

Шифр "IDSTOCN"

СИСТЕМА ВИЯВЛЕННЯ ВТОРГНЕНЬ У КОМП'ЮТЕРНУ МЕРЕЖУ

Анотація

Метою наукової роботи під шифром «“IDSTOCN”» є підвищення якості ідентифікації стану комп’ютерних систем шляхом розробки та удосконалення методів виявлення вторгнень в комп’ютерні мережі.

Об’єкт дослідження – процес виявлення вторгання в комп’ютерну мережу з використанням методів машинного та глибокого навчання.

Предмет дослідження – методи попередньої обробки та класифікації мережевих даних на основі технології глибокого навчання.

Досліджено методи та засоби виявлення вторгнень у комп’ютерній мережі. Проаналізовано методи попередньої обробки та класифікації даних.

У якості методів класифікації розглянуто: методи машинного навчання Support Vector Machine, K-Nearest Neighbors та моделі глибокого навчання Vision Transformer (Vit), Vision Transformer For Small-Size Datasets (ViTSD).

Розроблено програмне забезпечення мовою Python у середовищі Google Colab, реалізовано попередню обробку даних та налаштування класифікаторів. Виконано тестування розроблених класифікаторів та аналіз результатів класифікації.

За результатами дослідження запропоновано:

1. Процедуру зменшення кореляції вихідних даних за рахунок рекурсивного використання методу головних компонент (PCA), що дозволило зменшити об’єм вихідних даних та час навчання моделі.

2. Процедуру перетворення табличних вихідних даних у спеціальний формат зображень, необхідний для роботи моделей глибокого навчання Vision Transformer (ViT) та Vision Transformer for Small-size Datasets (ViTSD).

3. Метод виявлення вторгнень в комп’ютерні мережі, який відрізняється від відомих використанням алгоритму глибокого навчання Vision Transformer for Small-size Datasets (ViTSD) та спеціальної процедури зменшення кореляції вихідних даних, що дозволило підвищити точність ідентифікації.

Робота містить 63 сторінки тексту, серед них: 17 рисунки, 4 таблиці, список використаних джерел з 27 найменувань.

Ключові слова: машинне навчання, класифікація даних, попередня обробка даних, кореляція даних, глибоке навчання, моделі-трансформери.

ЗМІСТ

ПЕРЕЛІК ПОЗНАК ТА СКОРОЧЕНЬ	1
ВСТУП.....	2
1 ОГЛЯД СИСТЕМ ВИЯВЛЕННЯ ВТОРГНЕНЬ.....	4
1.1 Класифікація систем виявлення вторгнень.....	4
1.2 Методи виявлення вторгнень.	4
1.3 Структура систем виявлення вторгнень.....	4
1.4 Методи навчання сучасних СВВ.....	5
1.5 Постановка науково-технічної проблеми і задач досліджень	5
1.6 Висновки за розділом.	6
2 ОБГРУНТУВАННЯ ВИБОРУ МЕТОДІВ МАШИННОГО ТА ГЛИБОКОГО НАВЧАННЯ.....	7
2.1 Визначення машинного та глибокого навчання.....	7
2.2 Трансформер	8
2.2.1 Основні складові моделі Трансформер.....	8
2.2.2 Архітектура моделі Трансформер	9
2.2.3 Механізм уваги	10
2.2.4 Використання моделей-трансформерів в системах виявлення вторгнень у комп'ютерну мережу	12
2.3 Vision transformer	13
2.4 Vision transformer for small-size data sets	14
2.5 Висновки за розділом.	15
3 РОЗРОБКА ТА ОЦІНКА ЕФЕКТИВНОСТІ МЕТОДІВ ІДЕНТИФІКАЦІЇ СТАНУ МЕРЕЖ ТА ПРОГРАМНІ КОМПОНЕНТИ.....	17
3.1 Вибір набору даних для моделювання.	17
3.2 Попередня обробка даних.....	17
3.2.1 Метод головних компонент.....	17

3.2.2	Розробка процедури зменшення кореляції набору даних	18
3.3	Класичні моделі машинного навчання	20
3.4	Моделювання ViT та ViTSD моделей	20
3.4.1	Підготовка даних	20
3.4.2	Результати моделювання	21
3.5	Порівняльний аналіз результатів роботи моделей виявлення вторгнень на основі машинного та глибокого навчання.....	22
3.6	Висновки за розділом	23
ВИСНОВКИ.....		25
СПИСОК ДЖЕРЕЛ ІНФОРМАЦІ.....		26
Додаток А. Фрагменти програми ідентифікації стану комп'ютерної мережі з використанням алгоритмів глибокого навчання ViT та ViTSD на даних з попередньою обробкою		30

ПЕРЕЛІК ПОЗНАК ТА СКОРОЧЕНЬ

IDS – Intrusion Detection System
NIDS – Network Intrusion Detection Systems
HIDS – Host-based Intrusion Detection Systems
ML – Machine Learning
DL – Deep Learning
ІНМ – Штучна Нейронна Мережа
CNN – Convolutional Neural Network
RNN – Recurrent Neural Network
ViT – Vision Transformer
ViTSD – Vision Transformer for Small-Size Datasets
PCA – Principal Component Analysis
ПЗ – Програмне Забезпечення
СВВ – Система Виявлення Вторинень
SVM – Support Vector Machine
SVC – Support Vector Classification
SVR – Support Vector Regression
kNN – k-Nearest Neighbor
NLP – Natural Language Processing
BERT – Bidirectional Encoder Representations from Transformers
GPT – Generative Pre-trained Transformer
GPU – Graphics Processing Unit
LSTM – Long Short-Term Memory
GRU – Gated Recurrent Units
FFN – Feed-Forward Networks
SPT – Shifted Patch Tokenization
LSA – Locality Self-Attention

ВСТУП

Системи виявлення вторгнень (IDS) відіграють ключову роль у процесі захисту мережі та точної ідентифікації атак на систему. Зокрема, категорія мережевих IDS (NIDS) відстежує та аналізує мережевий трафік на різних рівнях, щоб виявити будь-які зловмисні та аномальні дії, які можуть бути частиною атаки.

Системи виявлення вторгнень в комп'ютерні мережі базуються на використанні методів машинного навчання (МН). Одним із найбільш популярних напрямків МН є методи глибокого навчання. Глибоке навчання представляє собою сектор в галузі машинного навчання, що ґрунтується на архітектурі штучної нейронної мережі (ШНМ). В сфері глибокого навчання найбільш поширеними архітектурами є нейронні мережі прямого зв'язку, згорткові нейронні мережі (CNN), рекурентні нейронні мережі (RNN) та трансформер (Transformer).

Разом із тим, збільшення кількості вторгнень в комп'ютерні мережі потребує удосконалення існуючих та розробку нових методів їх ідентифікації.

Метою роботи є підвищення якості ідентифікації стану комп'ютерних систем шляхом розробки та удосконалення методів виявлення вторгнень в комп'ютерні мережі.

Об'єкт дослідження – процес виявлення втручання в комп'ютерну мережу з використанням методів машинного та глибокого навчання.

Предмет дослідження – методи попередньої обробки та класифікації мережевих даних на основі технології глибокого навчання.

Методи дослідження. Теоретичні дослідження і методика рішення базується на наступних методах машинного навчання: Support Vector Machine, K-Nearest Neighbors та моделях глибокого навчання Vision Transformer (ViT), Vision Transformer For Small Datasets (ViTSD) з використанням методів попередньої обробки даних.

Наукова новизна одержаних результатів полягає в наступному:

1. Запропоновано процедуру зменшення кореляції вихідних даних за рахунок рекурсивного використання методу головних компонент (PCA), що дозволило зменшити об'єм вихідних даних та час навчання моделі.
2. Запропоновано процедуру перетворення табличних вихідних даних у спеціальний формат зображень, необхідний для роботи моделей глибокого навчання Vision Transformer (ViT) та Vision Transformer for Small-size Datasets (ViTSD).
3. Запропоновано метод виявлення вторгнень в комп'ютерні мережі, який відрізняється від відомих використанням алгоритму глибокого навчання Vision Transformer for Small-size Datasets (ViTSD) та спеціальної процедури зменшення кореляції вихідних даних, що дозволило підвищити точність ідентифікації.

