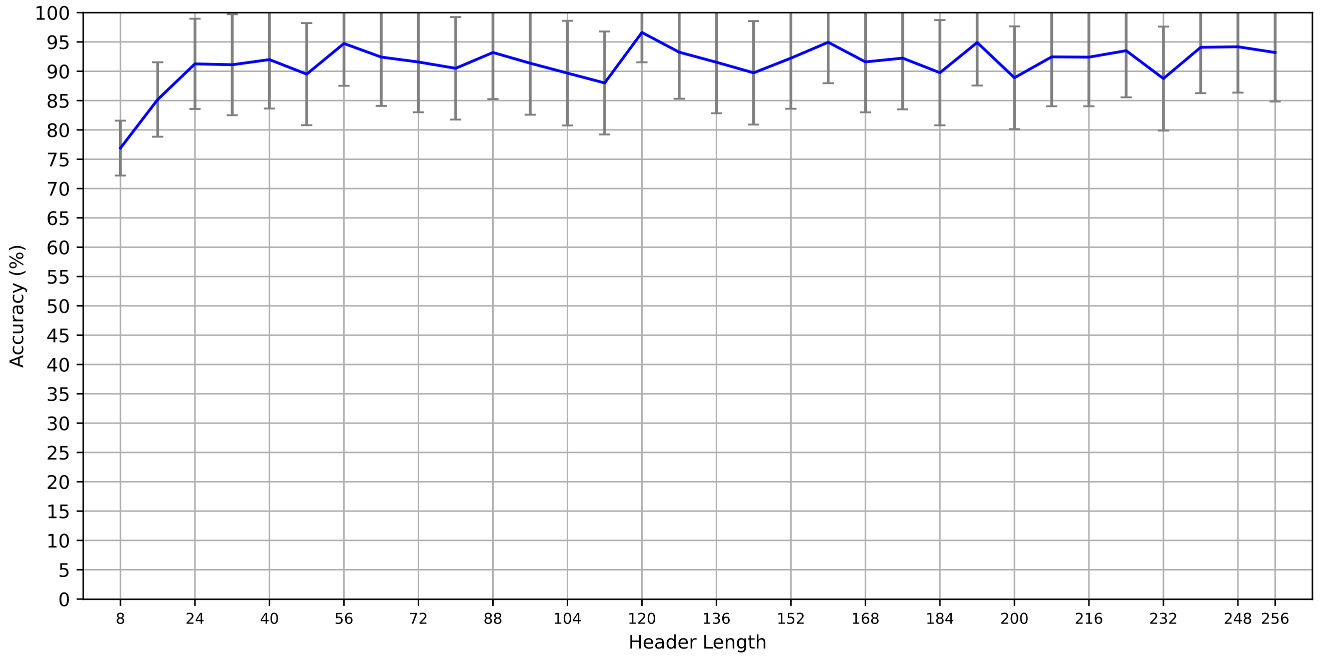


Figure A.70: Accuracy obtained by Entropy and DA of header as features for Random Forest on the Attack Dataset. Each lying point is the average results obtained from ten runs. The vertical lines represent the standard deviation

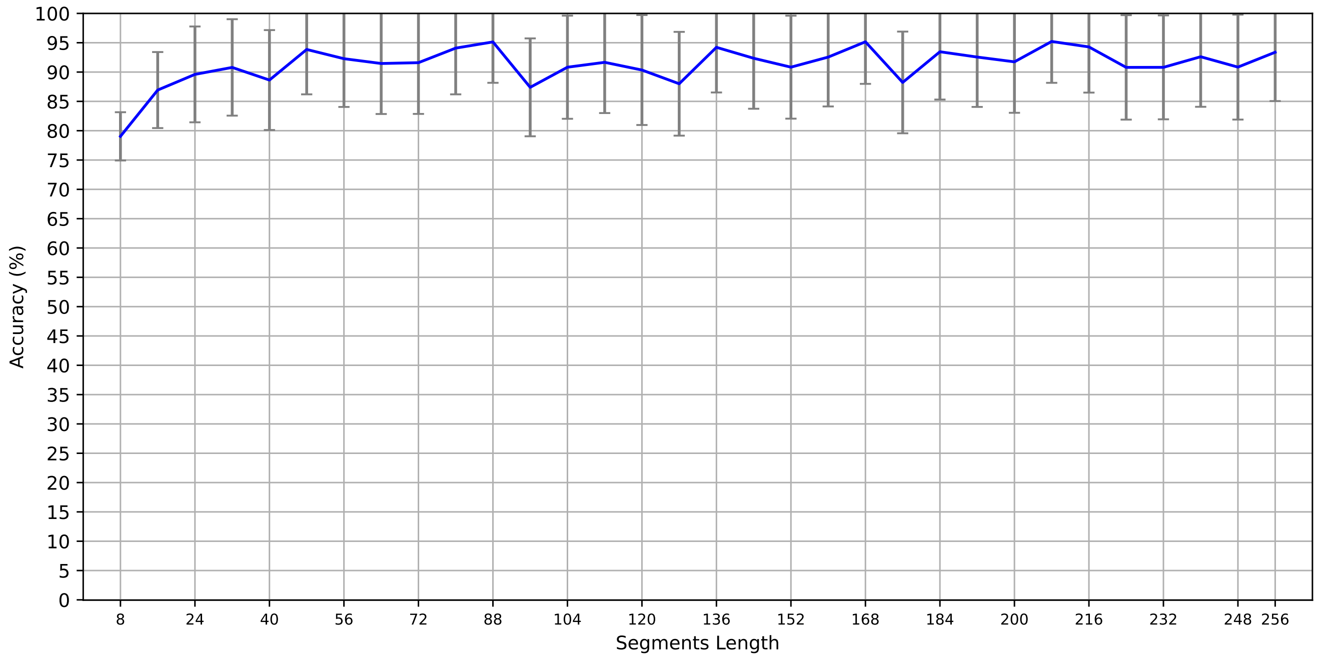


Figure A.71: Accuracy obtained by Entropy and DA of 2F as features for Random Forest on the Attack Dataset. Each lying point is the average results obtained from ten runs. The vertical lines represent the standard deviation

