**DEEP AND PARALLEL NETWORK IN NETWORK MODELS FOR ACCURATE ENCRYPTED NETWORK TRAFFIC CLASSIFICATION**

**ABSTRACT**

In the contemporary digital landscape, the exponential growth of network traffic has led to an increased need for efficient and secure network management. As the volume of data transmitted over networks continues to rise, so does the risk of unauthorized access, data breaches, and cyber threats. Network administrators and security professionals strive to implement robust solutions to safeguard sensitive information and ensure the integrity of network communications. The history of network traffic classification is closely tied to the evolution of communication networks. Traditional networks initially focused on providing basic connectivity, with little emphasis on security mechanisms. As digital communication became more prevalent, the need for identifying and classifying network traffic emerged to address security concerns. Early methods relied on simple rule-based systems and Port based methods categorize traffic based on the port numbers associated with specific applications. However, these methods are becoming less effective as applications increasingly use dynamic or non-standard ports. Signature-based approaches involve identifying known patterns or signatures associated with malicious traffic, but this is limited to known threats and struggles with zero-day attacks. The increasing sophistication of cyber threats and the prevalence of encrypted communication pose significant challenges for traditional network traffic classification systems. Encrypted traffic, while essential for protecting data privacy, also provides a cover for malicious activities. The pervasive use of encryption in modern communication protocols, such as HTTPS, presents a pressing need for advanced network traffic classification techniques. The inability to inspect encrypted traffic hampers the ability to detect and prevent malicious activities effectively. To address the challenges posed by encrypted network traffic, deep learning techniques, specifically deep and parallel Network-In-Network (NIN) models, have gained prominence. These models leverage neural networks to extract intricate features from encrypted packets, enabling the classification of traffic based on behavior. The parallelization of NIN models enhances computational efficiency, making them well-suited for real-time network traffic analysis.

**CHAPTER 1**

**INTRODUCTION**

**1.1 History**

In the rapidly evolving digital landscape, the surge in network traffic has necessitated the development of effective and secure network management solutions. As data transmission volumes continue to soar, the associated risks of unauthorized access, data breaches, and cyber threats have escalated. Network administrators and security professionals are confronted with the challenge of implementing robust measures to safeguard sensitive information and maintain the integrity of network communications. The history of network traffic classification is intricately linked to the evolution of communication networks. In the early stages, traditional networks primarily focused on providing basic connectivity, often neglecting the incorporation of robust security mechanisms. As digital communication became ubiquitous, the imperative to identify and classify network traffic emerged as a crucial aspect of addressing security concerns. Initial methods relied on simplistic rule-based systems and port-based approaches that categorized traffic based on specific application-associated port numbers. However, the effectiveness of these methods dwindled as applications increasingly began to utilize dynamic or non-standard ports. Signature-based approaches were subsequently introduced, involving the identification of known patterns or signatures associated with malicious traffic. Nevertheless, this method was constrained by its limitation to known threats and its susceptibility to zero-day attacks. The evolving landscape of cyber threats, coupled with the widespread adoption of encrypted communication, presents significant challenges for traditional network traffic classification systems. While encrypted traffic is indispensable for preserving data privacy, it concurrently serves as a veil for potentially malicious activities. The pervasive use of encryption in contemporary communication protocols, such as HTTPS, has underscored the urgent necessity for advanced network traffic classification techniques. To overcome the hurdles posed by encrypted network traffic, deep learning techniques have emerged as a pivotal solution, with a specific focus on deep and parallel Network-In-Network (NIN) models. These models harness the power of neural networks to extract intricate features from encrypted packets, enabling the classification of traffic based on behavioral patterns. The parallelization of NIN models further enhances computational efficiency, rendering them particularly well-suited for real-time network traffic analysis. This evolution reflects a proactive response to the escalating challenges posed by the dynamic nature of modern network communication and the imperative to ensure the security and integrity of digital data transmission.

* 1. **Problem statement**

In the current digital landscape, the surge in network traffic volume has given rise to a complex challenge demanding efficient and secure network management solutions. The risks associated with unauthorized access, data breaches, and cyber threats have escalated, necessitating robust measures to protect sensitive information and uphold the integrity of network communications. The historical evolution of network traffic classification underscores several challenges. Conventional port-based approaches face diminished effectiveness due to the dynamic and non-standard port usage by modern applications. Signature-based methods, while proficient against known threats, struggle with zero-day attacks and lack adaptability to emerging malicious patterns. The widespread adoption of encryption, notably in protocols like HTTPS, presents a formidable obstacle for traditional traffic classification systems. While encryption safeguards data privacy, it simultaneously conceals potential malicious activities, making it challenging to inspect and analyze encrypted traffic effectively.

The overarching problem lies in the inability of existing classification systems to adapt to these challenges. Encrypted traffic remains elusive to inspection, hindering the detection and prevention of malicious activities. To address this, there is a critical need for advanced network traffic classification techniques. This entails a shift towards innovative solutions employing deep learning, with a specific focus on deep and parallel Network-In-Network (NIN) models. These models leverage neural networks to extract intricate features from encrypted packets, enabling accurate classification based on behavioral patterns. Moreover, the parallelization of NIN models enhances computational efficiency, making them well-suited for real-time analysis. The objective is to overcome the limitations of current methods, particularly in the context of encrypted and dynamic network communication, and ensure the effective detection and mitigation of security risks.

* 1. **Research Motivation**

The motivation behind this research stems from the critical imperative to address the escalating challenges posed by the dynamic landscape of contemporary network communication. As the volume of network traffic continues to surge, the conventional methods of traffic classification are facing increasing limitations, particularly in the context of dynamic application port usage and the pervasive adoption of encryption protocols. The need for robust security measures has become more pressing than ever, given the heightened risks of unauthorized access, data breaches, and sophisticated cyber threats. The motivation also arises from the realization that traditional traffic classification systems, relying on rule-based, port-based, or signature-based approaches, are struggling to adapt to the evolving nature of cyber threats. The prevalence of encrypted communication, while essential for data privacy, introduces a significant hurdle in inspecting and analyzing traffic effectively. This research aims to provide a timely and innovative response to these challenges by exploring the potential of deep learning techniques, specifically deep and parallel Network-In-Network (NIN) models. By leveraging the power of neural networks to decipher intricate features from encrypted packets, the research seeks to advance the field of network traffic classification, enhance security measures, and contribute valuable insights to the ongoing discourse on securing digital communication in our interconnected and dynamic digital landscape.

* 1. **Applications**

The applications of the proposed research findings are broad and impactful across various domains of network security and communication. Firstly, the developed deep and parallel Network-In-Network (NIN) models for encrypted network traffic classification can significantly enhance the effectiveness of intrusion detection systems. By accurately identifying and categorizing encrypted traffic based on behavioral patterns, these models empower network administrators and security professionals to swiftly detect and mitigate potential threats, safeguarding the integrity of critical systems. Additionally, the research outcomes hold relevance for optimizing Quality of Service (QoS) in network environments, ensuring that different types of traffic receive appropriate priority and resource allocation. Furthermore, the findings can contribute to refining content filtering mechanisms, allowing for more nuanced and precise control over network access. Beyond security, the applications extend to bandwidth management, where the ability to classify encrypted traffic with high accuracy facilitates efficient allocation and utilization of network resources. Overall, the research outcomes offer versatile applications in enhancing the security, performance, and management of networked systems in the face of escalating challenges posed by encrypted communication and dynamic network landscapes.

**CHAPTER 2**

**LITERATURE SURVEY**

In this section, we compare recently published research relevant to our work. Conti et al. [[**8**](https://www.mdpi.com/1424-8220/22/19/7643#B8-sensors-22-07643)] developed a framework to infer user actions executed on mobile apps based on packet sizes and their order information. Park and Kim [[**9**](https://www.mdpi.com/1424-8220/22/19/7643#B9-sensors-22-07643)] target KakaoTalk, a mobile instant messaging service, and proposed a framework to infer user activities by passive analysis of network traffic. Saltaformaggio et al. [**10**] proposed NetScope, a tool to identify user activities generated by mobile apps, based on the statistics originating from the Internet Protocol (IP) headers. The AppScanner [**11**] framework was implemented for the real-time identification of Android apps from encrypted network traffic. All these methods employ classical ML algorithms such as k-nearest neighbour and random forest. However, their performance heavily depends on human-generated features, which is significantly time-consuming and limited in generalizability.

DL obviates the need to perform feature selections by a domain expert and it has a higher capacity to learn highly complicated patterns compared to traditional ML methods. Recent work has demonstrated the efficacy of DL methods to perform traffic classification. In [[**12**](https://www.mdpi.com/1424-8220/22/19/7643#B12-sensors-22-07643)], a framework named ‘Deep Packet’ was presented that achieved both traffic characterization, where the network traffic was categorized into major classes (e.g., FTP and P2P), and application identification (e.g., BitTorrent and Skype). Deep Packet framework embedded Stacked Autoencoder (SAE) and One Dimensional Convolutional Neural Networks (1D CNN) to classify network traffic. Experiments were conducted using the ISCX public dataset [[**13**](https://www.mdpi.com/1424-8220/22/19/7643#B13-sensors-22-07643)] and the model achieved a recall rate of 0.98 in the application identification task and 0.94 in the traffic categorization task.

Recurrent Neural Networks (RNN) and CNN were applied to perform application-level identification in [[**14**](https://www.mdpi.com/1424-8220/22/19/7643#B14-sensors-22-07643)]. Their DL model used packet-level features such as ports, payload bytes, TCP window size, inter-arrival times of packets, and packet direction. This combination of the RNN and CNN model attained an accuracy of 96%.

In [[**5**](https://www.mdpi.com/1424-8220/22/19/7643#B5-sensors-22-07643)], an end-to-end encrypted traffic classification model with 1D CNN was proposed. Feature extraction, feature selection, and classifiers were integrated into a single framework in the model, which automatically learned the non-linear relationship between the raw input and expected output. This model was validated with the ISCX public dataset and showed a significant improvement over the C4.5 ML method.

In [15], DataNet was introduced as an application-aware framework for smart home networks. It was developed and evaluated using three deep learning-based approaches, namely multilayer perceptron, stacked autoencoder, and CNN. Experiments were conducted using the ISCX public dataset with encrypted data samples from 15 applications. The experimental results showed that recall, precision, and f1 score were all greater than 92%.

A Server Name Identification (SNI) classification technique was proposed in [16], using HTTPS features (packet sizes, payload sizes, inter-arrival times, and direction). The model consisted of a combination of RNN and CNN and obtained an accuracy of 82.3%.

Aceto et al. proposed two frameworks named MIMETIC [[**17**](https://www.mdpi.com/1424-8220/22/19/7643#B17-sensors-22-07643)] and DISTILLER [[**18**](https://www.mdpi.com/1424-8220/22/19/7643#B18-sensors-22-07643)]. MIMETIC takes two inputs namely data payload and protocol/time series features. The DL model captures patterns in both input viewpoints to perform traffic classification. Multitask and multimodal DL is adapted to devise the DISTILLER to perform mobile app classification. In [[**19**](https://www.mdpi.com/1424-8220/22/19/7643#B19-sensors-22-07643)], FS-Net an end-to-end model was proposed that learns features from the raw flow sequences and makes classification to identify flows. FS-Net achieves a True Positive Rate of 99.14% in identifying 18 applications.