Практичне значення одержаних результатів роботи, полягає в тому, що: сформовано процедуру попередньої обробки даних, яка сфокусована що на зменшення кореляції вихідних даних; запропоновано процедуру перетворення табличних вихідних даних у спеціальний формат зображень; розроблено програмну модель виявлення вторгнень у комп'ютерну мережу на основі алгоритму Vision Transformer For Small-size Datasets. Отримані в межах даної роботи результати, є науково-методичною базою для розробки відповідних алгоритмів, мають програмні реалізації та можуть бути використані в навчальному процесі кафедри.

Галузь застосування розробленого методу, полягає у його використанні у якості додаткового засобу в системах виявлення вторгнень у комп'ютерну мережу.

1 ОГЛЯД СИСТЕМ ВИЯВЛЕННЯ ВТОРГНЕНЬ

1.1 Класифікація систем виявлення вторгнень

Система виявлення вторгнень – це програмний або апаратний засіб, призначений для виявлення фактів несанкціонованого доступу в комп'ютерну систему або мережу.

Системи виявлення вторгнень (IDS) діляться на:

- системи виявлення вторгнень у мережу (network intrusion detection systems, NIDS), які аналізують весь трафік даних у мережі щоб виявити спроби втручання.
- системи виявлення вторгнень засновані на аналізі хостів (host-based intrusion detection systems, HIDS), які виявляють зловмисні дії шляхом моніторингу системних файлів, реєстраційних файлів і інших джерел інформації.

1.2 Методи виявлення вторгнень.

- Аналіз сигнатур – це метод виявлення вторгнень, який порівнює трафік мережі з базою даних відомих атак.
- Аналіз протоколів – це метод виявлення вторгнень, який аналізує трафік мережі на наявність відхилень від нормального поведінки.
- Виявлення вторгнень на основі аномалій – це системи, які виявляють атаки, аналізуючи трафік мережі на наявність відхилень від нормальної поведінки.

1.3 Структура систем виявлення вторгнень

Сучасна структура СВВ включає в собі такі підсистеми:

- підсистема збору інформації про систему, яка підлягає захисту;
- підсистема пошуку атак та вторгнень у систему;
- підсистема представлення даних для контролю системи в режимі реального часу.

1.4 Методи навчання сучасних СВВ

Сучасні системи виявлення вторгнень застосовують дві основні групи методів в залежності від умов параметризації системи та процесу навчання.

Виявлення аномалії за методами контрольованого навчання («навчання з учителем») представлені такими системами:

- W&S. Метод виявлення втручання – це моделювання правил.
- IDES, NIDES, EMERLAND, JiNao, HayStack. Метод виявлення втручання – це описова статистика.
- Hyperview. Метод виявлення втручання – це нейронні мережі.

Виявлення аномалії за методами неконтрольованого навчання («навчання без учителя») представлені такими системами:

- DPEM, JANUS, Bro. Метод виявлення втручання – моделювання множини станів.
- MIDAS, NADIR, Haystack, NSM. Метод виявлення втручання – описова статистика.

1.5 Постановка науково-технічної проблеми і задач досліджень

1) Обґрунтувати вибір методів побудови моделей для виявлення втручання в комп'ютерну мережу.

2) Обґрунтувати вибір мови програмування та середовища розробки моделі. Розробити програмне забезпечення.

3) Виконати попередню обробку даних. Сформувані тренувальну та тестову вибірки.

4) Побудувати моделі, виконати їх налаштування.

5) Оцінити якість роботи моделей.

6) Проаналізувати результати дослідження, зробити висновки.

1.6 Висновки за розділом

У першому розділі визначено поняття системи виявлення вторгнень (СВВ). Розглянуто класифікацію СВВ та методи, які вони використовують. Досліджено структуру та методи навчання СВВ. Наведені приклади існуючих СВВ та принципи їх роботи. Визначено їх переваги та недоліки.

Оскільки метою даної роботи є підвищення точності та оперативності виявлення вторгнень у комп'ютерні мережі, а вихідні дані є маркованими, то для подальшого дослідження прийнято рішення розглянути методи класифікації даних із застосуванням сучасних розробок в галузі штучних нейронних мереж, з використанням технології «навчання з учителем».

Виконано постановку науково-технічного завдання розробки моделі виявлення вторгнень в комп'ютерну мережу та сформульовано завдання дослідження.

2 ОБГРУНТУВАННЯ ВИБОРУ МЕТОДІВ МАШИННОГО ТА ГЛИБОКОГО НАВЧАННЯ

2.1 Визначення машинного та глибокого навчання

Машинне навчання (Machine Learning, ML) – це галузь штучного інтелекту, яка досліджує розробку алгоритмів і моделей, які дають комп'ютерам можливість «навчатися» на основі даних та виконувати завдання без явного програмування.

Традиційні алгоритми машинного навчання, такі як дерева рішень [1], випадковий ліс [2], K найближчих сусідів [3] та байєсівські методи, були успішно використані в системах виявлення вторгнень протягом багатьох років. Однак вони можуть бути недостатніми для обробки великих обсягів високорозмірних даних, що генеруються сучасними мережами.

Глибоке навчання (DL) – це передова область машинного навчання (ML) з потужними аналітичними можливостями. DL забезпечує кращу продуктивність з великими даними, ніж ML.

Моделі виявлення вторгнень на основі глибокого навчання зазвичай використовують такі моделі, як згорткові нейронні мережі (CNN)[4], рекурентні нейронні мережі (RNN)[5], довгострокові мережі короткострокової пам'яті (LSTM)[6] та генеративні супротивні мережі (GAN).

Трансформер був вперше застосований для задач обробки природної мови. Він не використовує традиційні структури рекурентної нейронної мережі (RNN) та згорткової нейронної мережі (CNN), а лише механізм уваги.

На відміну від навчання RNN яке є ітеративним та послідовним, навчання Трансформер є паралельним, що дозволяє одночасно навчати всі функції. Це значно підвищує обчислювальну ефективність та зменшує час навчання моделі. Тому подальші дослідження будуть пов'язані з дослідженням можливості використання моделей трансформерів для виявлення вторгнень у комп'ютерні системи.

2.2 Трансформер

Трансформер – модель нейронної мережі яка спеціалізується на процесі глибинного навчання з використанням механізму «уваги» до кожного елементу в отриманому наборі вхідних даних

Модель трансформера (Transformer model) була представлена у статті «Attention Is All You Need» Філіпом Василевським та інженерами з Google Research у 2017 році.

2.2.1 Основні складові моделі Трансформер

Основними складовими моделі трансформера є:

Механізм уваги (Attention). Увага – це ключовий елемент архітектури трансформера. Він дозволяє моделі враховувати взаємозв'язки між словами або токенами у вхідних даних, що робить його особливо ефективним для обробки послідовних даних, таких як текст.

Паралельна обробка. Трансформери здатні виконувати обчислення паралельно, що робить їх швидкими та ефективними для навчання на багатоядерних процесорах та графічних процесорах (GPU).

Багатозадачність. Моделі трансформера можна налаштувати на виконання різних завдань NLP без істотних змін архітектури. Це робить їх гнучкими та універсальними.

Використання масок. У трансформерах використовуються маски, щоб керувати доступом моделі до різних частин вхідних даних. Наприклад, для машинного перекладу маска використовується для вказівки, які слова доступні на поточному етапі та які слід ігнорувати.

Багатошаровість. Моделі трансформери, зазвичай, складаються з кількох шарів уваги та прямих проходів, що дозволяє їм вивчати складніші залежності в даних.

Механізми уваги стали невід'ємною складовою моделей для обробки послідовностей та трансдукції в різних завданнях. Вони дозволяють моделювати

залежності без обмежень від відстані між символами у вхідних або вихідних послідовностях [7, 8]. Модель «Трансформер» дозволяє значно більше паралелізації і може досягти нового рівня в якості перекладу [9].

2.2.2 Архітектура моделі Трансформер

Більшість конкурентоздатних моделей трансдукції послідовностей мають структуру з кодером і декодером [7, 10]. Модель «Трансформер» слідує цій загальній архітектурі, використовуючи послідовно розташовані механізми самоуваги та повнозв'язні шари як для кодера, так і декодера (Рис.2.1).