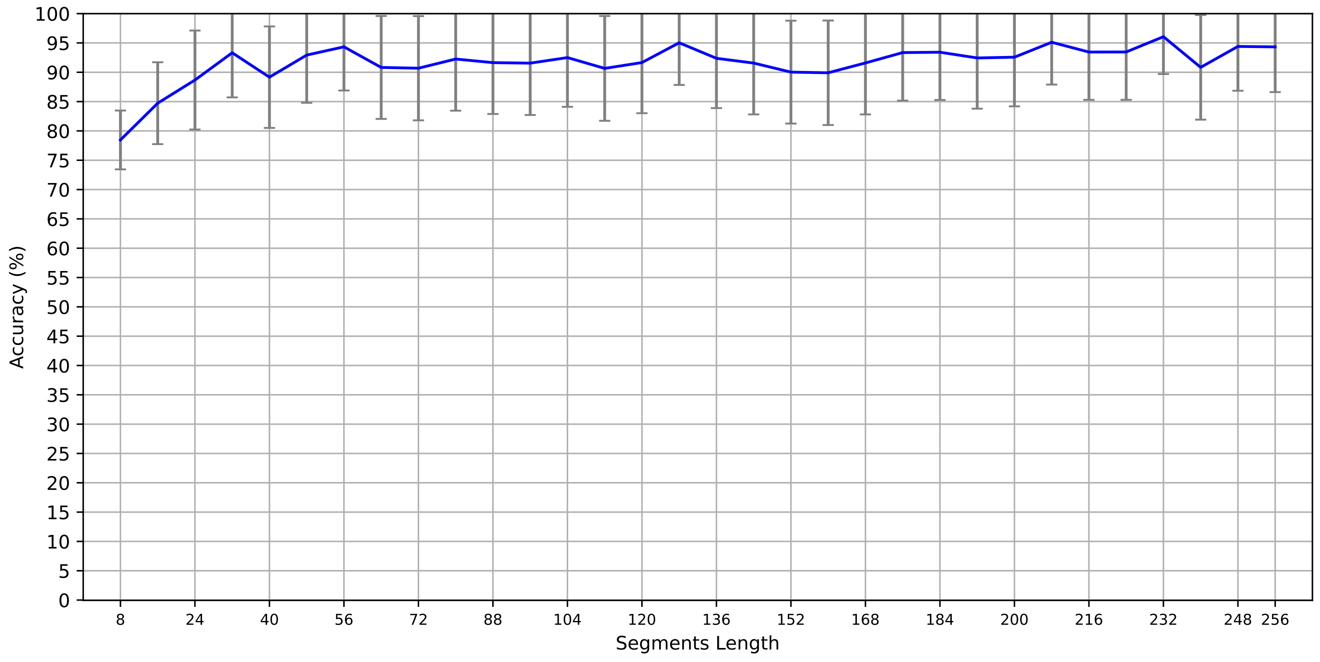


Figure A.72: Accuracy obtained by Entropy and DA of 3F as features for Random Forest on the Attack Dataset. Each lying point is the average results obtained from ten runs. The vertical lines represent the standard deviation

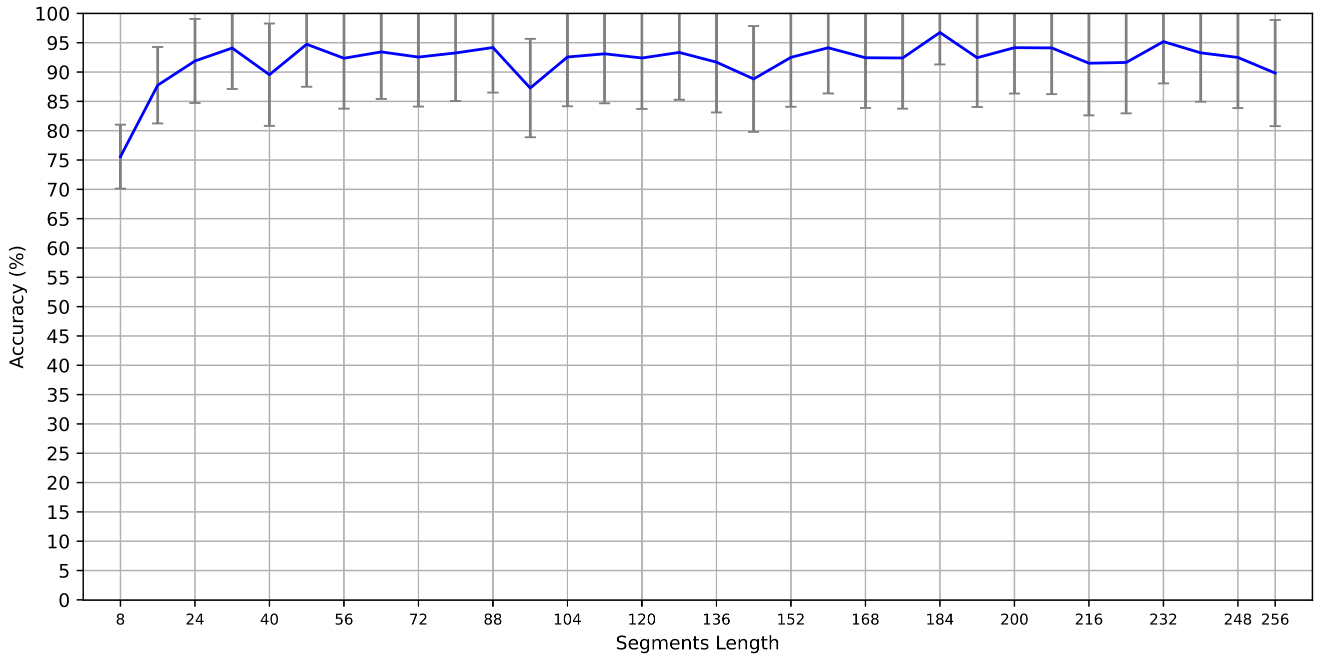


Figure A.73: Accuracy obtained by Entropy and DA of 4F as features for Random Forest on the Attack Dataset. Each lying point is the average results obtained from ten runs. The vertical lines represent the standard deviation