In [[**20**](https://www.mdpi.com/1424-8220/22/19/7643#B20-sensors-22-07643)], a traffic classification method based on CNN was proposed. This method used the NetFlow and packet-based features to identify Quick UDP Internet Connection (QUIC) protocol-based services such as Google Hangout chat, Google Hangout voice call, YouTube, File transfer, and Google Play music. The experiments demonstrated that this method could detect five kinds of QUIC-based services with an accuracy of approximately 99%. This work uses NetFlow-based features that lead to an increase in the runtime of processing and classification. Using all packets in the traffic flows is another disadvantage in this work, as this causes obstacles when the number of packets in flows is large.

In [[**21**](https://www.mdpi.com/1424-8220/22/19/7643#B21-sensors-22-07643)], a framework called ‘ActiveTracker’ was proposed to recognize app trajectory over encrypted Internet traffic streams. Experiments were conducted based on real-world encrypted mobile traffic as well as synthetic traffic. The proposed DNN-based classification model which consisted of an app filter and an activity classifier achieved up to 79.65% in recognizing app trajectory from a long traffic stream.

**CHAPTER 3**

**EXISTING SYSTEM**

In the rapidly evolving digital landscape, the escalating volume of network traffic has precipitated an urgent demand for robust and secure network management solutions. The proliferation of data transmission over networks not only amplifies connectivity but also heightens the susceptibility to unauthorized access, data breaches, and cyber threats. Consequently, network administrators and security professionals are ardently seeking efficient measures to protect sensitive information and uphold the integrity of network communications.

The history of network traffic classification closely aligns with the evolution of communication networks. In the nascent stages, traditional networks primarily focused on providing basic connectivity, with minimal attention to security mechanisms. However, as digital communication became ubiquitous, the imperative to identify and classify network traffic surfaced to address emerging security concerns. Early methods relied on rudimentary rule-based systems and port-based approaches, categorizing traffic based on specific application-associated port numbers. Nevertheless, the efficacy of these methods diminishes as applications increasingly adopt dynamic or non-standard ports.

Signature-based approaches entered the scene, involving the identification of known patterns or signatures associated with malicious traffic. Yet, this method is constrained by its reliance on known threats and grapples with the challenges posed by zero-day attacks. The growing sophistication of cyber threats, coupled with the prevalence of encrypted communication, poses formidable challenges for conventional network traffic classification systems. While encryption is indispensable for safeguarding data privacy, it concurrently provides a cloak for potentially malicious activities. The widespread adoption of encryption in modern communication protocols, exemplified by HTTPS, underscores the critical need for advanced network traffic classification techniques.

The inability to inspect encrypted traffic impedes the efficacy of detecting and preventing malicious activities effectively. To surmount these challenges, deep learning techniques, particularly deep and parallel Network-In-Network (NIN) models, have emerged as a noteworthy solution. These models harness neural networks to extract intricate features from encrypted packets, enabling the classification of traffic based on behavioral characteristics. The parallelization of NIN models augments computational efficiency, rendering them well-suited for real-time network traffic analysis. This existing system provides a foundation for understanding the historical context and the imperative for adopting advanced techniques in encrypted network traffic classification.

**Disadvantages**

Convolutional Neural Networks (CNNs) have proven to be powerful tools in various machine learning tasks, particularly in image and video analysis. However, like any technology, CNNs have certain drawbacks and limitations:

**Large Training Data Requirements:** CNNs often require a substantial amount of labeled training data to learn meaningful and generalizable features. In scenarios where acquiring a large and diverse dataset is challenging, CNNs may struggle to achieve optimal performance.

**Computational Intensity:** Training deep CNNs can be computationally intensive, demanding substantial processing power and memory resources. This can limit their practicality for deployment on resource-constrained devices or in real-time applications.

**Lack of Interpretability:** CNNs are often regarded as "black-box" models due to the complexity of their architectures. Understanding how they arrive at specific decisions or classifications can be challenging, limiting the interpretability of their results.

**Vulnerability to Adversarial Attacks:** CNNs are susceptible to adversarial attacks, where small, carefully crafted perturbations to input data can lead to misclassification. This vulnerability raises concerns about the robustness of CNNs in security-critical applications.

**Overfitting:** Deep CNNs, especially those with a large number of parameters, are prone to overfitting, meaning they may memorize training data instead of learning general patterns. Regularization techniques are often required to mitigate this issue.

**Limited Contextual Understanding:** CNNs process data in a local and fixed-size receptive field, which may limit their ability to capture long-range dependencies or understand global context in some scenarios, such as natural language processing tasks.

**Translation Invariance:** While CNNs exhibit translation invariance, meaning they can recognize patterns regardless of their position in the input space, they might not always capture spatial hierarchies effectively, impacting their ability to understand hierarchical structures in data.

**Data Augmentation Challenges:** Augmenting data to increase the diversity of the training set can be challenging, particularly in certain domains where transformations may not preserve semantic meaning or where annotated data is scarce.

**Limited Handling of Sequential Data**: While CNNs are effective for grid-like data structures (images), they may not be inherently designed for sequential data, requiring additional architectures like recurrent neural networks (RNNs) for tasks such as natural language processing.

**CHAPTER 4**

**PROPOSED SYSTEM**

**4.1 Overview**

This research represents a GUI-based application for encrypted network traffic classification using deep learning models from ISCX VPN-NonVPN dataset. In addition, the GUI interface makes it accessible to users without extensive programming knowledge. Further, the proposed model serves as a practical tool for network traffic analysis, allowing users to leverage advanced machine learning techniques for encrypted traffic classification. The user-friendly interface and detailed evaluation metrics make it valuable for both educational purposes and real-world applications in network security and management. Moreover, network administrators or analysts can use this tool to categorize encrypted network traffic, which is crucial for various security and optimization tasks.

1. Objective

The primary goal of this project is to classify encrypted network traffic into different categories using advanced machine learning techniques, specifically deep learning models. The application focuses on two main models: a Standard Convolutional Neural Network (CNN) and a Parallel Deep Network-in-Network (NIN) model.

2. User Interface

* The project provides a Graphical User Interface (GUI) implemented using tkinter, a Python library for creating GUI applications.
* Users can interact with the application through buttons and visualizations displayed on the GUI.

3. Functionality

* Dataset Upload and Preprocessing
  + Users can upload a dataset containing encrypted network traffic data.
  + The application preprocesses the dataset, which includes handling missing values, encoding non-numeric data, and normalizing the features.
  + Basic statistics and visualizations (such as bar charts) about the dataset are displayed to the user.
* Model Training
  + The application allows users to train two types of models: Standard CNN and Parallel Deep NIN model.
  + The Standard CNN is a traditional deep learning model for image classification.
  + The Parallel Deep NIN model is a more complex architecture designed for capturing intricate patterns in the data.
* Model Evaluation
  + After training, the models are evaluated using metrics such as accuracy, precision, recall, and F1-score.
  + Confusion matrices and graphical comparisons of model performance are provided.
* Traffic Classification
  + Users can upload encrypted test data.
  + The trained models classify the encrypted network traffic and display the results, including the predicted network category.

4. Models Used:

* Standard CNN:
  + A traditional Convolutional Neural Network used for image classification.
  + Consists of convolutional layers, max-pooling layers, flattening layers, and dense (fully connected) layers.
* Parallel Deep NIN Model:
  + A more complex architecture utilizing Network-in-Network (NIN) principles.
  + Includes multiple convolutional layers, max-pooling, batch normalization, and global average pooling layers.
  + Parallel processing of packet headers and bodies for improved feature extraction.

5. Visualization: The application provides graphical representations of the dataset (bar chart) and model performance (bar chart comparing metrics) for easy interpretation.

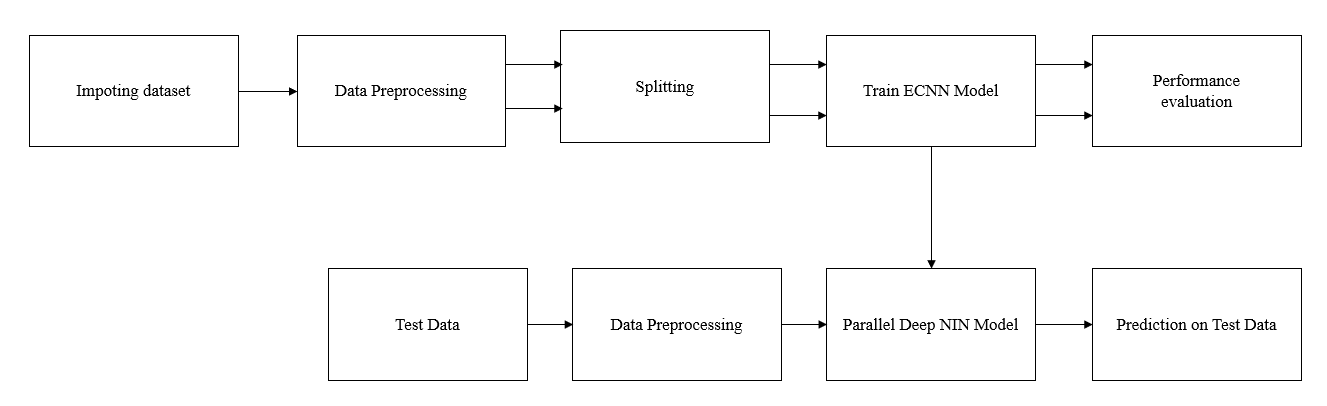
****

Figure 4.1: Architectural diagram of Proposed method.

**4.2 Parallel Deep NIN Model:**

The Network-In-Network (NIN) structure was first proposed in 2013, and has been applied to many tasks, such as image recognition , object detection , and speech recognition. Comparing with traditional CNN structure, it makes two modifications.

First, NIN adopts a micro network after each convolution layer to enhance its local modeling and abstraction ability. Figure 1 shows the comparison between a traditional convolution in CNNs and an MLP convolution in NINs which adopts a multilayer perceptron (MLP) as the instantiation of the micro network. For a traditional convolution, the computation on local receptive field can be seen as a single linear operation. Then the computation output is activated by nonlinear functions, e.g., Rectified Linear Unit (ReLU). In contrast, the MLP convolution owns an MLP-based micro network and there is a nonlinear activation after each layer in the micro network. Therefore, MLP convolution enhances model’s capacity of nonlinear expressiveness and allows complicated interactions across channels.

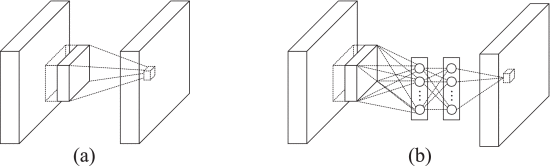
[[](https://ieeexplore.ieee.org/mediastore_new/IEEE/content/media/6287639/8948470/9144586/ling2ab-3010637-large.gif)](https://ieeexplore.ieee.org/mediastore_new/IEEE/content/media/6287639/8948470/9144586/ling2ab-3010637-large.gif)

Figure 4.2: Comparison between (a) a traditional convolution in CNNs and (b) an MLP convolution in NINs.

The NIN structure uses a global average pooling (GAP) layer to replace the fully connected layers after convolutions in conventional CNNs. The differences between these two structures are shown in Figure 2. In traditional CNNs for classification, the feature maps calculated by the last convolutional layer are flattened into a one-dimensional vector and then sent into fully connected layers. Finally, the output of the last fully connected layer is processed by a softmax layer. However, these fully connected layers contain a large amount of parameters and are prone to cause overfitting issues, resulting in unsatisfactory generalization ability of networks. In contrast, the GAP layer in NINs averages the feature maps given by the last convolution layer, and the output vector is directly fed into the softmax layer. The use of global average pooling can reduce model complexity and avoid model overfitting effectively.