Кодер як і декодер складається зі стеку $N = 6$ однакових шарів. Кожен шар має два підшари. Перший шар – це механізм багаторівневої самоуваги (multi-head self-attention), а другий – звичайний повнозв'язний шар. У декодера є ще третій підшар, який застосовує багаторівневу увагу (multi-head attention) до виходу стеку кодера.

Використовується залишкове з'єднання (Residual Connection) [11] навколо кожного з двох підшарів, за яким слідує нормалізація шару [12]. Іншими словами, вихід кожного підшару може бути знайдено як

$$\text{LayerNorm}(x + \text{Sublayer}(x)), \quad (2.1)$$

де x – вхідні дані, LayerNorm – функція нормалізації даних, $\text{Sublayer}(x)$ – це функція, реалізована самим підшаром.

Також модифікується підшар самоуваги в стеку декодера, щоб запобігти потраплянню інформації з наступних позицій.

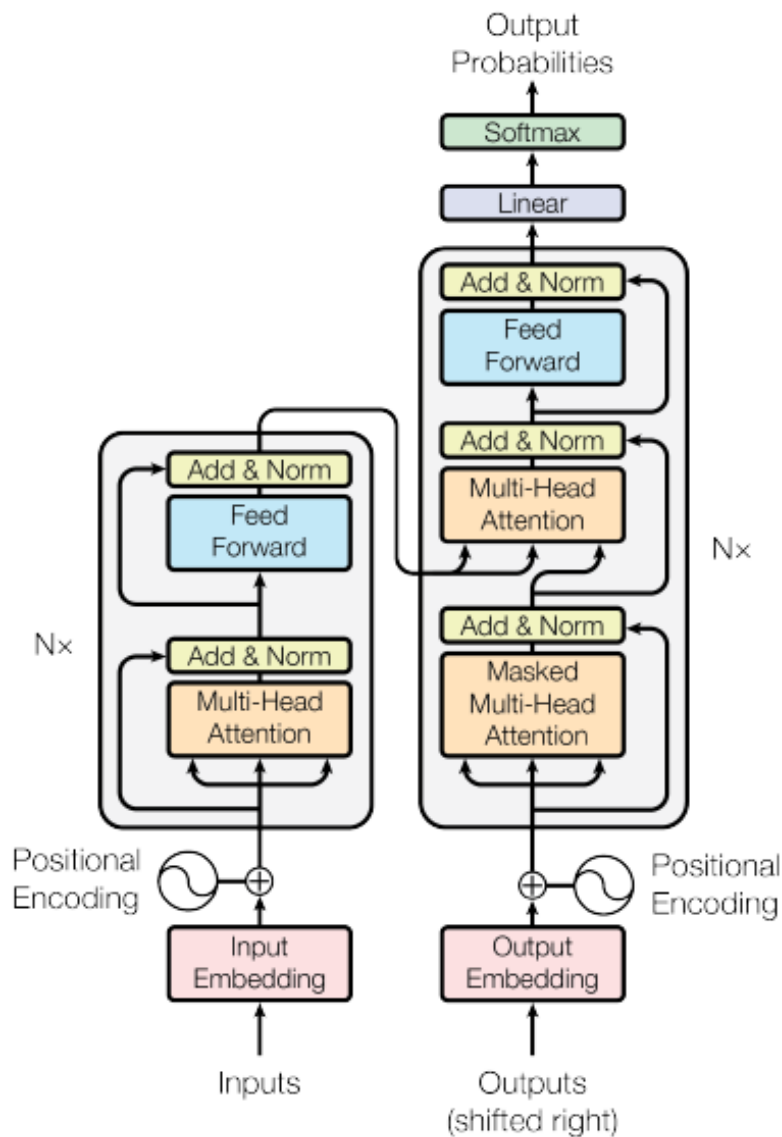


Рисунок 2.1 – Архітектура моделі «Трансформер». Ліворуч кодер, праворуч декодер

Крім підшарів уваги, кожен з шарів у кодері та декодері містить повнозв'язну мережу прямого розповсюдження (Position-wise Feed-Forward Networks, FFN).

2.2.3 Механізм уваги

Механізм уваги в моделях трансформерах функціонує наступним чином:

1. Спочатку кожен елемент вхідної послідовності (наприклад, слово або маркер) перетворюється у векторне представлення за допомогою технології

embedding – співставленню елементів у континуальному просторі безперервним векторам.

2. Далі для кожного елемента вхідної послідовності вираховуються ваги, які визначають, наскільки кожен елемент вхідних даних важливий для даного контексту.

3. Для кожного елемента послідовності виконується зведене підсумування, де кожен елемент множиться на власну увагу, і результати підсумовуються.

4. Отриманий після зваженого підсумовування контекстний вектор (Scaled Dot-Product Attention), є стислим представлення вхідних даних та враховує їх важливість для цього контексту.

Виділяють такі алгоритми формування функції уваги:

- Увага масштабованого скалярного добутку (Scaled Dot-Product Attention)
- Багаторівнева увага (multi-head attention).

Замість використання однієї функції уваги з ключами, значеннями та запитами розміром d_{model} , лінійно проектуються запити, ключі та значення h разів за допомогою різних навчених лінійних проєкцій до розмірів d_k , d_k та d_v , відповідно. Над кожною з цих проєктованих версій запитів, ключів і значень виконується функція уваги паралельно, отримуючи вихідні значення розміром d_v . Ці значення конкатенуються і знову проєктуються, що призводить до отримання кінцевих значень.

Багаторівнева увага дозволяє моделі спільно аналізувати інформацію з різних підпросторів, представлення в різних позиціях.

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_h)W^0 \quad (2.2)$$

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \quad (2.3)$$

де проекції – це матриці параметрів $W_i^Q \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{model} \times d_v}$ та $W^O \in \mathbb{R}^{hd_v \times d_{model}}$ (Рис. 2.2)

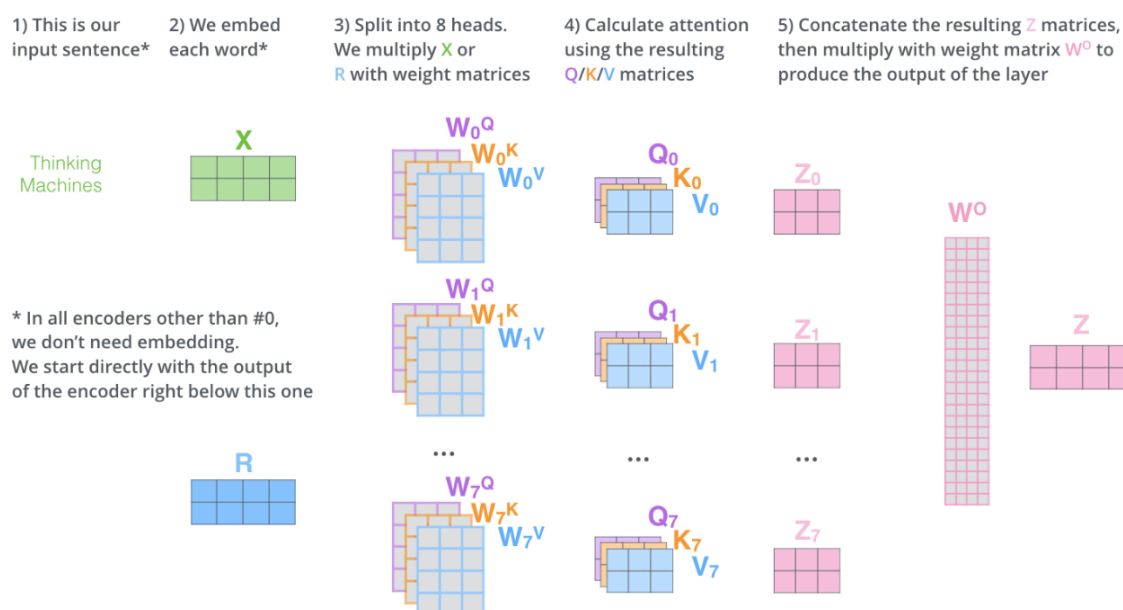


Рисунок 2.2 – Загальна схема роботи блоку багаторівневої уваги

2.2.4 Використання моделей-трансформерів в системах виявлення вторгнень у комп'ютерну мережу

Моделі трансформери можуть використовуватися в системах виявлення вторгнень у комп'ютерну мережу (Intrusion Detection Systems, IDS) для обробки та аналізу даних про мережевий трафік і виявлення потенційних аномалій або вторгнень. Вони можуть застосувати в таких системах: аналіз мережевого трафіку, виявлення аномалій, ідентифікація атак, прогнозування загроз, сегментація трафіку, аналіз журналів і подій, прогнозування загроз, покращення точності та швидкості дії.