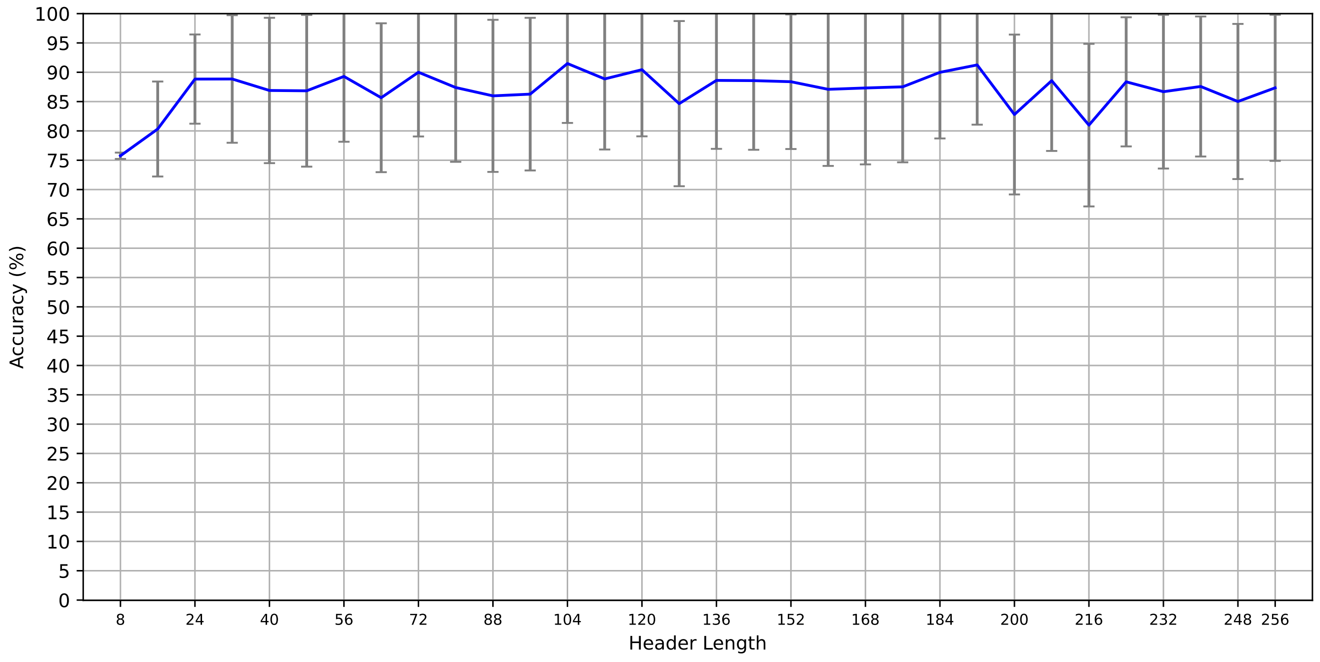


Figure A.74: Accuracy obtained by Entropy and DA of header as features for SVC on the NapierOne Dataset. Each lying point is the average results obtained from ten runs. The vertical lines represent the standard deviation

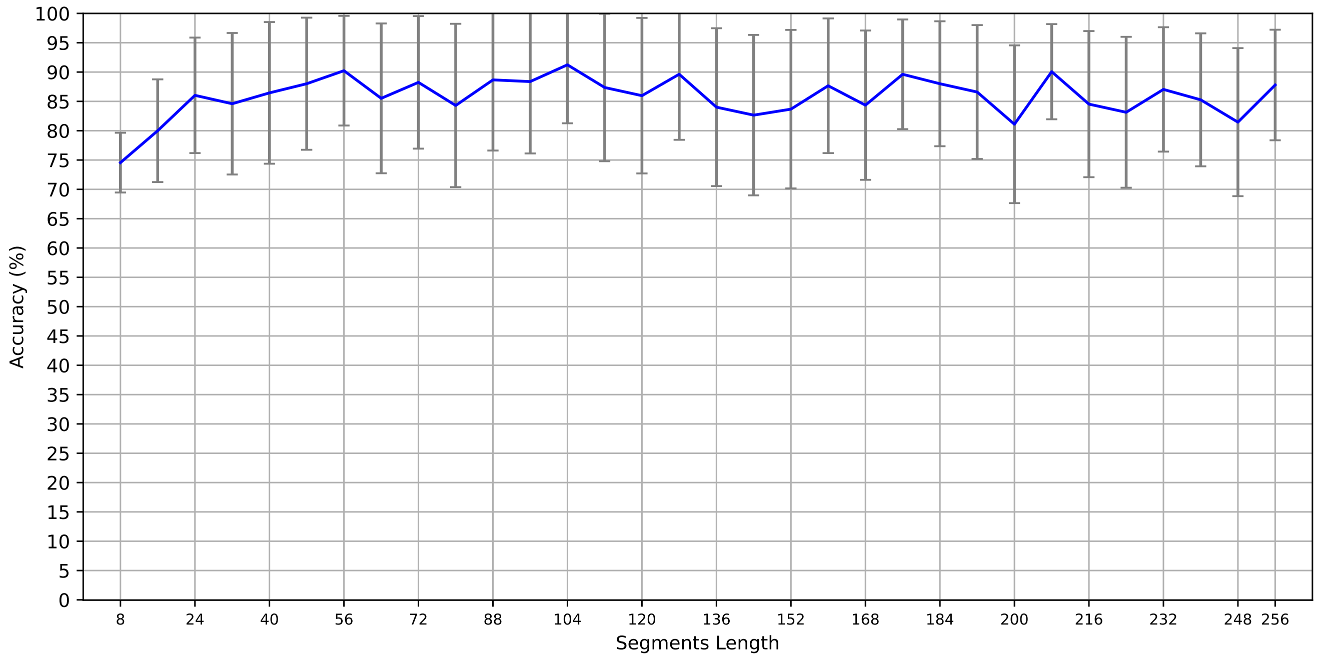


Figure A.75: Accuracy obtained by Entropy and DA of 2F as features for SVC on the NapierOne Dataset. Each lying point is the average results obtained from ten runs. The vertical lines represent the standard deviation