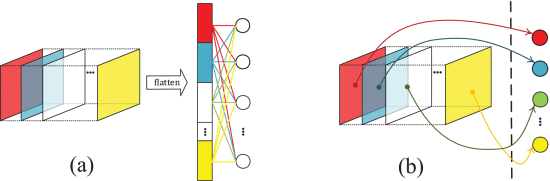
[](https://ieeexplore.ieee.org/mediastore_new/IEEE/content/media/6287639/8948470/9144586/ling3ab-3010637-large.gif)

Figure 4.3: Comparison between (a) fully connected layers in cnns and

(b) the global average pooling in nins.

The NINs into encrypted network traffic classification and the model structure is illustrated in Figure 3.

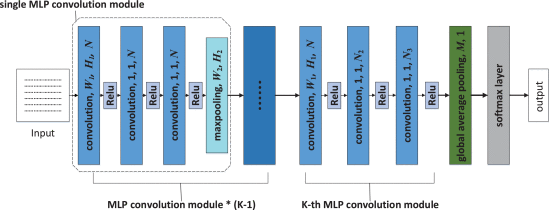
[[](https://ieeexplore.ieee.org/mediastore_new/IEEE/content/media/6287639/8948470/9144586/ling4-3010637-large.gif)](https://ieeexplore.ieee.org/mediastore_new/IEEE/content/media/6287639/8948470/9144586/ling4-3010637-large.gif)

Figure 4.4: The structure of the nin model used in our proposed method.

As shown in this figure, the NIN model contains K MLP convolution modules and the first K−1 ones have the same structure. Each MLP convolution module is composed of a linear convolution layer and two convolution layers with 1×1 convolution kernel. These two 1×1 convolution layers act as the MLP-based micro network mentioned above. Let F∈Rd′×N denote the output of the convolution layer as shown in below equation. Then, the output of the micro network is calculated as

F′=ReLU(ReLU(F⋅V1)⋅V2),(2)

where ⋅ stands for matrix multiplication and {V1,V2}∈RN×N are the parameters of the two 1×1 convolution layers, which enhances the nonlinear and cross-channel interactions in the model.

In the first K−1 MLP convolution modules, all convolution layers are activated by ReLU and have the same filter number N . Meanwhile, there is a max pooling layer at the end of each MLP convolution module with the kernel size W2 and the stride H2 . The structure of the last MLP convolution module is slightly different from the previous ones. First, the three convolution layers within this module have different filter numbers (N , N2 , N3 ) respectively, because a channel dimensionality reduction is necessary to meet the requirement of global average pooling. Here, N3 is the number of traffic or application types for classification. Second, there is no max pooling in this module. The output of all K MLP convolution modules is a two-dimensional matrix with dimensions of M×N3 , which refer to the size of each feature map and the number of feature maps respectively. Then for each feature map, a global average pooling with kernel size M is conducted. Finally, the resulted vector with size of 1×N3 is sent to the softmax layer to complete the prediction.

Parallel Decision Using NINs

The studies packet-level encrypted traffic classification. As shown in Figure 4, each data packet processed by neural networks is composed of three segments corresponding to the Network Layer, the Transport Layer and the Application Layer of TCP/IP model respectively. In our implementation, these three segments contain fixed 20, 20 and 1460 bytes respectively after padding or truncating during data pre-processing. In previous deep learning based traffic classification methods, a data packet is usually treated as a whole and is sent into a single model for classification. However, considering that different packet segments may provide different kinds of clues for traffic classification, such as the port numbers at the Transport Layer and the data patterns at the Application Layer,

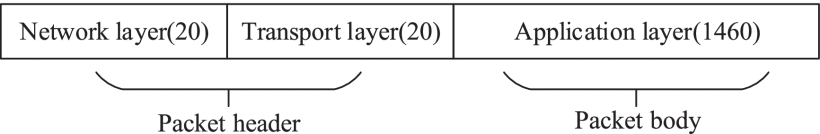


Figure 4.5: The structure of a data packet used in our implementation.

Therefore, The designs a parallel decision strategy for neural network based traffic classification. Its diagram is shown in Figure 5. A data packet is first split into two parts, a packet header and a packet body, according to Figure 4. Then, these two parts are sent into two NIN models to calculate classification probabilities separately. At last, the two results are fused to obtain the final one.

Let vectors

y1=[y1,1,…y1,N3]⊤ and y2=[y2,1,…y2,N3]⊤

 denote the softmax outputs of the two NIN models, where y1,m and y2,m stand for the probabilities that the packet should be classified as the m -the class calculated by the two NIN models. These two vectors are fused linearly as

y=αy1+(1−α)y2,(3)

where the vector y contains the final probability predicted for each category and α is a fusion weight.

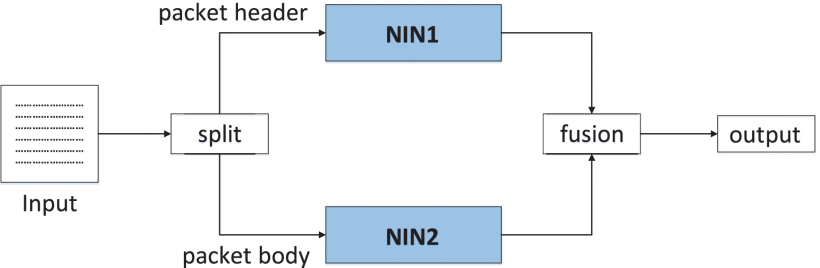
[](https://ieeexplore.ieee.org/mediastore_new/IEEE/content/media/6287639/8948470/9144586/ling6-3010637-large.gif)

Figure 4.6: The diagram of parallel decision for traffic classification.

At the training stage, all the data packets in the training set are first divided into headers and bodies, and then two NIN models are built accordingly. Then, we tune the weight α within the range of (0, 1) and observe the performance of fused probabilities y on the validation set. The optimal α is determined when the fused prediction achieves the best performance on the validation set.

**CHAPTER 5**

**UML DAIGRAMS**

UML stands for Unified Modeling Language. UML is a standardized general-purpose modeling language in the field of object-oriented software engineering. The standard is managed, and was created by, the Object Management Group. The goal is for UML to become a common language for creating models of object-oriented computer software. In its current form UML is comprised of two major components: a Meta-model and a notation. In the future, some form of method or process may also be added to; or associated with, UML.

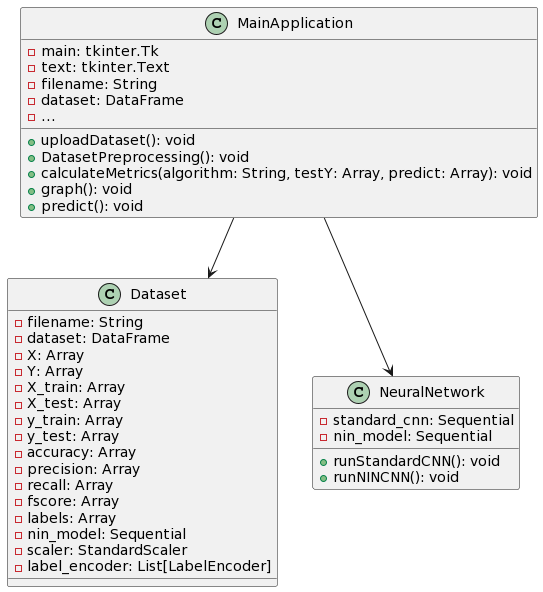
The Unified Modeling Language is a standard language for specifying, Visualization, Constructing and documenting the artifacts of software system, as well as for business modeling and other non-software systems. The UML represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems. The UML is a very important part of developing objects-oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects.

**GOALS:** The Primary goals in the design of the UML are as follows:

* Provide users a ready-to-use, expressive visual modeling Language so that they can develop and exchange meaningful models.
* Provide extendibility and specialization mechanisms to extend the core concepts.
* Be independent of particular programming languages and development process.
* Provide a formal basis for understanding the modeling language.
* Encourage the growth of OO tools market.
* Support higher level development concepts such as collaborations, frameworks, patterns and components.
* Integrate best practices.

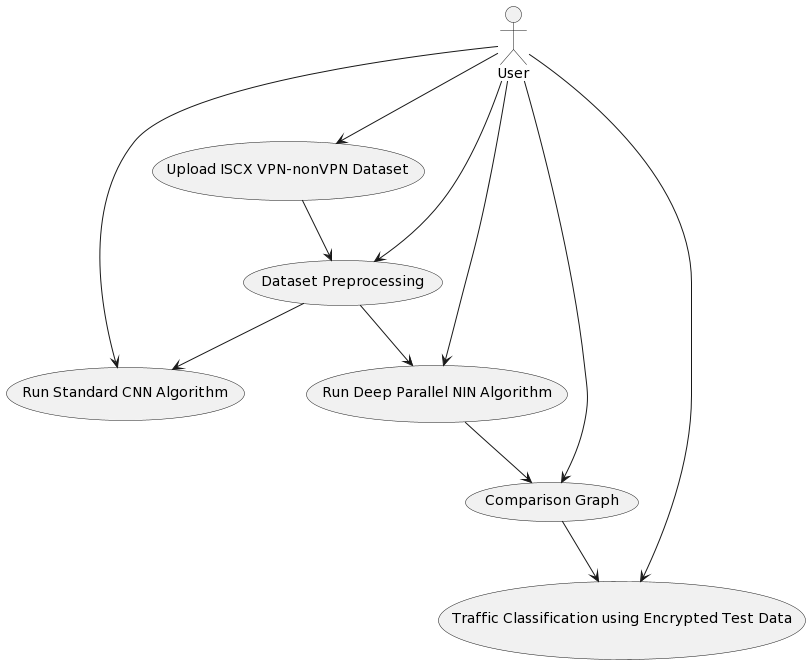
**Class diagram**

The class diagram is used to refine the use case diagram and define a detailed design of the system. The class diagram classifies the actors defined in the use case diagram into a set of interrelated classes. The relationship or association between the classes can be either an "is-a" or "has-a" relationship. Each class in the class diagram may be capable of providing certain functionalities. These functionalities provided by the class are termed "methods" of the class. Apart from this, each class may have certain "attributes" that uniquely identify the class.



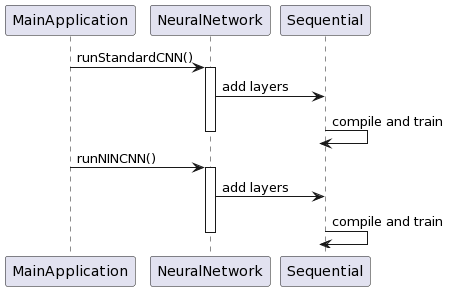
**Use case Diagram**

A use case diagram in the Unified Modeling Language (UML) is a type of behavioral diagram defined by and created from a Use-case analysis. Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals (represented as use cases), and any dependencies between those use cases. The main purpose of a use case diagram is to show what system functions are performed for which actor. Roles of the actors in the system can be depicted.