Трансформери можуть підвищити точність виявлення загроз і знизити долю помилок завдяки своїм можливостям аналізу складних залежностей у даних. Крім того, використання паралельної обробки збільшує швидкість аналізу трафіку.

2.3 Vision transformer

Для застосування моделі Transformer до зображень була розроблена модель Vision Transformer (ViT) [13]. Яка працює наступним чином:

- зображення розбивають на патчі фіксованого розміру;
- застосовують ембедінги для перетворення патчів на вектори;
- додають позиційне кодування;
- подають отриману послідовність векторів на стандартний кодувальник Transformer;
- щоб виконати класифікацію, використовують стандартний підхід додавання додаткового навчального «токену класифікації» до послідовності.

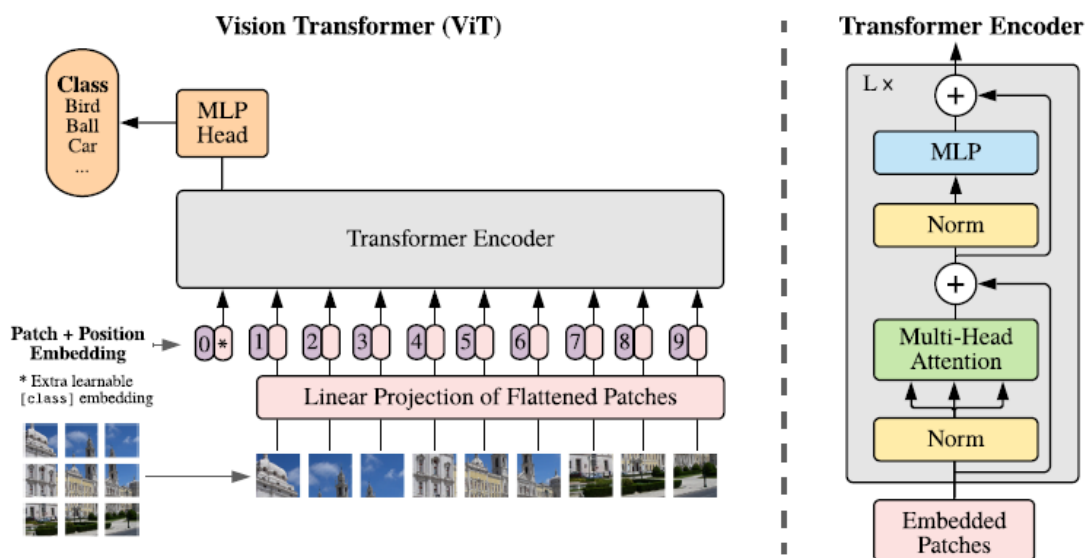


Рисунок 2.3 – Загальний опис моделі

При навчанні на середніх за розміром наборах даних, ViT дає скромну точність. Трансформери більш чутливі до змін в розташуванні об'єктів на зображенні, і вони можуть вимагати більше даних для навчання.

Однак ViT можна навчити на великому наборі даних, а потім використовувати для класифікації зображень на інших наборах даних без необхідності заново навчати всю модель.

2.4 Vision transformer for small-size data sets

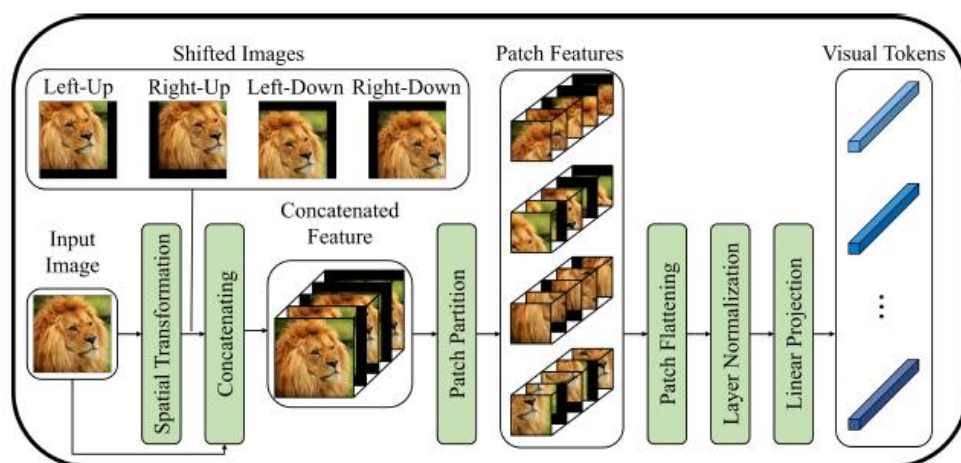
Існує дві проблеми, які знижують локальне індуктивне упередження і обмежують продуктивність ViT.

Перша проблема - це погана токенизація. ViT розділяє зображення на патчі рівного розміру, які не перекриваються, і потім лінійно проектує кожен патч у візуальний токен. Однак цей метод не враховує локальні відносини між патчами та інваріантний до перестановки патчів.

Друга проблема - це слабкий механізм уваги. Проблема поганого механізму уваги в ViT полягає в тому, що він не може локально звертати увагу на важливі візуальні токени.

Автори статті «Vision Transformer for Small-Size Datasets» [14] пропонують два рішення для ефективного покращення локальної індуктивної упередженості ViT для навчання з нуля на невеликих наборах даних.

Перше – це метод токенизації зображень зі зміщеними патчами (Shifted Patch Tokenization, SPT). SPT працює шляхом токенизації просторово зміщених зображень разом із вхідним зображенням. Це дозволяє ViT бачити ширшу область зображення та вловлювати більше просторових зв'язків між пікселями.



(a) Shifted Patch Tokenization

Рисунок 2.4 – Схема Shifted Patch Tokenization

Друге – локальна увага (Locality Self-Attention, LSA). LSA працює шляхом виключення власних токенів з обчислення балів уваги.

LSA також застосовує навчальну температуру до функції softmax. Це дозволяє регулювати піки розподілу балів уваги, що може допомогти ViT краще фокусуватися на локальних токенах.

Ядро LSA (Locality Self-Attention) – це діагональне маскування та шкала навчальної температури.

2.5 Висновки за розділом

В цьому розділі були розглянуті такі моделі машинного (SVM, KNN) та глибокого навчання (Transformer, ViT, ViTSD). Визначено їх переваги та недоліки.

Оскільки функціонування комп'ютерних мереж характеризується великим обсягом даних, то в сучасних умовах стрімкого збільшення кількості та якості кібератак, потрібно використовувати найсучасніші методи та підходи для виявлення вторгнень в комп'ютерну мережу. В сфері глибокого навчання найсучаснішою архітектурою є модель Трансформер. Трансформери не вимагають послідовної обробки вихідних даних. Завдяки цьому вони ефективніше розпаралелюються, що збільшує швидкість навчання моделі. Крім того, здатність фіксувати довготривалі залежності та ефективно обробляти послідовні дані робить їх дуже адаптивними та ефективними для вирішення різноманітних завдань.

3 РОЗРОБКА ТА ОЦІНКА ЕФЕКТИВНОСТІ МЕТОДІВ ІДЕНТИФІКАЦІЇ СТАНУ МЕРЕЖ ТА ПРОГРАМНІ КОМПОНЕНТИ

3.1 Вибір набору даних для моделювання.

У цій роботі у якості вихідних даних використано набір UNSW-NB 15, який був розроблений у лабораторії Cyber Range Австралійського центру кібербезпеки (ACCS) та містить інформацію про нормальне функціонування мережі та під час синтетичних вторгнень [15].

UNSW-NB15 представляє дев'ять основних груп атак за допомогою інструменту IXIA: PerfectStorm Analysis, Backdoors, DoS, Exploits, Fuzzers, Generic, Reconnaissance, Shellcode and Worms. Існує 45 ознак, розроблених за допомогою інструментів Argus, Bro-IDS та дванадцяти алгоритмів, які охоплюють характеристики мережевих пакетів.