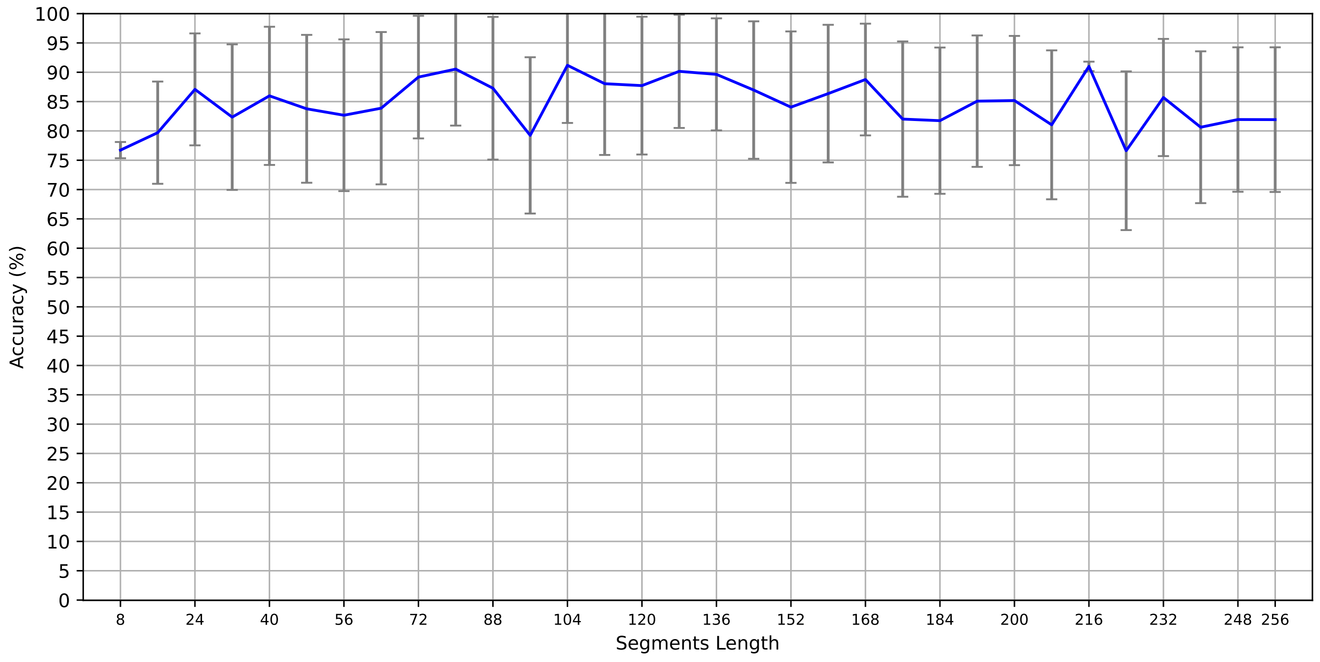


Figure A.76: Accuracy obtained by Entropy and DA of 3F as features for SVC on the NapierOne Dataset. Each lying point is the average results obtained from ten runs. The vertical lines represent the standard deviation

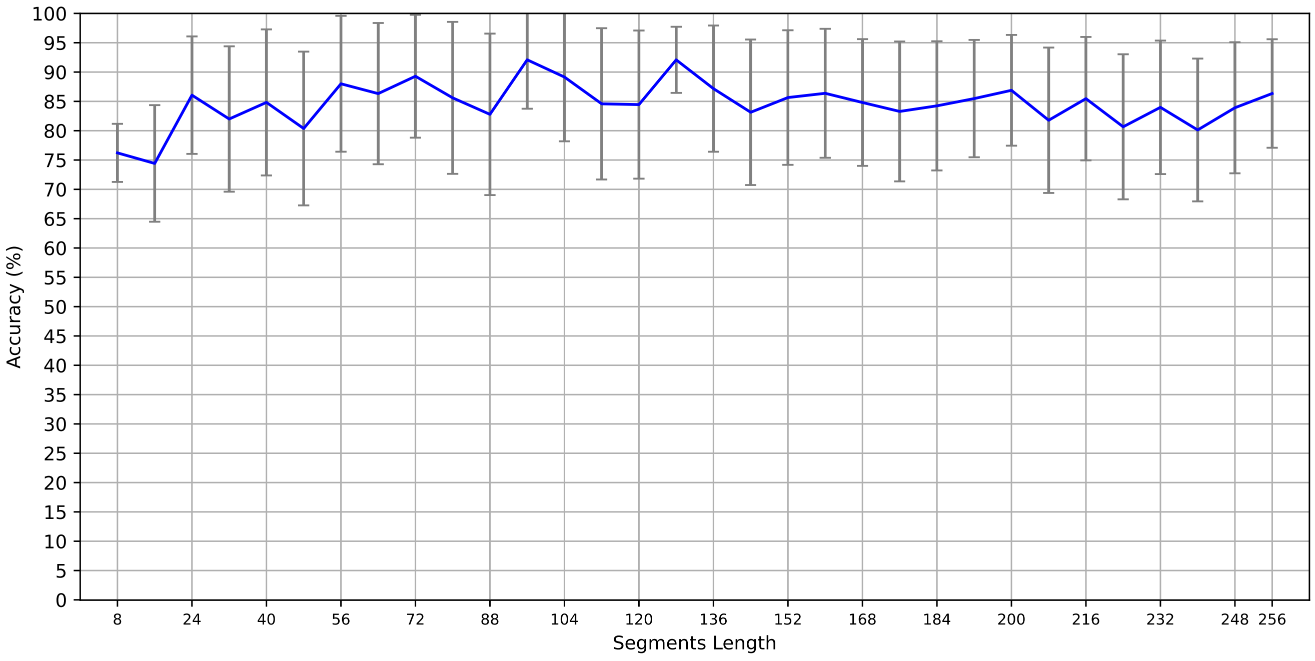


Figure A.77: Accuracy obtained by Entropy and DA of 4F as features for SVC on the NapierOne Dataset. Each lying point is the average results obtained from ten runs. The vertical lines represent the standard deviation