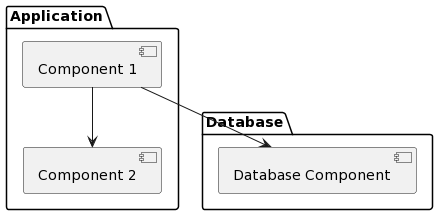


**Sequence Diagram**

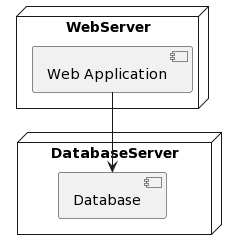
A **sequence diagram** in Unified Modeling Language (UML) is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message Sequence Chart. A sequence diagram shows, as parallel vertical lines ("lifelines"), different processes or objects that live simultaneously, and as horizontal arrows, the messages exchanged between them, in the order in which they occur. This allows the specification of simple runtime scenarios in a graphical manner.



**Component Diagram:**

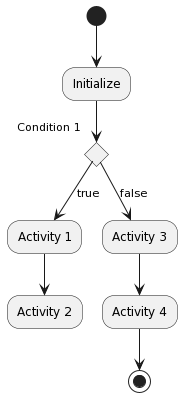


**Deployment diagram:** The deployment diagram visualizes the physical hardware on which the software will be deployed.



**Activity diagram**: Activity diagrams are graphical representations of Workflows of stepwise activities and actions with support for choice, iteration, and concurrency.

In the Unified Modeling Language, activity diagrams can be used to describe the business and operational step-by-step workflows of components in a system. An activity diagram shows the overall flow of control.



**CHAPTER 6**

**SOFTWARE ENVIRONMENT**

**What is Python?**

Below are some facts about Python.

* Python is currently the most widely used multi-purpose, high-level programming language.
* Python allows programming in Object-Oriented and Procedural paradigms. Python programs generally are smaller than other programming languages like Java.
* Programmers have to type relatively less and indentation requirement of the language, makes them readable all the time.
* Python language is being used by almost all tech-giant companies like – Google, Amazon, Facebook, Instagram, Dropbox, Uber… etc.

The biggest strength of Python is huge collection of standard libraries which can be used for the following –

* + Machine Learning
  + GUI Applications (like Kivy, Tkinter, PyQt etc.)
  + Web frameworks like Django (used by YouTube, Instagram, Dropbox)
  + Image processing (like Opencv, Pillow)
  + Web scraping (like Scrapy, BeautifulSoup, Selenium)
  + Test frameworks
  + Multimedia

**Advantages of Python**

Let’s see how Python dominates over other languages.

1. Extensive Libraries

Python downloads with an extensive library and it contain code for various purposes like regular expressions, documentation-generation, unit-testing, web browsers, threading, databases, CGI, email, image manipulation, and more. So, we don’t have to write the complete code for that manually.

1. Extensible

As we have seen earlier, Python can be extended to other languages. You can write some of your code in languages like C++ or C. This comes in handy, especially in projects.

1. Embeddable

Complimentary to extensibility, Python is embeddable as well. You can put your Python code in your source code of a different language, like C++. This lets us add scripting capabilities to our code in the other language.

1. Improved Productivity

The language’s simplicity and extensive libraries render programmers more productive than languages like Java and C++ do. Also, the fact that you need to write less and get more things done.

1. IOT Opportunities

Since Python forms the basis of new platforms like Raspberry Pi, it finds the future bright for the Internet of Things. This is a way to connect the language with the real world.

1. Simple and Easy

When working with Java, you may have to create a class to print ‘Hello World’. But in Python, just a print statement will do. It is also quite easy to learn, understand, and code. This is why when people pick up Python, they have a hard time adjusting to other more verbose languages like Java.

1. Readable

Because it is not such a verbose language, reading Python is much like reading English. This is the reason why it is so easy to learn, understand, and code. It also does not need curly braces to define blocks, and indentation is mandatory. These further aids the readability of the code.

1. Object-Oriented

This language supports both the procedural and object-oriented programming paradigms. While functions help us with code reusability, classes and objects let us model the real world. A class allows the encapsulation of data and functions into one.

1. Free and Open-Source

Like we said earlier, Python is freely available. But not only can you download Python for free, but you can also download its source code, make changes to it, and even distribute it. It downloads with an extensive collection of libraries to help you with your tasks.

1. Portable

When you code your project in a language like C++, you may need to make some changes to it if you want to run it on another platform. But it isn’t the same with Python. Here, you need to code only once, and you can run it anywhere. This is called Write Once Run Anywhere (WORA). However, you need to be careful enough not to include any system-dependent features.

1. Interpreted

Lastly, we will say that it is an interpreted language. Since statements are executed one by one, debugging is easier than in compiled languages.

Any doubts till now in the advantages of Python? Mention in the comment section.

**Advantages of Python Over Other Languages**

1. Less Coding

Almost all of the tasks done in Python requires less coding when the same task is done in other languages. Python also has an awesome standard library support, so you don’t have to search for any third-party libraries to get your job done. This is the reason that many people suggest learning Python to beginners.

1. Affordable

Python is free therefore individuals, small companies or big organizations can leverage the free available resources to build applications. Python is popular and widely used so it gives you better community support.

The 2019 Github annual survey showed us that Python has overtaken Java in the most popular programming language category.

1. Python is for Everyone

Python code can run on any machine whether it is Linux, Mac or Windows. Programmers need to learn different languages for different jobs but with Python, you can professionally build web apps, perform data analysis and machine learning, automate things, do web scraping and also build games and powerful visualizations. It is an all-rounder programming language.

**Disadvantages of Python**

So far, we’ve seen why Python is a great choice for your project. But if you choose it, you should be aware of its consequences as well. Let’s now see the downsides of choosing Python over another language.

1. Speed Limitations

We have seen that Python code is executed line by line. But since Python is interpreted, it often results in slow execution. This, however, isn’t a problem unless speed is a focal point for the project. In other words, unless high speed is a requirement, the benefits offered by Python are enough to distract us from its speed limitations.

2. Weak in Mobile Computing and Browsers

While it serves as an excellent server-side language, Python is much rarely seen on the client-side. Besides that, it is rarely ever used to implement smartphone-based applications. One such application is called Carbonnelle.

The reason it is not so famous despite the existence of Brython is that it isn’t that secure.

3. Design Restrictions

As you know, Python is dynamically-typed. This means that you don’t need to declare the type of variable while writing the code. It uses duck-typing. But wait, what’s that? Well, it just means that if it looks like a duck, it must be a duck. While this is easy on the programmers during coding, it can raise run-time errors.

4. Underdeveloped Database Access Layers

Compared to more widely used technologies like JDBC (Java DataBase Connectivity) and ODBC (Open DataBase Connectivity), Python’s database access layers are a bit underdeveloped. Consequently, it is less often applied in huge enterprises.

5. Simple

No, we’re not kidding. Python’s simplicity can indeed be a problem. Take my example. I don’t do Java, I’m more of a Python person. To me, its syntax is so simple that the verbosity of Java code seems unnecessary.

This was all about the Advantages and Disadvantages of Python Programming Language.

**History of Python**

What do the alphabet and the programming language Python have in common? Right, both start with ABC. If we are talking about ABC in the Python context, it's clear that the programming language ABC is meant. ABC is a general-purpose programming language and programming environment, which had been developed in the Netherlands, Amsterdam, at the CWI (Centrum Wiskunde &Informatica). The greatest achievement of ABC was to influence the design of Python. Python was conceptualized in the late 1980s. Guido van Rossum worked that time in a project at the CWI, called Amoeba, a distributed operating system. In an interview with Bill Venners1, Guido van Rossum said: "In the early 1980s, I worked as an implementer on a team building a language called ABC at Centrum voor Wiskunde en Informatica (CWI). I don't know how well people know ABC's influence on Python. I try to mention ABC's influence because I'm indebted to everything I learned during that project and to the people who worked on it. "Later on in the same Interview, Guido van Rossum continued: "I remembered all my experience and some of my frustration with ABC. I decided to try to design a simple scripting language that possessed some of ABC's better properties, but without its problems. So, I started typing. I created a simple virtual machine, a simple parser, and a simple runtime. I made my own version of the various ABC parts that I liked. I created a basic syntax, used indentation for statement grouping instead of curly braces or begin-end blocks, and developed a small number of powerful data types: a hash table (or dictionary, as we call it), a list, strings, and numbers."

**Python Development Steps**

Guido Van Rossum published the first version of Python code (version 0.9.0) at alt.sources in February 1991. This release included already exception handling, functions, and the core data types of lists, dict, str and others. It was also object oriented and had a module system.  
Python version 1.0 was released in January 1994. The major new features included in this release were the functional programming tools lambda, map, filter and reduce, which Guido Van Rossum never liked. Six and a half years later in October 2000, Python 2.0 was introduced. This release included list comprehensions, a full garbage collector and it was supporting unicode. Python flourished for another 8 years in the versions 2.x before the next major release as Python 3.0 (also known as "Python 3000" and "Py3K") was released. Python 3 is not backwards compatible with Python 2.x. The emphasis in Python 3 had been on the removal of duplicate programming constructs and modules, thus fulfilling or coming close to fulfilling the 13th law of the Zen of Python: "There should be one -- and preferably only one -- obvious way to do it."Some changes in Python 7.3:

* Print is now a function.
* Views and iterators instead of lists
* The rules for ordering comparisons have been simplified. E.g., a heterogeneous list cannot be sorted, because all the elements of a list must be comparable to each other.
* There is only one integer type left, i.e., int. long is int as well.
* The division of two integers returns a float instead of an integer. "//" can be used to have the "old" behaviour.
* Text Vs. Data Instead of Unicode Vs. 8-bit

**Purpose**

We demonstrated that our approach enables successful segmentation of intra-retinal layers—even with low-quality images containing speckle noise, low contrast, and different intensity ranges throughout—with the assistance of the ANIS feature.

**Python**

Python is an interpreted high-level programming language for general-purpose programming. Created by Guido van Rossum and first released in 1991, Python has a design philosophy that emphasizes code readability, notably using significant whitespace.

Python features a dynamic type system and automatic memory management. It supports multiple programming paradigms, including object-oriented, imperative, functional and procedural, and has a large and comprehensive standard library.

* Python is Interpreted − Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.
* Python is Interactive − you can actually sit at a Python prompt and interact with the interpreter directly to write your programs.

Python also acknowledges that speed of development is important. Readable and terse code is part of this, and so is access to powerful constructs that avoid tedious repetition of code. Maintainability also ties into this may be an all but useless metric, but it does say something about how much code you have to scan, read and/or understand to troubleshoot problems or tweak behaviors. This speed of development, the ease with which a programmer of other languages can pick up basic Python skills and the huge standard library is key to another area where Python excels. All its tools have been quick to implement, saved a lot of time, and several of them have later been patched and updated by people with no Python background - without breaking.

**Modules Used in Project**

**TensorFlow**

TensorFlow is a free and open-source software library for dataflow and differentiable programming across a range of tasks. It is a symbolic math library and is also used for machine learning applications such as neural networks. It is used for both research and production at Google.‍

TensorFlow was developed by the Google Brain team for internal Google use. It was released under the Apache 2.0 open-source license on November 9, 2015.

**NumPy**

NumPy is a general-purpose array-processing package. It provides a high-performance multidimensional array object, and tools for working with these arrays.

It is the fundamental package for scientific computing with Python. It contains various features including these important ones:

* A powerful N-dimensional array object
* Sophisticated (broadcasting) functions
* Tools for integrating C/C++ and Fortran code
* Useful linear algebra, Fourier transform, and random number capabilities

Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary datatypes can be defined using NumPy which allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

**Pandas**

Pandas is an open-source Python Library providing high-performance data manipulation and analysis tool using its powerful data structures. Python was majorly used for data munging and preparation. It had very little contribution towards data analysis. Pandas solved this problem. Using Pandas, we can accomplish five typical steps in the processing and analysis of data, regardless of the origin of data load, prepare, manipulate, model, and analyze. Python with Pandas is used in a wide range of fields including academic and commercial domains including finance, economics, Statistics, analytics, etc.