3.2 Попередня обробка даних

Попередня обробка даних у машинному навчанні – це процес підготовки даних для використання в моделі машинного навчання. Цей процес включає в себе такі завдання: очищення даних, вилучення ознак, масштабування даних, балансування даних.

Попередня обробка даних може значно покращити продуктивність моделі машинного навчання, зробивши її більш точною та надійною.

3.2.1 Метод головних компонент

Наявність ознак, які корелюють, негативно впливає на якість моделі. Вони роблять модель менш стійкою і більш чутливою до шуму в даних.

Для боротьби з проблемою кореляції ознак запропоновано використання методу головних компонент (Principal Component Analysis, PCA)[16]

РСА виконує проєкцію даних у новий набір змінних, які називаються головними компонентами, зберігаючи, при цьому, найбільшу кількість інформації, зменшує розмірність даних та прибирає кореляцію між ознаками.

3.2.2 Розробка процедури зменшення кореляції набору даних

У даній роботі запропоновано спеціальну процедуру зменшення кореляції вихідних даних, що дозволило підвищити оперативність процесу ідентифікації.

Процедура зменшення кореляції вихідних даних виконується на етапі попередньої обробки даних та має наступний алгоритм:

Крок 1. Всі категоріальні ознаки перетворюємо на числові значення методом `factorize()`. Заповнюємо пропуски даних.

Крок 2. Аналізуємо ознаки та видаляємо з набору даних неінформативні ознаки.

Крок 3. Будуємо кореляційну матрицю.

Крок 4. Якщо є ознаки що корелюються між собою більш як на 90 відсотків, то обробляємо їх методом головних компонент (РСА). Для цього формуємо датафрейми з двох ознак, що максимально корелюють між собою та застосовуємо метод головних компонент. Кожний набір перетворюємо на нову ознаку. Після формування нових ознак видаляємо старі та додаємо нові ознаки до основного набору даних.

Крок 5. Будуємо модель та оцінюємо її якість. Якщо якість моделі суттєво не змінилася та вище заданого порогового значення, то повертаємося до кроку 4, інакше до кроку 6.

Крок 6. Якщо точність моделі суттєво знизилась, то виконуємо аналіз ознак які були оброблені методом головних компонент на кроці 4 і приймаємо рішення щодо їх відновлення. Завершуємо процес.

Відповідно до вищенаведеного алгоритму виконано попередню обробку даних набору UNSW-NB 15. Попередній аналіз даних показав наявність в даних ознак, що корелюють (Рис. 3.1).

Дані які мають кореляцію вище 90% були спроектовані у новий набір даних за допомогою методу головних компонент (PCA).

Таким чином, за результатами експертного аналізу та використання вищенаведеного алгоритму 42 ознаки датасету трансформовано в 31 ознаку.

```
corr = pre_drop_df.corr()
corr.style.background_gradient(cmap='coolwarm')
```

	dur	proto	service	state	spkts	dpkts	sbytes	dbytes	rate	sttl	dttl	sload	dload	sloss
dur	1.000000	-0.053138	-0.104148	0.079980	0.280239	0.217507	0.225432	0.172492	-0.118032	-0.000990	0.090048	-0.076344	-0.047033	0.240113
proto	-0.053138	1.000000	-0.239996	-0.228909	-0.033177	-0.040910	-0.012132	-0.023490	0.208858	0.199680	-0.221595	0.139353	-0.071932	-0.019758
service	-0.104148	-0.239996	1.000000	-0.135552	-0.026797	-0.046527	0.004752	-0.033948	0.286647	0.112926	-0.378092	0.002966	-0.143624	-0.004532
state	0.079980	-0.228909	-0.135552	1.000000	0.050412	0.045430	0.033448	0.025496	-0.394435	-0.537155	0.295439	-0.270630	0.076395	0.041305
spkts	0.280239	-0.033177	-0.026797	0.050412	1.000000	0.369554	0.965750	0.198324	-0.068249	-0.092536	0.054601	-0.044194	0.074440	0.973644
dpkts	0.217507	-0.040910	-0.046527	0.045430	0.369554	1.000000	0.175834	0.976419	-0.083173	-0.163830	0.036483	-0.054145	0.133835	0.189060
sbytes	0.225432	-0.012132	0.004752	0.033448	0.965750	0.175834	1.000000	0.010036	-0.025102	-0.017866	0.049891	-0.015228	-0.006428	0.995027
dbytes	0.172492	-0.023490	-0.033948	0.025496	0.198324	0.976419	0.010036	1.000000	-0.047978	-0.114537	0.012537	-0.031266	0.100923	0.014561
rate	-0.118032	0.208858	0.286647	-0.394435	-0.068249	-0.083173	-0.025102	-0.047978	1.000000	0.388155	-0.453913	0.550104	-0.138441	-0.040139
sttl	-0.000990	0.199680	0.112926	-0.537155	-0.092536	-0.163830	-0.017866	-0.114537	0.388155	1.000000	-0.033338	0.252901	-0.386224	-0.038088
dttl	0.090048	-0.221595	-0.378092	0.295439	0.054601	0.036483	0.049891	0.012537	-0.453913	-0.033338	1.000000	-0.293939	-0.139491	0.061249
sload	-0.076344	0.139353	0.002966	-0.270630	-0.044194	-0.054145	-0.015228	-0.031266	0.550104	0.252901	-0.293939	1.000000	0.092772	-0.025938
dload	-0.047033	-0.071932	-0.143624	0.076395	0.074440	0.133835	-0.006428	0.100923	-0.138441	-0.386224	-0.139491	-0.092772	1.000000	0.009210
sloss	0.240113	-0.019758	-0.004532	0.041305	0.973644	0.189060	0.995027	0.014561	-0.040139	-0.038088	0.061249	-0.025938	0.009210	1.000000
dloss	0.171182	-0.030430	-0.037502	0.029645	0.198683	0.981506	0.007091	0.997109	-0.062073	-0.137737	0.021966	-0.040456	0.117303	0.014661
sinpkt	0.079840	-0.036653	-0.086310	-0.061994	-0.014501	-0.017141	-0.005399	-0.010201	-0.065681	-0.179270	-0.075620	-0.041718	-0.032094	-0.008124
dinpkt	0.150801	-0.025898	-0.059731	0.076754	-0.003309	-0.007181	-0.001432	-0.007266	-0.052206	-0.006154	0.090734	-0.033787	-0.024481	-0.000470
sjit	0.146598	-0.030106	-0.067714	0.036344	-0.002407	-0.003862	-0.002675	-0.005182	-0.061961	0.030062	0.140634	-0.040018	-0.028759	-0.000422
djit	0.165419	-0.039621	-0.059706	0.032881	0.010481	0.034276	-0.003050	0.029201	-0.081591	-0.084072	0.113799	-0.052820	-0.031342	0.002057

Рисунок 3.1 – Кореляційна матриця.

Надалі виконано нормалізацію даних. Крім того, оскільки в цьому наборі класи розподілені приблизно порівну (з 82332 загальної кількості 45332 це атаки), то балансування класів не виконувалась.

Наша мета навчити модель відрізняти нормальну поведінку мережі від атаки. Цільова ознака – label, яка містить два класи: 0 – нормальна поведінка мережі; 1 – будь-яка атака.

Моделювання будемо проводити на наборі даних в якому кількість ознак зменшена методом головних компонент (PCA) та наборі даних з повним набором ознак (raw).

3.3 Класичні моделі машинного навчання

Спершу для класифікації мережевих даних з набору UNSW-NB15 використано Support vector machine та K-nearest neighbor.

Отримані результати наведено в табл. 3.2.

Таблиця 3.2 – Результати моделювання Support vector machine та K-nearest neighbor.

Показники якості	З використанням методу PCA		Без попередньої обробки даних	
	SVM	KNN	SVM	KNN
accuracy	0.910	0.933	0.908	0.931
precision	0.91	0.93	0.91	0.93
Recall	0.91	0.93	0.91	0.93
F1-score	0.91	0.93	0.91	0.93
Час навчання, с	107	0.06	128	0.02
Час розпізнавання, с	21	10	18	12

Завдяки методу головних компонент для методу SVM маємо вигреш в часі навчання але програш в часі розпізнавання. Для методу KNN навпаки маємо збільшення часу навчання але зменшення часу розпізнавання.