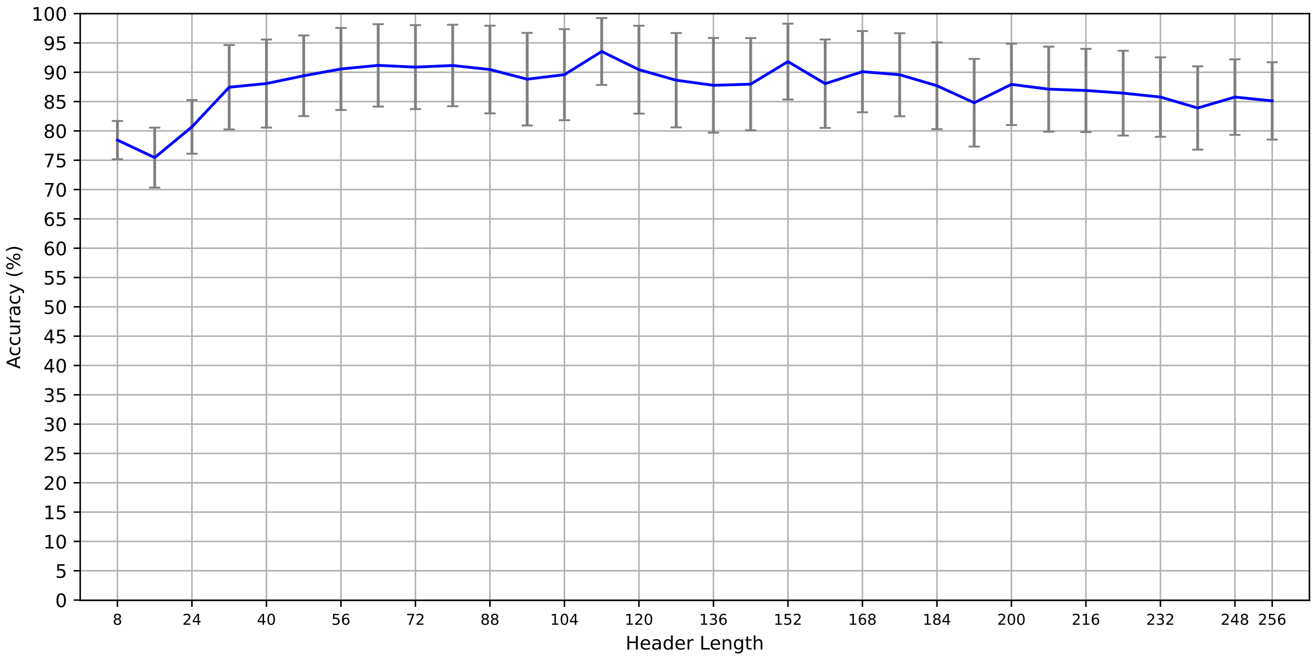


Figure A.78: Accuracy obtained by Entropy and DA of header as features for SVC on the Attack Dataset. Each lying point is the average results obtained from ten runs. The vertical lines represent the standard deviation

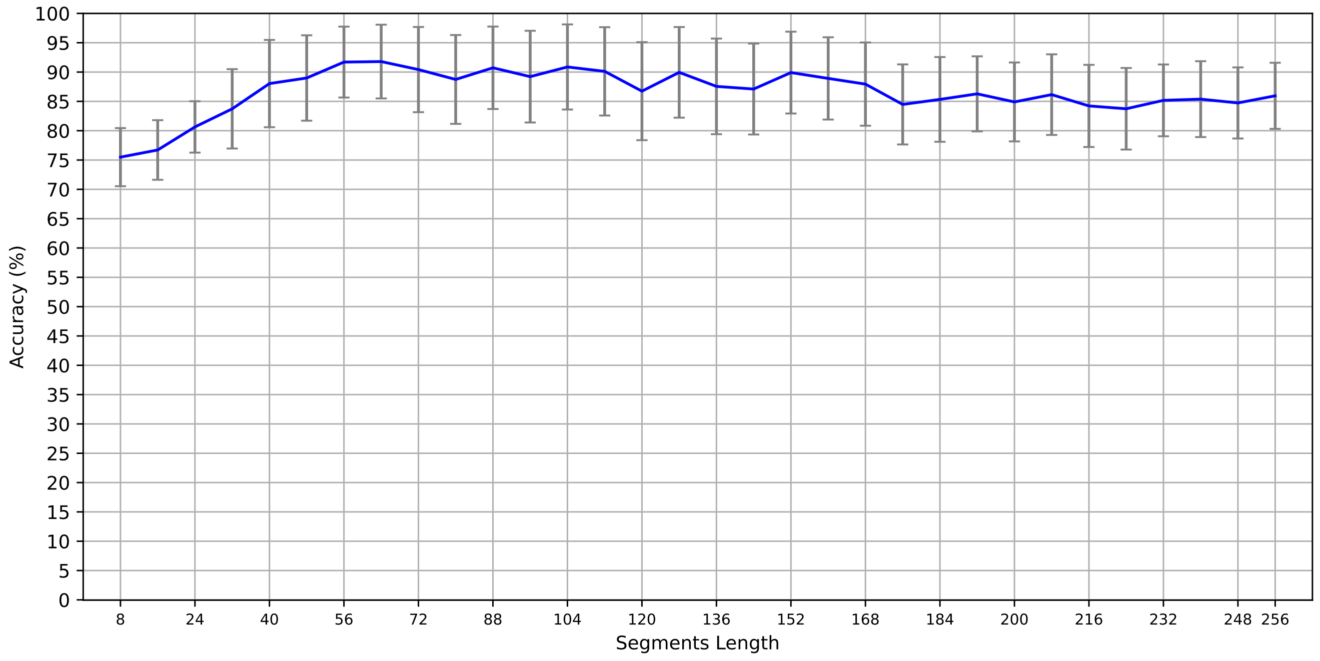


Figure A.79: Accuracy obtained by Entropy and DA of 2F as features for SVC on the Attack Dataset. Each lying point is the average results obtained from ten runs. The vertical lines represent the standard deviation