**Matplotlib**

Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, the Python and IPython shells, the Jupyter Notebook, web application servers, and four graphical user interface toolkits. Matplotlib tries to make easy things easy and hard things possible. You can generate plots, histograms, power spectra, bar charts, error charts, scatter plots, etc., with just a few lines of code. For examples, see the sample plots and thumbnail gallery.

For simple plotting the pyplot module provides a MATLAB-like interface, particularly when combined with IPython. For the power user, you have full control of line styles, font properties, axes properties, etc, via an object-oriented interface or via a set of functions familiar to MATLAB users.

**Scikit – learn**

Scikit-learn provides a range of supervised and unsupervised learning algorithms via a consistent interface in Python. It is licensed under a permissive simplified BSD license and is distributed under many Linux distributions, encouraging academic and commercial use. Python

Python is an interpreted high-level programming language for general-purpose programming. Created by Guido van Rossum and first released in 1991, Python has a design philosophy that emphasizes code readability, notably using significant whitespace.

Python features a dynamic type system and automatic memory management. It supports multiple programming paradigms, including object-oriented, imperative, functional and procedural, and has a large and comprehensive standard library.

* Python is Interpreted − Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.
* Python is Interactive − you can actually sit at a Python prompt and interact with the interpreter directly to write your programs.

Python also acknowledges that speed of development is important. Readable and terse code is part of this, and so is access to powerful constructs that avoid tedious repetition of code. Maintainability also ties into this may be an all but useless metric, but it does say something about how much code you have to scan, read and/or understand to troubleshoot problems or tweak behaviors. This speed of development, the ease with which a programmer of other languages can pick up basic Python skills and the huge standard library is key to another area where Python excels. All its tools have been quick to implement, saved a lot of time, and several of them have later been patched and updated by people with no Python background - without breaking.

**Install Python Step-by-Step in Windows and Mac**

Python a versatile programming language doesn’t come pre-installed on your computer devices. Python was first released in the year 1991 and until today it is a very popular high-level programming language. Its style philosophy emphasizes code readability with its notable use of great whitespace.

The object-oriented approach and language construct provided by Python enables programmers to write both clear and logical code for projects. This software does not come pre-packaged with Windows.

**How to Install Python on Windows and Mac**

There have been several updates in the Python version over the years. The question is how to install Python? It might be confusing for the beginner who is willing to start learning Python but this tutorial will solve your query. The latest or the newest version of Python is version 3.7.4 or in other words, it is Python 3.

Note: The python version 3.7.4 cannot be used on Windows XP or earlier devices.

Before you start with the installation process of Python. First, you need to know about your System Requirements. Based on your system type i.e., operating system and based processor, you must download the python version. My system type is a Windows 64-bit operating system. So, the steps below are to install python version 3.7.4 on Windows 7 device or to install Python 3. Download the Python Cheatsheet here. The steps on how to install Python on Windows 10, 8 and 7 are divided into 4 parts to help understand better.

**Download the Correct version into the system**

Step 1: Go to the official site to download and install python using Google Chrome or any other web browser. OR Click on the following link: https://www.python.org

A screenshot of a computer

Description automatically generated with medium confidence

Now, check for the latest and the correct version for your operating system.

Step 2: Click on the Download Tab.

Graphical user interface, application

Description automatically generated

Step 3: You can either select the Download Python for windows 3.7.4 button in Yellow Color or you can scroll further down and click on download with respective to their version. Here, we are downloading the most recent python version for windows 3.7.4

Graphical user interface, application

Description automatically generated

Step 4: Scroll down the page until you find the Files option.

Step 5: Here you see a different version of python along with the operating system.

Graphical user interface, text

Description automatically generated

* To download Windows 32-bit python, you can select any one from the three options: Windows x86 embeddable zip file, Windows x86 executable installer or Windows x86 web-based installer.
* To download Windows 64-bit python, you can select any one from the three options: Windows x86-64 embeddable zip file, Windows x86-64 executable installer or Windows x86-64 web-based installer.

Here we will install Windows x86-64 web-based installer. Here your first part regarding which version of python is to be downloaded is completed. Now we move ahead with the second part in installing python i.e., Installation

Note: To know the changes or updates that are made in the version you can click on the Release Note Option.

Installation of Python

Step 1: Go to Download and Open the downloaded python version to carry out the installation process.

Graphical user interface, text, application

Description automatically generated

Step 2: Before you click on Install Now, make sure to put a tick on Add Python 3.7 to PATH.

Graphical user interface, text, application, chat or text message

Description automatically generated

Step 3: Click on Install NOW After the installation is successful. Click on Close.

Graphical user interface, text, application, chat or text message

Description automatically generated

With these above three steps on python installation, you have successfully and correctly installed Python. Now is the time to verify the installation.

Note: The installation process might take a couple of minutes.

Verify the Python Installation

Step 1: Click on Start

Step 2: In the Windows Run Command, type “cmd”.

Graphical user interface, application

Description automatically generated

Step 3: Open the Command prompt option.

Step 4: Let us test whether the python is correctly installed. Type python –V and press Enter.

A screenshot of a computer

Description automatically generated with medium confidence

Step 5: You will get the answer as 3.7.4

Note: If you have any of the earlier versions of Python already installed. You must first uninstall the earlier version and then install the new one.

Check how the Python IDLE works

Step 1: Click on Start

Step 2: In the Windows Run command, type “python idle”.

Application

Description automatically generated with low confidence

Step 3: Click on IDLE (Python 3.7 64-bit) and launch the program

Step 4: To go ahead with working in IDLE you must first save the file. Click on File > Click on Save

Graphical user interface, text, application, email

Description automatically generated

Step 5: Name the file and save as type should be Python files. Click on SAVE. Here I have named the files as Hey World.

Step 6: Now for e.g., enter print (“Hey World”) and Press Enter.

Graphical user interface, text, application, email

Description automatically generated

You will see that the command given is launched. With this, we end our tutorial on how to install Python. You have learned how to download python for windows into your respective operating system.

Note: Unlike Java, Python does not need semicolons at the end of the statements otherwise it won’t work.

**CHAPTER 7**

**SYSTEM REQUIREMENTS SPECIFICATIONS**

**Software Requirements**

The functional requirements or the overall description documents include the product perspective and features, operating system and operating environment, graphics requirements, design constraints and user documentation.

The appropriation of requirements and implementation constraints gives the general overview of the project in regard to what the areas of strength and deficit are and how to tackle them.

* Python IDLE 3.7 version (or)
* Anaconda 3.7 (or)
* Jupiter (or)
* Google colab

**Hardware Requirements**

Minimum hardware requirements are very dependent on the particular software being developed by a given Enthought Python / Canopy / VS Code user. Applications that need to store large arrays/objects in memory will require more RAM, whereas applications that need to perform numerous calculations or tasks more quickly will require a faster processor.

Operating system : Windows, Linux

Processor : minimum intel i3

Ram : minimum 4 GB

Hard disk : minimum 250GB

**CHAPTER 8**

**FUNCTIONAL REQUIREMENTS**

**Output Design**

Outputs from computer systems are required primarily to communicate the results of processing to users. They are also used to provides a permanent copy of the results for later consultation. The various types of outputs in general are:

* External Outputs, whose destination is outside the organization
* Internal Outputs whose destination is within organization and they are the
* User’s main interface with the computer.
* Operational outputs whose use is purely within the computer department.
* Interface outputs, which involve the user in communicating directly.

**Output Definition**

The outputs should be defined in terms of the following points:

* Type of the output
* Content of the output
* Format of the output
* Location of the output
* Frequency of the output
* Volume of the output
* Sequence of the output

It is not always desirable to print or display data as it is held on a computer. It should be decided as which form of the output is the most suitable.

**Input Design**

Input design is a part of overall system design. The main objective during the input design is as given below:

* To produce a cost-effective method of input.
* To achieve the highest possible level of accuracy.
* To ensure that the input is acceptable and understood by the user.

**Input Stages**

The main input stages can be listed as below:

* Data recording
* Data transcription
* Data conversion
* Data verification
* Data control
* Data transmission
* Data validation
* Data correction

**Input Types**

It is necessary to determine the various types of inputs. Inputs can be categorized as follows:

* External inputs, which are prime inputs for the system.
* Internal inputs, which are user communications with the system.
* Operational, which are computer department’s communications to the system?
* Interactive, which are inputs entered during a dialogue.

**Input Media**

At this stage choice has to be made about the input media. To conclude about the input media consideration has to be given to;

* Type of input
* Flexibility of format
* Speed
* Accuracy
* Verification methods
* Rejection rates
* Ease of correction
* Storage and handling requirements
* Security
* Easy to use
* Portability

Keeping in view the above description of the input types and input media, it can be said that most of the inputs are of the form of internal and interactive. As

Input data is to be the directly keyed in by the user, the keyboard can be considered to be the most suitable input device.

**Error Avoidance**

At this stage care is to be taken to ensure that input data remains accurate form the stage at which it is recorded up to the stage in which the data is accepted by the system. This can be achieved only by means of careful control each time the data is handled.

**Error Detection**

Even though every effort is made to avoid the occurrence of errors, still a small proportion of errors is always likely to occur, these types of errors can be discovered by using validations to check the input data.

**Data Validation**

Procedures are designed to detect errors in data at a lower level of detail. Data validations have been included in the system in almost every area where there is a possibility for the user to commit errors. The system will not accept invalid data. Whenever an invalid data is keyed in, the system immediately prompts the user and the user has to again key in the data and the system will accept the data only if the data is correct. Validations have been included where necessary.

The system is designed to be a user friendly one. In other words the system has been designed to communicate effectively with the user. The system has been designed with popup menus.

**User Interface Design**

It is essential to consult the system users and discuss their needs while designing the user interface:

**User Interface Systems Can Be Broadly Clasified As:**

* User initiated interface the user is in charge, controlling the progress of the user/computer dialogue. In the computer-initiated interface, the computer selects the next stage in the interaction.
* Computer initiated interfaces

In the computer-initiated interfaces the computer guides the progress of the user/computer dialogue. Information is displayed and the user response of the computer takes action or displays further information.

**User Initiated Interfaces**

User initiated interfaces fall into two approximate classes:

* Command driven interfaces: In this type of interface the user inputs commands or queries which are interpreted by the computer.
* Forms oriented interface: The user calls up an image of the form to his/her screen and fills in the form. The forms-oriented interface is chosen because it is the best choice.

**Computer-Initiated Interfaces**

The following computer – initiated interfaces were used:

* The menu system for the user is presented with a list of alternatives and the user chooses one; of alternatives.
* Questions – answer type dialog system where the computer asks question and takes action based on the basis of the users reply.

Right from the start the system is going to be menu driven, the opening menu displays the available options. Choosing one option gives another popup menu with more options. In this way every option leads the users to data entry form where the user can key in the data.

**Error Message Design**

The design of error messages is an important part of the user interface design. As user is bound to commit some errors or other while designing a system the system should be designed to be helpful by providing the user with information regarding the error he/she has committed.

This application must be able to produce output at different modules for different inputs.