3.4 Моделювання ViT та ViTSD моделей

3.4.1 Підготовка даних

ViT та ViTSD працюють з зображеннями, а точніше з тривимірними масивами. У нас, наприклад, для датасету попередньо обробленого за допомогою методу головних компонент (PCA), буде масив розмірністю (31, 31, 3), де перші два значення це висота та ширина кадру в пікселях а останнє значення це три каналу кольору (RGB).

Отже, після завантаження набору даних нормалізуємо значення від 0 до 255 за допомогою `MinMaxScaler()`, адже кожен канал у системі RGB має значення у діапазоні від 0 до 255. Перетворюємо наш датасет у масив масивів по 31 елементу. Відкидаємо останні кілька масивів для того щоб можна було розділити датасет на 31. Ділимо масив масивів ще на підмасиви по 31 масиву в кожному. Отримуємо структуру (2744, 31, 31).

Але нам потрібна структура (2744, 31, 31, 3), тобто 2744 «зображення» 31x31x3. Для цього ми продублюємо наш масив тричі, об'єднаємо та перетворимо на зображення, а потім із зображення у масив з потрібною розмірністю.

Ми перетворили дані у потрібний формат, але міток класів тепер забагато. Тому ми так само об'єднаємо їх у масиви по 31 елементу, а потім сформуємо новий масив за таким правилом: якщо в масиві є одиниця в новий масив додаємо одиницю, тобто маркуємо як атаку, якщо немає додаємо нуль.

Таким чином, в рамках роботи запропоновано процедуру перетворення табличних вихідних даних у спеціальний формат зображень, необхідний для роботи моделей глибокого навчання Vision Transformer (ViT) та Vision Transformer for Small-size Datasets (ViTSD).

3.4.2 Результати моделювання

Після навчання та тестування отримаємо результати, наведені в Табл. 3.3.

Таблиця 3.3 – Результати моделювання ViT та ViTSD.

Показники якості	З використанням методу PCA		Без попередньої обробки даних	
	ViT-	ViTSD	ViT	ViTSD
аccuracy	0.973	0.987	0.761	0.952
Час навчання, с	1056	1176	816	817
Час розпізнавання, с	20	14	20	10

Як видно із таблиці, застосування методу головних компонент для ViT дозволило значно покращити точність розпізнавання при цьому час розпізнавання не змінився.

Для методу ViTSD точність розпізнавання теж зросла до 3.5 відсотків, хоча й збільшився час розпізнавання.

Збільшення часу навчання при використанні методу головних компонент пов'язано з особливістю представлення даних для цих моделей. Для не оброблених даних маємо 82332 рядки та 42 ознаки, отже отримуємо приблизно 1960 зображень. Для даних, оброблених методом головних компонент, маємо 82332 рядки та 31 ознаку тож отримуємо приблизно 2655 зображень. Тобто, збільшення обсягу вихідних даних призводить до збільшення часу навчання.

3.5 Порівняльний аналіз результатів роботи моделей виявлення вторгнень на основі машинного та глибокого навчання

Результати роботи моделей машинного та глибокого навчання наведено в табл. 3.4.

Таблиця 3.4 – Результати роботи моделей машинного та глибокого навчання.

Показники якості		Accuracy	Час навчання	Час розпізнавання
PCA	SVM	0.910	107	21
	KNN	0.933	0.06	10
	ViT	0.973	1056	20
	ViTSD	0.987	1176	14
raw	SVM	0.908	128	18
	KNN	0.931	0.02	12
	ViT	0.761	816	20

	ViTSD	0.952	817	10
--	-------	-------	-----	----

Як бачимо з таблиці найкращу точність розпізнавання дає модель глибокого навчання Vision Transformer For Small-Size Datasets з попередньою обробкою запропонованим методом зменшення простору ознак.

3.6 Висновки за розділом

В цьому розділі розроблені моделі виявлення вторгнень в комп'ютерні мережі на основі технології машинного та глибокого навчання.

З метою підвищення якості моделі, запропоновано процедуру зменшення кореляції вихідних даних за рахунок рекурсивного використання методу головних компонент (PCA), що дозволило зменшити об'єм вихідних даних та час навчання моделі.

Розроблено процедуру перетворення табличних вихідних даних у спеціальний формат зображень, необхідний для роботи моделей глибокого навчання Vision Transformer (ViT) та Vision Transformer for Small-size Datasets (ViTSD).

Запропоновано метод виявлення вторгнень в комп'ютерні мережі, який відрізняється від відомих використанням алгоритму глибокого навчання Vision Transformer for Small-size Datasets (ViTSD) та спеціальної процедури зменшення кореляції вихідних даних.

У якості вихідних даних використано набір UNSW-NB 15, який був розроблений у лабораторії Cyber Range Австралійського центру кібербезпеки (ACCS) та містить інформацію про нормальне функціонування мережі та під час синтетичних вторгнень.

Для дослідження ефективності моделі у середовищі GOOGLE Colab Python розроблено їх програмні моделі. Якість моделі оцінено за допомогою точності (Accuracy), часу навчання моделі та часу розпізнавання вторгнень.

Використання розробленого методу на основі алгоритму ViTSD, та методу головних компонент для зменшення простору ознак, надало можливість підвищити точності розпізнавання атак на комп'ютерну мережу до 7,9% при використанні моделі на основі алгоритму SVM, до 5,6% при використанні моделі на основі алгоритму KNN та до **22,6%**, при використанні моделі на основі алгоритму ViT.

ВИСНОВКИ

У даній роботі проведено аналіз існуючих систем виявлення вторгнень в комп'ютерні мережі, методів попередньої обробки даних, машинного та глибокого навчання, які використовуються для задач класифікації.

У якості вихідних даних використано набір UNSW-NB 15. Для дослідження ефективності моделей у середовищі GOOGLE Colab Python розроблено їх програмні моделі. Якість моделі оцінено за допомогою точності (Accuracy), часу навчання моделі та часу розпізнавання вторгнень.

За результатами роботи:

1. Запропоновано процедуру зменшення кореляції вихідних даних за рахунок рекурсивного використання методу головних компонент (PCA), що дозволило зменшити об'єм вихідних даних та час навчання моделі.

2. Запропоновано процедуру перетворення табличних вихідних даних у спеціальний формат зображень, необхідний для роботи моделей глибокого навчання Vision Transformer (ViT) та Vision Transformer for Small-size Datasets (ViTSD).

3. Запропоновано метод виявлення вторгнень в комп'ютерні мережі, який відрізняється від відомих використанням алгоритму глибокого навчання Vision Transformer for Small-size Datasets (ViTSD) та спеціальної процедури зменшення кореляції вихідних даних, що дозволило підвищити точність ідентифікації.

Використання розробленого методу на основі алгоритму ViTSD, та методу головних компонент для зменшення простору ознак, надало можливість підвищити точності розпізнавання атак на комп'ютерну мережу до 7,9% при використанні моделі на основі алгоритму SVM, до 5,6% при використанні моделі на основі алгоритму KNN та до **22,6%**, при використанні моделі на основі алгоритму ViT.