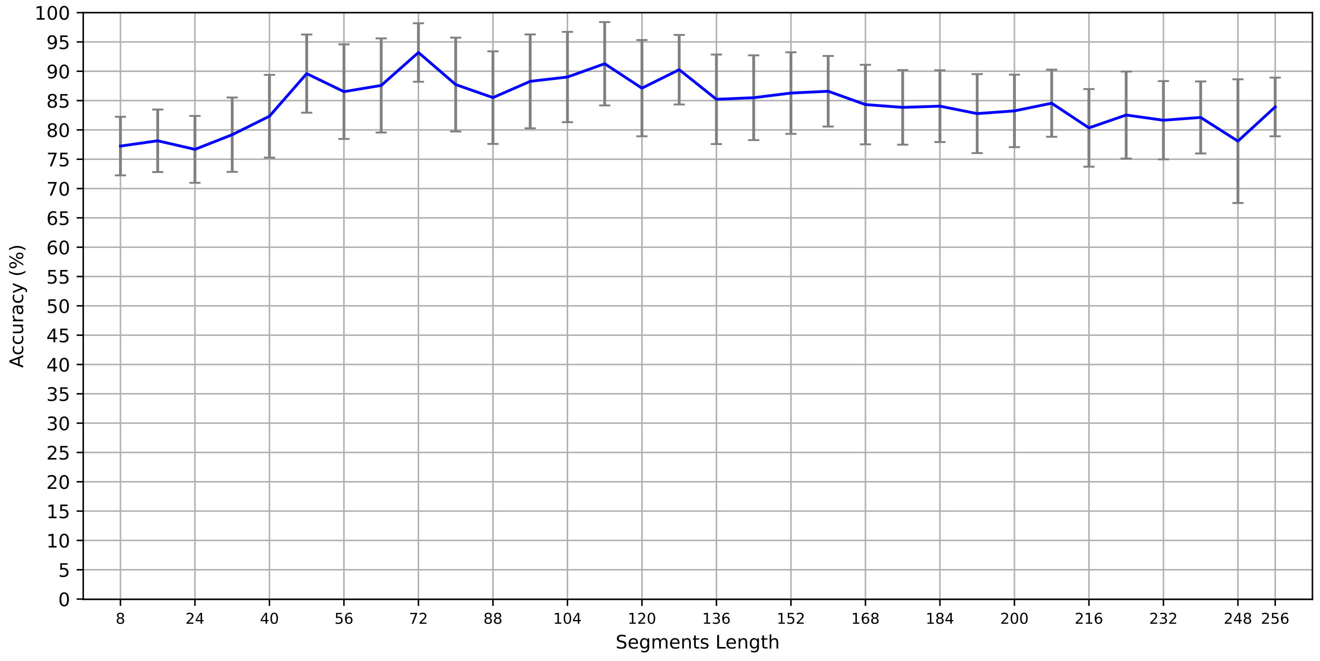


Figure A.80: Accuracy obtained by Entropy and DA of 3F as features for SVC on the Attack Dataset. Each lying point is the average results obtained from ten runs. The vertical lines represent the standard deviation

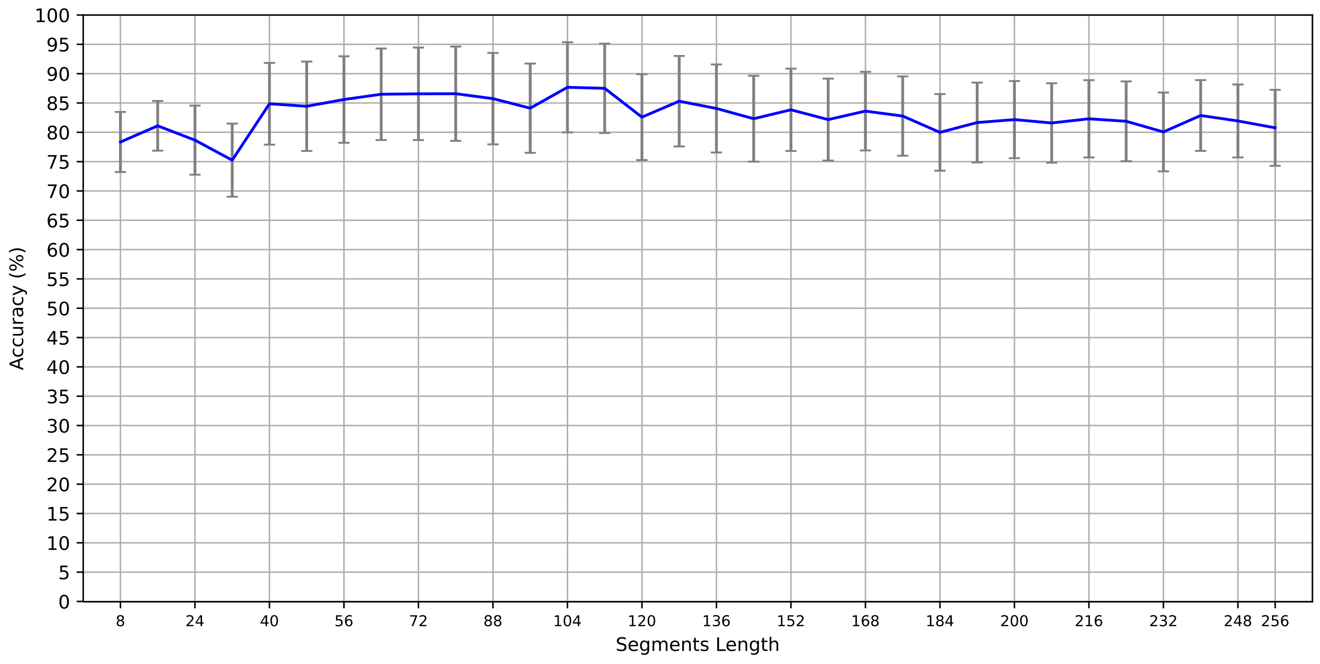


Figure A.81: Accuracy obtained by Entropy and DA of 4F as features for SVC on the Attack Dataset. Each lying point is the average results obtained from ten runs. The vertical lines represent the standard deviation

References

[1] S. R. Davies, R. Macfarlane, and W. J. Buchanan, “Differential area analysis for ransomware attack detection within mixed file datasets,” *Computers & Security*, vol. 108, p. 102377, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404821002017

[2] J. Lee, S.-Y. Lee, K. Yim, and K. Lee, “Neutralization method of ransomware detection technology using format preserving encryption,” *Sensors*, vol. 23, no. 10, 2023. [Online]. Available: https://www.mdpi.com/1424-8220/23/10/4728

[3] M. Venturini, F. Freda, E. Miotto, A. Giaretta, and M. Conti, “Differential area analysis for ransomware: Attacks, countermeasures, and limitations,” 2023.