**Performance Requirements**

Performance is measured in terms of the output provided by the application. Requirement specification plays an important part in the analysis of a system. Only when the requirement specifications are properly given, it is possible to design a system, which will fit into required environment. It rests largely in the part of the users of the existing system to give the requirement specifications because they are the people who finally use the system. This is because the requirements have to be known during the initial stages so that the system can be designed according to those requirements. It is very difficult to change the system once it has been designed and on the other hand designing a system, which does not cater to the requirements of the user, is of no use.

The requirement specification for any system can be broadly stated as given below:

* The system should be able to interface with the existing system
* The system should be accurate
* The system should be better than the existing system
* The existing system is completely dependent on the user to perform all the duties.

**CHAPTER 9**

**SOURCE CODE**

from tkinter import messagebox

from tkinter import \*

from tkinter import simpledialog

import tkinter

import matplotlib.pyplot as plt

import numpy as np

from tkinter import simpledialog

from tkinter import filedialog

import os

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import confusion\_matrix

from sklearn.metrics import precision\_score

from sklearn.metrics import recall\_score

from sklearn.metrics import f1\_score

import seaborn as sns

import pandas as pd

from sklearn.preprocessing import LabelEncoder

from keras.callbacks import ModelCheckpoint

from keras.layers import MaxPooling2D

from keras.layers import Dense, Dropout, Activation, Flatten

from keras.layers import Convolution2D

from keras.models import Sequential, load\_model

from keras.utils.np\_utils import to\_categorical

from keras.layers import Conv2D, MaxPool2D, InputLayer, BatchNormalization, GlobalAveragePooling2D

from keras.optimizers import Adam

main = tkinter.Tk()

main.title("Deep and Parallel Network-in-Network Models for Accurate Encrypted Network Traffic Classification") #designing main screen

main.geometry("1300x1200")

global filename, dataset

global X, Y

global X\_train, X\_test, y\_train, y\_test

global accuracy, precision, recall, fscore, labels, nin\_model

global scaler, labels, label\_encoder

def uploadDataset():

global filename, dataset, labels

filename = filedialog.askopenfilename(initialdir="Dataset")

text.delete('1.0', END)

text.insert(END,filename+" loaded\n\n")

dataset = pd.read\_csv(filename)

text.insert(END,str(dataset))

labels, label\_count = np.unique(dataset['Label'], return\_counts=True)

label = dataset.groupby('Label').size()

label.plot(kind="bar")

plt.xlabel("Network Category Type")

plt.ylabel("Count")

plt.title("Network Category Graph")

plt.show()

def DatasetPreprocessing():

text.delete('1.0', END)

global X, Y, dataset, label\_encoder

global X\_train, X\_test, y\_train, y\_test, scaler

#dataset contains non-numeric values but ML algorithms accept only numeric values so by applying Lable

#encoding class converting all non-numeric data into numeric data

dataset.fillna(0, inplace = True)

dataset.drop(['Traffic\_Type'], axis = 1,inplace=True)

label\_encoder = []

columns = dataset.columns

types = dataset.dtypes.values

for i in range(len(types)):

name = types[i]

if name == 'object': #finding column with object type

le = LabelEncoder()

dataset[columns[i]] = pd.Series(le.fit\_transform(dataset[columns[i]].astype(str)))#encode all str columns to numeric

label\_encoder.append(le)

text.insert(END,"Dataset Normalization & Preprocessing Task Completed\n\n")

text.insert(END,str(dataset)+"\n\n")

#dataset preprocessing such as replacing missing values, normalization and splitting dataset into train and test

data = dataset.values

X = data[:,0:data.shape[1]-1] #extracting X and Y Features from the dataset

Y = data[:,data.shape[1]-1]

print(X.shape)

print(np.unique(Y))

print(Y)

Y = Y.astype(int)

indices = np.arange(X.shape[0])

np.random.shuffle(indices) #shuffling the dataset

X = X[indices]

Y = Y[indices]

#normalizing or scaling values

scaler = StandardScaler()

X = scaler.fit\_transform(X)

#reshape dataset as 3 dimenssion

X = np.reshape(X, (X.shape[0], X.shape[1], 1, 1))

Y = to\_categorical(Y)

#splitting dataset into train and test where application using 80% dataset for training and 20% for testing

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, Y, test\_size=0.2) #split dataset into train and test

text.insert(END,"Dataset Train & Test Splits\n")

text.insert(END,"Total images found in dataset : "+str(X.shape[0])+"\n")

text.insert(END,"80% dataset used for training : "+str(X\_train.shape[0])+"\n")

text.insert(END,"20% dataset user for testing : "+str(X\_test.shape[0])+"\n")

def calculateMetrics(algorithm, testY, predict):

global labels

p = precision\_score(testY, predict,average='macro') \* 100

r = recall\_score(testY, predict,average='macro') \* 100

f = f1\_score(testY, predict,average='macro') \* 100

a = accuracy\_score(testY,predict)\*100

accuracy.append(a)

precision.append(p)

recall.append(r)

fscore.append(f)

text.insert(END,algorithm+" Accuracy : "+str(a)+"\n")

text.insert(END,algorithm+" Precision : "+str(p)+"\n")

text.insert(END,algorithm+" Recall : "+str(r)+"\n")

text.insert(END,algorithm+" FSCORE : "+str(f)+"\n\n")

conf\_matrix = confusion\_matrix(testY, predict)

ax = sns.heatmap(conf\_matrix, xticklabels = labels, yticklabels = labels, annot = True, cmap="viridis" ,fmt ="g");

ax.set\_ylim([0,len(labels)])

plt.title(algorithm+" Confusion matrix")

plt.ylabel('True class')

plt.xlabel('Predicted class')

plt.show()

#now train existing standard CNN algorithm

def runStandardCNN():

text.delete('1.0', END)

global accuracy, precision, recall, fscore

global X\_train, y\_train, X\_test, y\_test

accuracy = []

precision = []

recall = []

fscore = []

#training standard CNN without multilayer perceptron and global average pooling

standard\_cnn = Sequential()

standard\_cnn.add(Convolution2D(32, (1, 1), input\_shape = (X\_train.shape[1], X\_train.shape[2], X\_train.shape[3]), activation = 'relu'))

standard\_cnn.add(MaxPooling2D(pool\_size = (1, 1)))

standard\_cnn.add(Convolution2D(32, (1, 1), activation = 'relu'))

standard\_cnn.add(MaxPooling2D(pool\_size = (1, 1)))

standard\_cnn.add(Flatten())

standard\_cnn.add(Dense(units = 256, activation = 'relu'))

standard\_cnn.add(Dense(units = y\_train.shape[1], activation = 'softmax'))

standard\_cnn.compile(optimizer = 'adam', loss = 'categorical\_crossentropy', metrics = ['accuracy'])

#train and load the model

if os.path.exists("model/standard\_cnn\_weights.hdf5") == False:

model\_check\_point = ModelCheckpoint(filepath='model/standard\_cnn\_weights.hdf5', verbose = 1, save\_best\_only = True)

hist = standard\_cnn.fit(X\_train, y\_train, batch\_size = 256, epochs = 10, validation\_data=(X\_test, y\_test), callbacks=[model\_check\_point], verbose=1)

else:

standard\_cnn = load\_model("model/standard\_cnn\_weights.hdf5")

predict = standard\_cnn.predict(X\_test)

predict = np.argmax(predict, axis=1)

testY = np.argmax(y\_test, axis=1)

calculateMetrics("Existing Standard CNN", testY, predict)

#run propose NIN model

def runNINCNN():

global nin\_model

global X\_train, y\_train, X\_test, y\_test

#now creating multi layer perceptron with global average pooling layer where first 3 layers are used to processed packet header and remaining layer

#will process packet body

nin\_model = Sequential()

nin\_model.add(InputLayer(input\_shape=(X\_train.shape[1], X\_train.shape[2], X\_train.shape[3])))

#creating first hidden layer with 25 neurons to filter data 25 times for packet header

nin\_model.add(Conv2D(25, (5, 5), activation='relu', strides=(1, 1), padding='same'))

#defining another layer

nin\_model.add(MaxPool2D(pool\_size=(2, 2), padding='same'))

nin\_model.add(Conv2D(50, (5, 5), activation='relu', strides=(2, 2), padding='same'))

nin\_model.add(MaxPool2D(pool\_size=(2, 2), padding='same'))

nin\_model.add(BatchNormalization())

#now creating second filtration layer to filter packet body

nin\_model.add(Conv2D(70, (3, 3), activation='relu', strides=(2, 2), padding='same'))

nin\_model.add(MaxPool2D(pool\_size=(1, 1), padding='valid'))

nin\_model.add(BatchNormalization())

nin\_model.add(Dense(units=100, activation='relu'))

nin\_model.add(Dense(units=100, activation='relu'))

nin\_model.add(Dropout(0.25))

#now adding global average pooling layer

nin\_model.add(GlobalAveragePooling2D())

nin\_model.add(Dense(units=y\_train.shape[1], activation='softmax'))

#compiling the model

nin\_model.compile(loss='categorical\_crossentropy', optimizer="adam", metrics=['accuracy'])

#now train and load the model

if os.path.exists("model/nin\_weights.hdf5") == False:

model\_check\_point = ModelCheckpoint(filepath='model/nin\_weights.hdf5', verbose = 1, save\_best\_only = True)

hist = nin\_model.fit(X\_train, y\_train, batch\_size = 256, epochs = 10, validation\_data=(X\_test, y\_test), callbacks=[model\_check\_point], verbose=1)

else:

nin\_model.load\_weights("model/nin\_weights.hdf5")

#performing prediction on test data and calculate accuracy and other metrics

predict = nin\_model.predict(X\_test)

predict = np.argmax(predict, axis=1)

testY = np.argmax(y\_test, axis=1)

calculateMetrics("Proposed Parallel Deep NIN Model", testY, predict)

def graph():

df = pd.DataFrame([['Standard CNN','Accuracy',accuracy[0]],['Standard CNN','Precision',precision[0]],['Standard CNN','Recall',recall[0]],['Standard CNN','FSCORE',fscore[0]],

['Proposed Parallel Deep NIN Model','Accuracy',accuracy[1]],['Proposed Parallel Deep NIN Model','Precision',precision[1]],['Proposed Parallel Deep NIN Model','Recall',recall[1]],['Proposed Parallel Deep NIN Model','FSCORE',fscore[1]],