СПИСОК ДЖЕРЕЛ ІНФОРМАЦІ

- [1] Ingre, Bhupendra & Yadav, Anamika & Soni, Atul. (2017). Decision Tree Based Intrusion Detection System for NSL-KDD Dataset. 10.1007/978-3-319-63645-0_23.https://www.researchgate.net/publication/315797238_Decision_Tree_Based_Intrusion_Detection_System_for_NSL-KDD_Dataset
- [2] Zhang, Jiong & Zulkernine, Mohammad & Haque, A. (2008). Random-Forests-Based Network Intrusion Detection Systems. Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on. 38.649-659.10.1109/TSMCC.2008.923876.https://www.researchgate.net/publication/3421958_Random-Forests-Based_Network_Intrusion_Detection_Systems
- [3] Liao, Yihua & Vemuri, Rao. (2002). Use of K-Nearest Neighbor classifier for intrusion detection. Computers & Security. 21.439-448.10.1016/S0167-4048(02)00514-X. https://www.researchgate.net/publication/220614594_Use_of_K-Nearest_Neighbor_classifier_for_intrusion_detection
- [4] Wei Wang, «Malware Traffic Classification Using Convolutional Neural Network for Representation Learning». DOI: 10.1109/ICOIN.2017.7899588, January 2017.
- [5] Ashfaq Khan, Muhammad. (2021). HCRNNIDS: Hybrid Convolutional Recurrent Neural Network-Based Network Intrusion Detection System. Processes. 9. 834.10.3390/pr9050834.https://www.researchgate.net/publication/351476948_HCRNNIDS_Hybrid_Convolutional_Recurrent_Neural_Network-Based_Network_Intrusion_Detection_System
- [6] Laghrissi, F., Douzi, S., Douzi, K. et al. Intrusion detection systems using long short-term memory (LSTM). J Big Data 8, 65 (2021). <https://doi.org/10.1186/s40537-021-00448-4>
- [7] Bahdanau, Dzmitry & Cho, Kyunghyun & Bengio, Y. (2014). Neural Machine Translation by Jointly Learning to Align and Translate. ArXiv. 1409.

https://www.researchgate.net/publication/265252627_Neural_Machine_Translation_by_Jointly_Learning_to_Align_and_Translate

[8] Kim, Yoon & Denton, Carl & Hoang, Luong & Rush, Alexander. (2017). Structured Attention Networks. arXiv:1702.00887v3 [cs.CL] 16 Feb 2017. https://www.researchgate.net/publication/313365622_Structured_Attention_Networks

[9] Vaswani, Ashish & Shazeer, Noam & Parmar, Niki & Uszkoreit, Jakob & Jones, Llion & Gomez, Aidan & Kaiser, Lukasz & Polosukhin, Illia, «Attention is all you need», 2017. NIPS'17: Proceedings of the 31st International Conference on Neural Information Processing Systems. <https://dl.acm.org/doi/10.5555/3295222.3295349>

[10] Cho, Kyunghyun & Merriënboer, Bart & Gulcehre, Caglar & Bougares, Fethi & Schwenk, Holger & Bengio, Y. (2014). Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. 10.3115/v1/D14-1179. https://www.researchgate.net/publication/262877889_Learning_Phrase_Representations_using_RNN_Encoder-Decoder_for_Statistical_Machine_Translation

[11] T. M. Khan, M. Alhussein, K. Aurangzeb, M. Arsalan, S. S. Naqvi and S. J. Nawaz, «Residual Connection-Based Encoder Decoder Network (RCED-Net) for Retinal Vessel Segmentation» in IEEE Access, vol. 8, pp. 131257-131272, 2020. DOI: 10.1109/ACCESS.2020.3008899.

[12] Ba, Jimmy, Jamie Ryan Kiros and Geoffrey E. Hinton. «Layer Normalization.» ArXiv abs/1607.06450 (2016): n. pag. <https://arxiv.org/abs/1607.06450>

[13] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, Neil Houlsby. «An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale». arXiv:2010.11929v2 [cs.CV] 3 Jun 2021. <https://arxiv.org/pdf/2010.11929.pdf>

[14] Lee, Seung & Lee, Seunghyun & Song, Byung. «Vision Transformer for Small-Size Datasets». arXiv:2112.13492v1 [cs.CV] 27 Dec 2021. <https://arxiv.org/pdf/2112.13492v1.pdf>

[15] Moustafa, Nour & Slay, Jill. (2015). UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set). 10.1109/MilCIS.2015.7348942. https://www.researchgate.net/publication/287330529_UNSW-NB15_a_comprehensive_data_set_for_network_intrusion_detection_systems_UNSW-NB15_network_data_set

[16] Ibrahim, S.; Nazir, S.; Velastin, S.A. Feature Selection Using Correlation Analysis and Principal Component Analysis for Accurate Breast Cancer Diagnosis. J. Imaging, 2021, Vol.7, pp. 225-241. <https://doi.org/10.3390/jimaging7110225>

[21] Yang Xin, «Machine Learning and Deep Learning Methods for Cybersecurity», IEEE Access PP(99):1-1. DOI: 10.1109/ACCESS.2018.2836950, May 2018.

[22] Jun Lin, «Intrusion Detection of Imbalanced Network Traffic Based on Machine Learning and Deep Learning», IEEE Access VOLUME 9, 2021. DOI: 10.1109/ACCESS.2020.3048198.

[23] Zihao Wang, Kar-Wai Fok, Vrizzlynn L. L. Thing, «Machine Learning for Encrypted Malicious Traffic Detection: Approaches, Datasets and Comparative Study», arXiv:2203.09332v1 [cs.CR] 17 Mar 2022. <https://arxiv.org/pdf/2203.09332.pdf>

[24] A. Ugendhar, «A Novel Intelligent-Based Intrusion Detection System Approach Using Deep Multilayer Classification», Hindawi Mathematical Problems in Engineering Volume 2022, Article ID 8030510, 10 pages. <https://doi.org/10.1155/2022/8030510>

[25] Ricardo Flores Moyano, «Standard Latent Space Dimension for Network Intrusion Detection Systems Datasets», IEEE Access VOLUME 11, 2023, DOI: 10.1109/ACCESS.2023.3283567.

[26] Md. Alamgir Hossain, Md. Saiful Islam, «Ensuring network security with a robust intrusion detection system using ensemble-based machine learning», Array 19 (2023). <https://www.sciencedirect.com/science/article/pii/S2590005623000310>

[27] Eufemia Lella, «Improving the Robustness of DNNs-based Network Intrusion Detection Systems through Adversarial Training», sisinflab.poliba.it 2023. https://sisinflab.poliba.it/publications/2023/LMPLAN23/SPLITECH_2023_Improving_Robustness_DNNS_%5BCamera_Ready%5D.pdf

Додаток А. Фрагменти програми ідентифікації стану комп'ютерної мережі з використанням алгоритмів глибокого навчання ViT та ViTSD на даних з попередньою обробкою

```

!pip install -qq -U tensorflow-addons

import math
import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
import tensorflow as tf
from tensorflow import keras
import tensorflow_addons as tfa
import matplotlib.pyplot as plt
from keras import layers
from PIL import Image
import time

# Setting seed for reproducibility
SEED = 42
keras.utils.set_random_seed(SEED)

pd.options.display.expand_frame_repr = False
pd.set_option('display.max_columns', 100)
pd.set_option('display.max_rows', 100)

NUM_CLASSES = 2
INPUT_SHAPE = (31, 31, 3)

X_bal=pd.read_csv('/content/drive/MyDrive/NTU_KhPI/PCA vs raw datasets/UNSW_NB15_PCA_3.csv')

print(X_bal)

Y_balanced = X_bal['label']
X_balanced = X_bal.drop(columns=['label']).copy()

# Ініціалізуємо MinMaxScaler
scaler = MinMaxScaler(feature_range=(0, 255))

# Використовуємо fit_transform для масштабування кожного стовпця
scaled = scaler.fit_transform(X_balanced)

# Конвертуємо масштабовані дані назад у DataFrame
scaled_df = pd.DataFrame(scaled, columns=X_balanced.columns)

arrays = scaled_df.iloc[:,].to_numpy()
arrays = arrays[:-27]

len(arrays)

frames = []

```

```

block_size = 31 # Розмір кожного блоку

for i in range(0, len(arrays), block_size):
    block = arrays[i:i+block_size]
    # Об'єднуємо всі списки всередині блоку
    merged_block = []

    for sublist in block:
        for item in sublist:
            merged_block.append(item)

    frames.append(np.array(merged_block).reshape(31, 31))

red = []
green = []
blue = []
for i in range(0, len(frames)):
    red.append(Image.fromarray(frames[i]).convert("L"))
    green.append(Image.fromarray(frames[i]).convert("L"))
    blue.append(Image.fromarray(frames[i]).convert("L"))

images = []
for i in range(0, len(red)):
    image = Image.merge("RGB", (red[i], green[i], blue[i]))
    images.append(image)

img_arrs = []
for i in range(0, len(images)):
    img_arrs.append(np.asarray(images[i]))

img_arrs = np.array(img_arrs)

Y_balanced = Y_balanced[:-27]
labels = []
for i in range(0, len(Y_balanced), block_size):
    block = Y_balanced[i:i+block_size]
    labels.append(np.array(block))

Y_labels = []
units = 0
for inner_list in labels:
    if 1 in inner_list:
        Y_labels.append(1)
        units+=1
    else:
        Y_labels.append(0)
print(units)
print(len(Y_labels))
print(len(Y_labels) - units)

X_train, X_test, y_train, y_test = train_test_split(img_arrs, Y_labels, test_size=0.3, random_state=42)