[4] N. J. Nils, *Introduction to Machine Learning*. Nilsson J. Nils, Robotics Laboratory,Department of Computer,Science,Stanford University,Stanford, CA 94305, 1998.

[5] T. McIntosh, J. Jang-Jaccard, P. Watters, and T. Susnjak, “The inadequacy of entropy-based ransomware detection,” in *Neural Information Processing*, T. Gedeon, K. W. Wong, and M. Lee, Eds. Cham: Springer International Publishing, 2019, pp. 181–189.

[6] J. Lee and K. Lee, “A method for neutralizing entropy measurement-based ransomware detection technologies using encoding algorithms,” *Entropy*, vol. 24, no. 2, 2022. [Online]. Available: https://www.mdpi.com/1099-4300/24/2/239

[7] J. Lee, J. Yun, and K. Lee, “A study on countermeasures against neutralizing technology: Encoding algorithm-based ransomware detection methods using machine learning,” *Electronics*, vol. 13, no. 6, 2024. [Online]. Available: https://www.mdpi.com/2079-9292/13/6/1030

[8] J. Bang, J. N. Kim, and S. Lee, "Entropy sharing in ransomware: Bypassing entropy-based detection of cryptographic operations," *Sensors*, vol. 24, no. 5, 2024. [Online]. Available: https://www.mdpi.com/1424-8220/24/5/1446

[9] A. Gazet, "Comparative analysis of various ransomware virii," *Journal in Computer Virology*, vol. 6, no. 1, pp. 77–90, Feb 2010. [Online]. Available: https://doi.org/10.1007/s11416-008-0092-2

[10] P. O'Kane, S. Sezer, and D. Carlin, "Evolution of ransomware," *IET Networks*, vol. 7, no. 5, pp. 321–327, 2018. [Online]. Available: https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/iet-net.2017.0207

[11] A. Tandon and A. Nayyar, "A comprehensive survey on ransomware attack: A growing havoc cyberthreat," in *Data Management, Analytics and Innovation*, V. E. Balas, N. Sharma, and A. Chakrabarti, Eds. Singapore: Springer Singapore, 2019, pp. 403–420.

[12] A. Zimba and M. Chishimba, "On the economic impact of crypto-ransomware attacks: The state of the art on enterprise systems," *European Journal for Security Research*, vol. 4, no. 1, pp. 3–31, Apr 2019. [Online]. Available: https://doi.org/10.1007/s41125-019-00039-8

[13] J. M. Ehrenfeld, "Wannacry, cybersecurity and health information technology: A time to act," *Journal of Medical Systems*, vol. 41, no. 7, p. 104, May 2017. [Online]. Available: https://doi.org/10.1007/s10916-017-0752-1

[14] B. A. S. Al-rimy, M. A. Maarof, and S. Z. M. Shaid, "Ransomware threat success factors, taxonomy, and countermeasures: A survey and research directions," *Computers & Security*, vol. 74, pp. 144–166, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S016740481830004X

[15] I. V. Alekseev and V. V. Platonov, "Detection of encrypted executable files based on entropy analysis to determine the randomness measure of byte sequences," *Automatic Control and Computer Sciences*, vol. 51, no. 8, pp. 915–920, Dec 2017. [Online]. Available: https://doi.org/10.3103/S0146411617080041

[16] M. Egele, T. Scholte, E. Kirda, and C. Kruegel, "A survey on automated dynamic malware-analysis techniques and tools," *ACM Comput. Surv.*, vol. 44, no. 2, mar 2008. [Online]. Available: https://doi.org/10.1145/2089125.2089126

[17] T. R. McIntosh, J. Jang-Jaccard, and P. A. Watters, "Large scale behavioral analysis of ransomware attacks," in *Neural Information Processing*, L. Cheng, A. C. S. Leung, and S. Ozawa, Eds. Cham: Springer International Publishing, 2018, pp. 217–229.

[18] N. Scaife, H. Carter, P. Traynor, and K. R. B. Butler, "Cryptolock (and drop it): Stopping ransomware attacks on user data," in *2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS)*, 2016, pp. 303–312.

[19] A. Continella, A. Guagnelli, G. Zingaro, G. De Pasquale, A. Barenghi, S. Zanero, and F. Maggi, "Shieldfs: a self-healing, ransomware-aware filesystem," in *Proceedings of the 32nd Annual Conference on Computer Security Applications*, ser. ACSAC '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 336–347. [Online]. Available: https://doi.org/10.1145/2991079.2991110

[20] A. Kharraz, S. Arshad, C. Mulliner, W. Robertson, and E. Kirda, "Unveil: A large-scale, automated approach to detecting ransomware," 2016, Conference paper, p. 757 – 772, cited by: 316. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85064872000&partnerID=40&md5=2c3c2e37ec4c4271f5b628d0a4a062f2

[21] K. Lee, S.-Y. Lee, and K. Yim, "Effective ransomware detection using entropy estimation of files for cloud services," in *Pervasive Systems, Algorithms and Networks*, C. Esposito, J. Hong, and K.-K. R. Choo, Eds. Cham: Springer International Publishing, 2019, pp. 133–139.

[22] ——, "Machine learning based file entropy analysis for ransomware detection in backup systems," *IEEE Access*, vol. 7, pp. 110 205–110 215, 2019.

[23] F. Tang, B. Ma, J. Li, F. Zhang, J. Su, and J. Ma, "Ransomspector: An introspection-based approach to detect crypto ransomware," *Computers Security*, vol. 97, p. 101997, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404820302704

[24] C.-M. Hsu, C.-C. Yang, H.-H. Cheng, P. E. Setiasabda, and J.-S. Leu, "Enhancing file entropy analysis to improve machine learning detection rate of ransomware," *IEEE Access*, vol. 9, pp. 138 345–138 351, 2021.

[25] T. Kim and E. Im, "Hybrid framework for ransomware protection using effective decoy deployment and file traversal entropy check," *Available at SSRN 4496374*.