],columns=['Algorithms','Accuracy','Value'])

df.pivot("Algorithms", "Accuracy", "Value").plot(kind='bar')

plt.title("All Algorithm Comparison Graph")

plt.show()

def predict():

global nin\_model, scaler, label\_encoder, labels

text.delete('1.0', END)

filename = filedialog.askopenfilename(initialdir="Dataset")#upload test data

dataset = pd.read\_csv(filename)#read data from uploaded file

dataset.fillna(0, inplace = True)#removing missing values

index = 0

columns = dataset.columns

types = dataset.dtypes.values

for i in range(len(types)): #label encoding to convert non-numeric data to numeric data

name = types[i]

if name == 'object': #finding column with object type

dataset[columns[i]] = pd.Series(label\_encoder[index].fit\_transform(dataset[columns[i]].astype(str)))

index = index + 1

dataset = dataset.values

X = scaler.transform(dataset)#normalizing values

X = np.reshape(X, (X.shape[0], X.shape[1], 1, 1))

traffic\_type\_predict = nin\_model.predict(X)#performing prediction on test data

for i in range(len(X)):

text.insert(END,"Traffic Test Data : "+str(dataset[i]))

text.insert(END,"Network Traffic Classified As ===> "+labels[int(np.argmax(traffic\_type\_predict[i]))])

text.insert(END,"\n")

font = ('times', 16, 'bold')

title = Label(main, text='Deep and Parallel Network-in-Network Models for Accurate Encrypted Network Traffic Classification')

title.config(bg='LightGoldenrod1', fg='dark orchid')

title.config(font=font)

title.config(height=3, width=120)

title.place(x=0,y=5)

font1 = ('times', 12, 'bold')

text=Text(main,height=22,width=140)

scroll=Scrollbar(text)

text.configure(yscrollcommand=scroll.set)

text.place(x=10,y=200)

text.config(font=font1)

font1 = ('times', 12, 'bold')

uploadButton = Button(main, text="Upload ISCX Dataset", command=uploadDataset)

uploadButton.place(x=50,y=100)

uploadButton.config(font=font1)

preButton = Button(main, text="Dataset Preprocessing", command=DatasetPreprocessing)

preButton.place(x=250,y=100)

preButton.config(font=font1)

nbButton = Button(main, text="Standard CNN Model", command=runStandardCNN)

nbButton.place(x=450,y=100)

nbButton.config(font=font1)

rfButton = Button(main, text="Deep Parallel NIN Model", command=runNINCNN)

rfButton.place(x=660,y=100)

rfButton.config(font=font1)

graphButton = Button(main, text="Comparison Graph", command=graph)

graphButton.place(x=50,y=150)

graphButton.config(font=font1)

predictButton = Button(main, text="Traffic Classification using Encrypted Test Data", command=predict)

predictButton.place(x=270,y=150)

predictButton.config(font=font1)

main.config(bg='OliveDrab2')

main.mainloop()

**CHAPTER 10**

**RESULTS AND DISCUSSION**

**10.1 Implementation description**

This research develops a GUI application for performing encrypted network traffic classification using deep learning models. It imports various libraries required for the GUI application, including libraries for building and training deep learning models, handling GUI components, and performing data preprocessing. This implementation provides a user-friendly interface for uploading, preprocessing, training, and evaluating deep learning models for encrypted network traffic classification. The models used here are a standard CNN and a parallel deep NIN model. The application allows users to visualize the dataset, train the models, compare their performance, and classify encrypted network traffic based on the trained models. Below is the step-by-step process of implementing the proposed model:

1. Libraries Used

* tkinter: It provides the graphical user interface components.
* matplotlib: Used for creating visualizations like bar charts.
* numpy: For numerical operations.
* pandas: For data manipulation and analysis.
* seaborn: For creating visualizations like heatmaps.
* scikit-learn (sklearn): For machine learning tasks, such as data preprocessing, model training, and evaluation.
* keras: For deep learning tasks, such as load model, batch normalization, dense, activation, drop out, flatten etc.
* pickle: For serializing and deserializing Python objects.

2: GUI Components

* Main Window: The GUI window is created with a specific size and a title.
* Buttons: Several buttons are placed on the GUI for actions such as uploading a dataset, preprocessing the dataset, splitting the dataset, running an ensemble model, comparing performance, and predicting traffic types from test data.
* Text Box: A text box is provided for displaying messages and results to the user.
* File Dialogs: File dialogs are used for uploading datasets and saving the model.

3: Function Definitions

* uploadDataset(): Allows the user to upload a dataset, preprocesses it, and displays basic statistics.
* DatasetPreprocessing(): Preprocesses the loaded dataset, encodes non-numeric data, normalizes the data, and splits it into training and testing sets.
* runStandardCNN(): Builds and trains a standard Convolutional Neural Network (CNN) model using the loaded dataset.
* runNINCNN(): Builds and trains a parallel deep Network-in-Network (NIN) model using the loaded dataset.
* calculateMetrics(algorithm, testY, predict): Calculates and displays accuracy, precision, recall, F1 score, and confusion matrix for the given algorithm.
* graph(): Generates a bar graph comparing the accuracy, precision, recall, and F1 score of the standard CNN and the parallel deep NIN model.
* predict(): Allows the user to upload encrypted test data, classifies the traffic using the trained model, and displays the results.

4. Main Loop: The main.mainloop() function enters the main event loop of the tkinter application, where it waits for events to occur (button clicks, mouse actions, etc.) and responds to them by calling the appropriate functions.

**10.2 Dataset description**

This project considered ISCXVPN2016 dataset, which contains network flow data with various features related to network traffic. Each row in the dataset represents a network flow and includes the following columns:

* Flow ID: Unique identifier for the flow.
* Src IP: Source IP address of the flow.
* Src Port: Source port number.
* Dst IP: Destination IP address of the flow.
* Dst Port: Destination port number.
* Protocol: Network protocol used for the flow (e.g., TCP, UDP).
* Timestamp: Time when the flow occurred.
* Flow Duration: Duration of the flow.
* Total Fwd Packet: Total number of packets sent in the forward direction.
* Total Bwd Packets: Total number of packets sent in the backward direction.
* Total Length of Fwd Packet: Total length of packets sent in the forward direction.
* Total Length of Bwd Packet: Total length of packets sent in the backward direction.
* Fwd Packet Length Max: Maximum length of packets in the forward direction.
* Fwd Packet Length Min: Minimum length of packets in the forward direction.
* Fwd Packet Length Mean: Mean length of packets in the forward direction.
* Fwd Packet Length Std: Standard deviation of packet length in the forward direction.
* Bwd Packet Length Max: Maximum length of packets in the backward direction.
* Bwd Packet Length Min: Minimum length of packets in the backward direction.
* Bwd Packet Length Mean: Mean length of packets in the backward direction.
* Bwd Packet Length Std: Standard deviation of packet length in the backward direction.
* Flow Bytes/s: Flow bytes per second.
* Flow Packets/s: Flow packets per second.
* Flow IAT Mean: Mean inter-arrival time of the flow.
* Flow IAT Std: Standard deviation of inter-arrival time of the flow.
* Flow IAT Max: Maximum inter-arrival time of the flow.
* Flow IAT Min: Minimum inter-arrival time of the flow.
* Fwd IAT Total: Total inter-arrival time of forward packets.
* Fwd IAT Mean: Mean inter-arrival time of forward packets.
* Fwd IAT Std: Standard deviation of inter-arrival time of forward packets.
* Fwd IAT Max: Maximum inter-arrival time of forward packets.
* Fwd IAT Min: Minimum inter-arrival time of forward packets.
* Bwd IAT Total: Total inter-arrival time of backward packets.
* Bwd IAT Mean: Mean inter-arrival time of backward packets.
* Bwd IAT Std: Standard deviation of inter-arrival time of backward packets.
* Bwd IAT Max: Maximum inter-arrival time of backward packets.
* Bwd IAT Min: Minimum inter-arrival time of backward packets.
* Fwd PSH Flags: Number of times the PSH flag was set in forward packets.
* Bwd PSH Flags: Number of times the PSH flag was set in backward packets.
* Fwd URG Flags: Number of times the URG flag was set in forward packets.
* Bwd URG Flags: Number of times the URG flag was set in backward packets.
* Fwd Header Length: Length of the header in forward packets.
* Bwd Header Length: Length of the header in backward packets.
* Fwd Packets/s: Forward packets per second.
* Bwd Packets/s: Backward packets per second.
* Packet Length Min: Minimum length of all packets.
* Packet Length Max: Maximum length of all packets.
* Packet Length Mean: Mean length of all packets.
* Packet Length Std: Standard deviation of packet length.
* Packet Length Variance: Variance of packet length.
* FIN Flag Count: Number of packets with FIN flags.
* SYN Flag Count: Number of packets with SYN flags.
* RST Flag Count: Number of packets with RST flags.
* PSH Flag Count: Number of packets with PSH flags.
* ACK Flag Count: Number of packets with ACK flags.
* URG Flag Count: Number of packets with URG flags.
* CWE Flag Count: Number of packets with CWE flags.
* ECE Flag Count: Number of packets with ECE flags.
* Down/Up Ratio: Ratio of downlink to uplink traffic.
* Average Packet Size: Average size of packets.
* Fwd Segment Size Avg: Average segment size in the forward direction.
* Bwd Segment Size Avg: Average segment size in the backward direction.
* Fwd Bytes/Bulk Avg: Average number of bytes in bulk in forward packets.
* Fwd Packet/Bulk Avg: Average number of packets in bulk in forward packets.
* Fwd Bulk Rate Avg: Average bulk rate in forward packets.
* Bwd Bytes/Bulk Avg: Average number of bytes in bulk in backward packets.
* Bwd Packet/Bulk Avg: Average number of packets in bulk in backward packets.
* Bwd Bulk Rate Avg: Average bulk rate in backward packets.
* Subflow Fwd Packets: Number of packets in the forward subflow.
* Subflow Fwd Bytes: Number of bytes in the forward subflow.
* Subflow Bwd Packets: Number of packets in the backward subflow.
* Subflow Bwd Bytes: Number of bytes in the backward subflow.
* FWD Init Win Bytes: Initial window size in the forward direction.
* Bwd Init Win Bytes: Initial window size in the backward direction.
* Fwd Act Data Pkts: Number of forward packets with actual data.
* Fwd Seg Size Min: Minimum segment size in the forward direction.
* Active Mean: Mean time a flow was active before becoming idle.
* Active Std: Standard deviation of time a flow was active before becoming idle.
* Active Max: Maximum time a flow was active before becoming idle.
* Active Min: Minimum time a flow was active before becoming idle.
* Idle Mean: Mean time a flow was idle.
* Idle Std: Standard deviation of time a flow was idle.
* Idle Max: Maximum time a flow was idle.
* Idle Min: Minimum time a flow was idle.
* Traffic\_Type: Type of network traffic.
* Label: Classification label indicating the class of the network traffic flow with 11 classes.

**10.3 Results description**

Figure 1 shows a representation of the GUI application developed for encrypted network traffic classification. The GUI include buttons, input fields, and visualizations, allowing users to interact with the application and perform tasks related to network traffic classification. Figure 2 depicts the GUI interface after loading the ISCX network traffic dataset. The dataset contains 141,530 rows and 85 columns. In the GUI, this dataset is displayed, providing users with an overview of the data they are working with. This view includes a table-like structure showing the sample dataset’s rows and columns.

A screenshot of a computer

Description automatically generated

Figure 1: GUI application of encrypted network traffic classification using deep and parallel NIN model.