[26] M. A. Ayub, A. Siraj, B. Filar, and M. Gupta, "Rwarmor: a static-informed dynamic analysis approach for early detection of cryptographic windows ransomware," *International Journal of Information Security*, vol. 23, no. 1, pp. 533–556, Feb 2024. [Online]. Available: https://doi.org/10.1007/s10207-023-00758-z

[27] C. J. W. Chew, V. Kumar, P. Patros, and R. Malik, "Real-time system call-based ransomware detection," *International Journal of Information Security*, vol. 23, no. 3, pp. 1839–1858, Jun 2024. [Online]. Available: https://doi.org/10.1007/s10207-024-00819-x

[28] P. G. A. Hall and W. P. Davis, "Sliding window measurement for file type identification," 2007. [Online]. Available: https://api.semanticscholar.org/CorpusID:14149550

[29] S. Jung and Y. Won, "Ransomware detection method based on context-aware entropy analysis," *Soft Computing*, vol. 22, no. 20, pp. 6731–6740, Oct 2018. [Online]. Available: https://doi.org/10.1007/s00500-018-3257-z

[30] B. Zhao, Q. Liu, and X. Liu, "Evaluation of encrypted data identification methods based on randomness test," in *2011 IEEE/ACM International Conference on Green Computing and Communications*, 2011, pp. 200–205.

[31] S. K. Shaukat and V. J. Ribeiro, "Ransomwall: A layered defense system against cryptographic ransomware attacks using machine learning," in *2018 10th International Conference on Communication Systems & Networks (COMSNETS)*, 2018, pp. 356–363.

[32] G. Y. Kim, J.-Y. Paik, Y. Kim, and E.-S. Cho, "Byte frequency based indicators for crypto-ransomware detection from empirical analysis," *Journal of Computer Science and Technology*, vol. 37, no. 2, pp. 423–442, Apr 2022. [Online]. Available: https://doi.org/10.1007/s11390-021-0263-x

[33] S. R. Davies, R. Macfarlane, and W. J. Buchanan, "Comparison of entropy calculation methods for ransomware encrypted file identification," *Entropy*, vol. 24, no. 10, 2022. [Online]. Available: https://www.mdpi.com/1099-4300/24/10/1503

[34] C. E. Shannon, "A mathematical theory of communication," *The Bell System Technical Journal*, vol. 27, no. 3, pp. 379–423, 1948.

[35] M. McDaniel and M. Heydari, "Content based file type detection algorithms," in *36th Annual Hawaii International Conference on System Sciences, 2003. Proceedings of the*, 2003, pp. 10 pp.–.

[36] S. Razaulla, C. Fachkha, C. Markarian, A. Gawanmeh, W. Mansoor, B. C. M. Fung, and C. Assi, "The age of ransomware: A survey on the evolution, taxonomy, and research directions," *IEEE Access*, vol. 11, pp. 40 698–40 723, 2023.

[37] F. Noorbehbahani, F. Rasouli, and M. Saberi, "Analysis of machine learning techniques for ransomware detection," in *2019 16th International ISC (Iranian Society of Cryptology) Conference on Information Security and Cryptology (ISCISC)*, 2019, pp. 128–133.

[38] J. Ispahany, M. R. Islam, M. Z. Islam, and M. A. Khan, "Ransomware detection using machine learning: A review, research limitations and future directions," *IEEE Access*, vol. 12, pp. 68 785–68 813, 2024.

[39] S. Poudyal, K. P. Subedi, and D. Dasgupta, "A framework for analyzing ransomware using machine learning," in *2018 IEEE Symposium Series on Computational Intelligence (SSCI)*, 2018, pp. 1692–1699.

[40] G. Cusack, O. Michel, and E. Keller, "Machine learning-based detection of ransomware using sdn," in *Proceedings of the 2018 ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization*, ser. SDN-NFV Sec'18. New York, NY, USA: Association for Computing Machinery, 2018, p. 1–6. [Online]. Available: https://doi.org/10.1145/3180465.3180467

[41] S. I. Bae, G. B. Lee, and E. G. Im, "Ransomware detection using machine learning algorithms," *Concurrency and Computation: Practice and Experience*, vol. 32, no. 18, p. e5422, 2020, e5422 cpe.5422. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.5422

[42] W.-J. Li, K. Wang, S. Stolfo, and B. Herzog, "Fileprints: identifying file types by n-gram analysis," in *Proceedings from the Sixth Annual IEEE SMC Information Assurance Workshop*, 2005, pp. 64–71.

[43] K. Atkinson, *An introduction to numerical analysis*. John wiley & sons, 1991.

[44] D. J. William, *Undergraduate Fundamentals of Machine Learning*. Deuschle J. William, CS181 staff, Harvard, 2017.

[45] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, Oct 2001. [Online]. Available: https://doi.org/10.1023/A:1010933404324

[46] S. R. Davies, R. Macfarlane, and W. J. Buchanan, "Napierone: A modern mixed file data setÂ alternative to govdocs1," *Forensic Science International: Digital Investigation*, vol. 40, p. 301330, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2666281721002560

[47] J. Katz and Y. Lindell, *Introduction to Modern Cryptography*. CRC PRESS, 2007.

Acknowledgments

I would like to express my deepest gratitude to Professor Mauro Conti and PhD. Alberto Giaretta for their invaluable assistance with my master thesis. In particular, without the support of PhD. Giaretta and the University of Örebro, none of this would have been possible. They provided me with the opportunity to access the ORU-DGX data center, essential for testing my algorithms and collecting the necessary data. Moreover, they guided me through every step of this journey, from configuring the tools to providing critical insights on machine learning algorithms and much more. I am profoundly thankful for their unwavering support and encouragement.

I extend my heartfelt thanks to my mother, Annamaria, and my two sisters, Carlotta and Camilla. Your endless support throughout my bachelor and master degrees has been a pillar of strength for me, and I will be forever grateful for your unconditioned love and encouragement.

Lastly, I wish to thank all my colleagues and friends, both new and old, whom I have met along this journey. Your companionship has made this experience truly unforgettable, and your support has been invaluable. Thank you for making this chapter of my life so memorable.