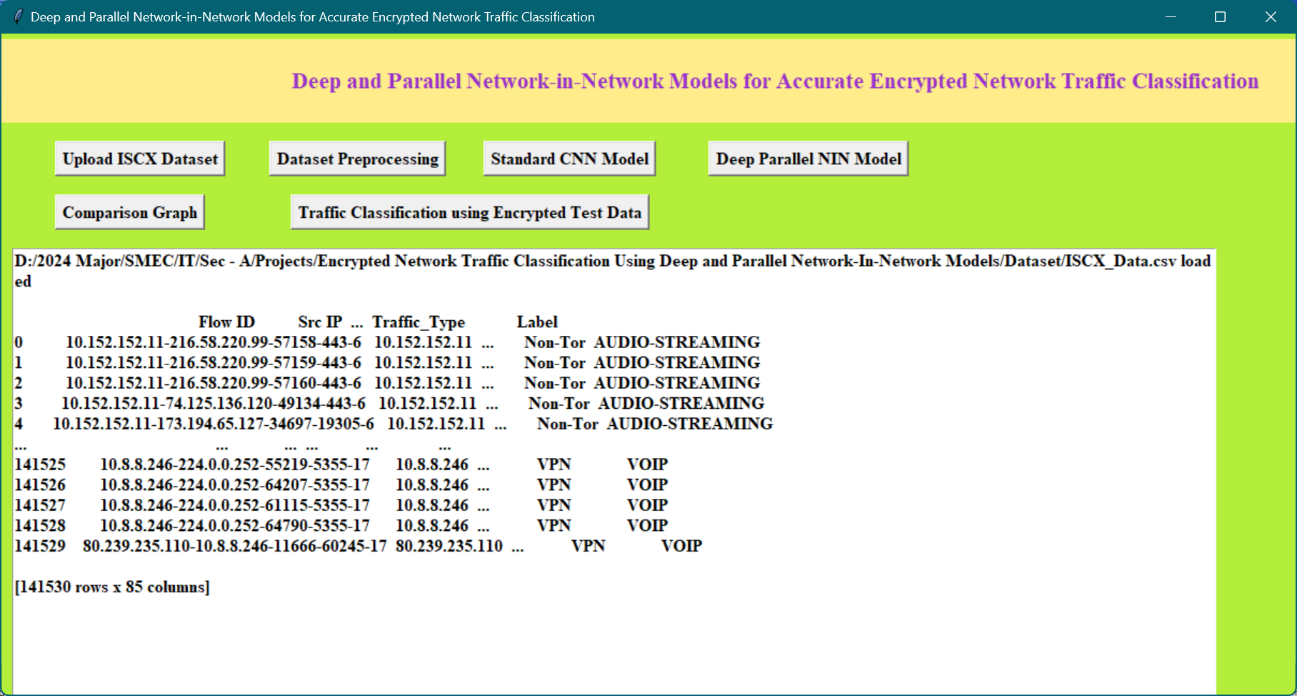


Figure 2: GUI application after loading the ISCX network traffic dataset of 141530 rows, and 85 columns.

Figure 3 represents a count plot, a type of bar chart, displaying the distribution of network traffic categories present in the loaded dataset. Each category is shown on the X-axis, and the corresponding count or frequency of each category is represented on the Y-axis. This visualization provides insights into the dataset's class distribution, which is crucial for understanding the balance or imbalance among different traffic categories.

Figure 4 shows the GUI interface after performing data preprocessing tasks such as normalization, label encoding, and standard scaling. These preprocessing steps are essential in preparing the dataset for training machine learning models. Normalization ensures that all features have a consistent scale, label encoding converts categorical data into numerical values, and standard scaling standardizes the feature values to have a mean of 0 and a standard deviation of 1.

Figure 5(a) displays the confusion matrix generated by evaluating the predictions of the existing Standard CNN model. A confusion matrix is a table used to evaluate the performance of a classification algorithm. It shows the true positive, true negative, false positive, and false negative values, allowing a detailed analysis of the model's performance on each class. Figure 5(b) displays the confusion matrix generated by evaluating the predictions of the proposed Parallel Deep NIN model. Like the confusion matrix of the existing Standard CNN model, this matrix provides detailed information about the performance of the Parallel Deep NIN model, allowing a comparative analysis of the two models.

A screenshot of a computer

Description automatically generated

Figure 3: Count plot of network traffic category.

A screenshot of a computer

Description automatically generated

Figure 4: GUI application after performing data preprocessing, normalization, label encoding, and standard scaling operations.

A screenshot of a computer

Description automatically generated

(a)

A screenshot of a computer

Description automatically generated

(b)

Figure 5: Confusion matrices. (a) existing standard CNN model. (b) proposed parallel deep NIN model.

A screenshot of a computer

Description automatically generated

Figure 6: Performance comparison of obtained quality metrics using existing standard CNN and proposed parallel deep NIN models.

Figure 6 presents a visual comparison of the quality metrics obtained from the existing Standard CNN model and the proposed Parallel Deep NIN model. These metrics includes accuracy, precision, recall, and F1-score. A bar chart visualization method is used to clearly show the differences in performance between the two models, indicating which model performs better in each metric. Figure 7 shows the GUI interface displaying a sample of predicted traffic classification results on encrypted test data. It includes the input data or features and the corresponding predicted traffic category, allowing users to see how the trained model classifies specific instances of encrypted network traffic.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Figure 7: GUI application displaying the sample predicted traffic classification on encrypted test data.

**CHAPTER 11**

**CONCLUSION AND FUTURE SCOPE**

This work has successfully developed a user-friendly graphical interface for encrypted network traffic classification. By implementing and comparing two significant models – the traditional Standard CNN and the more complex Parallel Deep NIN model – the application provides valuable insights into network traffic patterns. Through rigorous evaluation, it was evident that the Parallel Deep NIN model outperformed the Standard CNN, showcasing superior accuracy, precision, recall, and F1-score. This achievement not only validates the effectiveness of advanced deep learning techniques but also highlights the potential impact of sophisticated models in network traffic analysis. The application's user-friendly interface enhances accessibility for network administrators and analysts, allowing them to leverage these advanced techniques without extensive programming expertise. Visualizations, including bar charts and confusion matrices, have been incorporated to facilitate intuitive interpretation of the classification results. This research’s core contribution lies in its ability to accurately classify network traffic, making it applicable for various security and optimization tasks within network management.

**Future Scope**

There are several avenues for further exploration and enhancement of this project’s capabilities. Firstly, fine-tuning and optimizing the Parallel Deep NIN model's hyperparameters and architecture could potentially lead to even more impressive results. Additionally, diversifying the dataset to encompass a broader range of network traffic patterns would improve the models' ability to generalize to real-world scenarios. Integrating real-time data feeds from network devices could facilitate continuous learning, enabling the models to adapt to new traffic patterns over time. By continually researching and implementing advancements in these areas, the application can evolve into an indispensable tool for network security and management professionals, contributing significantly to the field of network traffic analysis.

**References**

1. Flurry Analytics, ComScore, Pandora, Facebook, NetMarketShare. 2015. Available online: [**https://www.marketingcharts.com/industries/media-and-entertain-ment58693/attachment/flurry-share-time-spent-mobile-devices-sept**](https://www.marketingcharts.com/industries/media-and-entertain-ment58693/attachment/flurry-share-time-spent-mobile-devices-sept) (accessed on 15 December 2021).
2. Al-Mejibli, I.S.; Alharbe, N.R. Analyzing and evaluating the security standards in wireless network: A review study. *Iraqi J. Comput. Inform.* **2020**, *46*, 32–39.
3. Taylor, V.F.; Spolaor, R.; Conti, M.; Martinovic, I. Robust smartphone app identification via encrypted network traffic analysis. *IEEE Trans. Inf. Forensics Secur.* **2017**, *13*, 63–78.
4. Aceto, G.; Ciuonzo, D.; Montieri, A.; Pescape, A. Mobile Encrypted Traffic Classification Using Deep Learning. In Proceedings of the Network Traffic Measurement and Analysis Conference (TMA), Vienna, Austria, 26–29 June 2018; pp. 1–8.
5. Wang, W.; Zhu, M.; Wang, J.; Zeng, X.; Yang, Z. End-to-end encrypted traffic classification with one-dimensional convolution neural networks. In Proceedings of the IEEE International Conference on Intelligence and Security Informatics (ISI), Beijing, China, 22–24 July 2017; pp. 43–48.
6. Pathmaperuma, M.H.; Rahulamathavan, Y.; Dogan, S.; Kondoz, A.M. In-App Activity Recognition from Wi-Fi Encrypted Traffic. In *Intelligent Computing. SAI 2020*; Arai, K., Kapoor, S., Bhatia, R., Eds.; Advances in Intelligent Systems and Computing; Springer: Cham, Switzerland, 2020; Volume 1228.
7. Zhang, F.; He, W.; Liu, X.; Bridges, P.G. Inferring users’ online activities through traffic analysis. In Proceedings of the ACM Conference on Wireless Network Security, Hamburg, Germany, 14–17 June 2011.
8. Conti, M.; Mancini, L.V.; Spolaor, R.; Verde, N.V. Analyzing Android encrypted network traffic to identify user actions. *IEEE Inf. Forensics Secur.* **2016**, *11*, 114–125.
9. Park, K.; Kim, H. Encryption is not enough: Inferring user activities on KakaoTalk with traffic analysis. In *16th International Workshop on Information Security Applications*; Kim, H.-W., Choi, D., Eds.; ser. WISA ’15; Springer: Berlin/Heidelberg, Germany, 2015; pp. 254–265.
10. Saltaformaggio, B.; Choi, H.; Johnson, K.; Kwon, Y.; Zhang, Q.; Zhang, X.; Xu, D.; Qian, J. Eavesdropping on fine-grained user activities within smartphone apps over encrypted network traffic. In *WOOT’16 Proceedings of the 10th USENIX Conference on Offensive Technologies*; USENIX Association: Berkeley, CA, USA, 2016; pp. 69–78.
11. Taylor, V.F.; Spolaor, R.; Conti, M.; Martinovic, I. AppScanner: Automatic Fingerprinting of Smartphone Apps from Encrypted Network Traffic. In Proceedings of the IEEE European Symposium on Security and Privacy (Euro S&P), Saarbruecken, Germany, 21–24 March 2016; pp. 439–454.
12. Lotfollahi, M.; Zade, R.S.H.; Siavoshani, M.J.; Saberian, M. Deep packet: A novel approach for encrypted traffic classification using deep learning. *Soft Comput.* **2020**, *24*, 1999–2012.
13. Draper-Gil, G.; Lashkari, A.H.; Mamun, M.S.I.; Ghorbani, A. Characterization of encrypted and VPN traffic using time-related features. In Proceedings of the ICISSP, Rome, Italy, 19–21 February 2016; pp. 407–414.
14. Lopez-Martin, M.; Carro, B.; Sanchez-Esguevillas, A.; Lloret, J. Net-work traffic classifier with convolutional and recurrent neural networks for internet of things. *IEEE Access* **2017**, *5*, 18042–18050.
15. Wang, P.; Ye, F.; Chen, X.; Qian, Y. DataNet: Deep learning based en-crypted network traffic classification in SDN home gateway. *IEEE Access* **2018**, *6*, 55380–55391.
16. Bayat, N.; Jackson, W.; Liu, D. Deep Learning for Network Traffic Classification. *arXiv* **2021**, arXiv:2106.12693.
17. Aceto, G.; Ciuonzo, D.; Montieri, A.; Pescapé, A. MIMETIC: Mobile encrypted traffic classification using multimodal deep learning. *Comput. Netw.* **2019**, *165*, 106944.
18. Aceto, G.; Ciuonzo, D.; Montieri, A.; Pescapé, A. DISTILLER: Encrypted traffic classification via multimodal multitask deep learning. *J. Netw. Comput. Appl.* **2021**, *183*, 102985.
19. Liu, C.; He, L.; Xiong, G.; Cao, Z.; Li, Z. FS-Net: A Flow Sequence Network for Encrypted Traffic Classification. In Proceedings of the IEEE Conference on Computer Communications (INFOCOM), Paris, France, 29 April–2 May 2019; pp. 1171–1179.
20. Tong, V.; Tran, H.-A.; Souihi, S.; Mellouk, A. A novel QUIC traffic classifier based on convolutional neural networks. In Proceedings of the 2018 IEEE Global Communications Conference (GLOBECOM), Abu Dhabi, United Arab Emirates, 9–13 December 2018; pp. 1–6.