```

```

y_train = np.array(y_train)
y_test = np.array(y_test)

y_train = np.expand_dims(y_train, axis=1)
y_test = np.expand_dims(y_test, axis=1)

print(f"x_train shape: {X_train.shape} - y_train shape: {y_train.shape}")
print(f"x_test shape: {X_test.shape} - y_test shape: {y_test.shape}")

# DATA
BUFFER_SIZE = 512
BATCH_SIZE = 128

# AUGMENTATION
IMAGE_SIZE = 72
PATCH_SIZE = 6
NUM_PATCHES = (IMAGE_SIZE // PATCH_SIZE) ** 2

# OPTIMIZER
LEARNING_RATE = 0.001
WEIGHT_DECAY = 0.0001

# TRAINING
EPOCHS = 10

# ARCHITECTURE
LAYER_NORM_EPS = 1e-6
TRANSFORMER_LAYERS = 8
PROJECTION_DIM = 64
NUM_HEADS = 4
TRANSFORMER_UNITS = [
    PROJECTION_DIM * 2,
    PROJECTION_DIM,
]
MLP_HEAD_UNITS = [2048, 1024]

data_augmentation = keras.Sequential(
    [
        layers.Normalization(),
        layers.Resizing(IMAGE_SIZE, IMAGE_SIZE),
        layers.RandomFlip("horizontal"),
        layers.RandomRotation(factor=0.02),
        layers.RandomZoom(height_factor=0.2, width_factor=0.2),
    ],
    name="data_augmentation",
)
# Compute the mean and the variance of the training data for normalization.
data_augmentation.layers[0].adapt(X_train)

class ShiftedPatchTokenization(layers.Layer):
    def __init__(
        self,
        image_size=IMAGE_SIZE,

```

```

patch_size=PATCH_SIZE,
num_patches=NUM_PATCHES,
projection_dim=PROJECTION_DIM,
vanilla=False,
**kwargs,
):
    super().__init__(**kwargs)
    self.vanilla = vanilla # Flag to switch to vanilla patch extractor
    self.image_size = image_size
    self.patch_size = patch_size
    self.half_patch = patch_size // 2
    self.flatten_patches = layers.Reshape((num_patches, -1))
    self.projection = layers.Dense(units=projection_dim)
    self.layer_norm = layers.LayerNormalization(epsilon=LAYER_NORM_EPS)

def crop_shift_pad(self, images, mode):
    # Build the diagonally shifted images
    if mode == "left-up":
        crop_height = self.half_patch
        crop_width = self.half_patch
        shift_height = 0
        shift_width = 0
    elif mode == "left-down":
        crop_height = 0
        crop_width = self.half_patch
        shift_height = self.half_patch
        shift_width = 0
    elif mode == "right-up":
        crop_height = self.half_patch
        crop_width = 0
        shift_height = 0
        shift_width = self.half_patch
    else:
        crop_height = 0
        crop_width = 0
        shift_height = self.half_patch
        shift_width = self.half_patch

    # Crop the shifted images and pad them
    crop = tf.image.crop_to_bounding_box(
        images,
        offset_height=crop_height,
        offset_width=crop_width,
        target_height=self.image_size - self.half_patch,
        target_width=self.image_size - self.half_patch,
    )
    shift_pad = tf.image.pad_to_bounding_box(
        crop,
        offset_height=shift_height,
        offset_width=shift_width,
        target_height=self.image_size,
        target_width=self.image_size,
    )

```

```

return shift_pad

def call(self, images):
    if not self.vanilla:
        # Concat the shifted images with the original image
        images = tf.concat(
            [
                images,
                self.crop_shift_pad(images, mode="left-up"),
                self.crop_shift_pad(images, mode="left-down"),
                self.crop_shift_pad(images, mode="right-up"),
                self.crop_shift_pad(images, mode="right-down"),
            ],
            axis=-1,
        )
        # Patchify the images and flatten it
        patches = tf.image.extract_patches(
            images=images,
            sizes=[1, self.patch_size, self.patch_size, 1],
            strides=[1, self.patch_size, self.patch_size, 1],
            rates=[1, 1, 1, 1],
            padding="VALID",
        )
        flat_patches = self.flatten_patches(patches)
        if not self.vanilla:
            # Layer normalize the flat patches and linearly project it
            tokens = self.layer_norm(flat_patches)
            tokens = self.projection(tokens)
        else:
            # Linearly project the flat patches
            tokens = self.projection(flat_patches)
        return (tokens, patches)

# Get a random image from the training dataset
# and resize the image
image = X_train[np.random.choice(range(X_train.shape[0]))]
resized_image = tf.image.resize(
    tf.convert_to_tensor([image]), size=(IMAGE_SIZE, IMAGE_SIZE)
)

# Vanilla patch maker: This takes an image and divides into
# patches as in the original ViT paper
(token, patch) = ShiftedPatchTokenization(vanilla=True)(resized_image / 255.0)
(token, patch) = (token[0], patch[0])
n = patch.shape[0]
count = 1
plt.figure(figsize=(4, 4))
for row in range(n):
    for col in range(n):
        plt.subplot(n, n, count)
        count = count + 1
        image = tf.reshape(patch[row][col], (PATCH_SIZE, PATCH_SIZE, 3))

```

```

plt.imshow(image)
plt.axis("off")
plt.show()

# Shifted Patch Tokenization: This layer takes the image, shifts it
# diagonally and then extracts patches from the concatenated images
(token, patch) = ShiftedPatchTokenization(vanilla=False)(resized_image / 255.0)
(token, patch) = (token[0], patch[0])
n = patch.shape[0]
shifted_images = ["ORIGINAL", "LEFT-UP", "LEFT-DOWN", "RIGHT-UP", "RIGHT-DOWN"]
for index, name in enumerate(shifted_images):
    print(name)
    count = 1
    plt.figure(figsize=(4, 4))
    for row in range(n):
        for col in range(n):
            plt.subplot(n, n, count)
            count = count + 1
            image = tf.reshape(patch[row][col], (PATCH_SIZE, PATCH_SIZE, 5 * 3))
            plt.imshow(image[..., 3 * index : 3 * index + 3])
            plt.axis("off")
    plt.show()

```

```

class PatchEncoder(layers.Layer):
    def __init__(
        self, num_patches=NUM_PATCHES, projection_dim=PROJECTION_DIM, **kwargs
    ):
        super().__init__(**kwargs)
        self.num_patches = num_patches
        self.position_embedding = layers.Embedding(
            input_dim=num_patches, output_dim=projection_dim
        )
        self.positions = tf.range(start=0, limit=self.num_patches, delta=1)

    def call(self, encoded_patches):
        encoded_positions = self.position_embedding(self.positions)
        encoded_patches = encoded_patches + encoded_positions
        return encoded_patches

```

```

class MultiHeadAttentionLSA(tf.keras.layers.MultiHeadAttention):
    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        # The trainable temperature term. The initial value is
        # the square root of the key dimension.
        self.tau = tf.Variable(math.sqrt(float(self._key_dim)), trainable=True)

    def _compute_attention(self, query, key, value, attention_mask=None, training=None):
        query = tf.multiply(query, 1.0 / self.tau)
        attention_scores = tf.einsum(self._dot_product_equation, key, query)
        attention_scores = self._masked_softmax(attention_scores, attention_mask)
        attention_scores_dropout = self._dropout_layer(

```

```

        attention_scores, training=training
    )
    attention_output = tf.einsum(
        self._combine_equation, attention_scores_dropout, value
    )
    return attention_output, attention_scores

```

```

def mlp(x, hidden_units, dropout_rate):
    for units in hidden_units:
        x = layers.Dense(units, activation=tf.nn.gelu)(x)
        x = layers.Dropout(dropout_rate)(x)
    return x

```

```

# Build the diagonal attention mask
diag_attn_mask = 1 - tf.eye(NUM_PATCHES)
diag_attn_mask = tf.cast([diag_attn_mask], dtype=tf.int8)

```

```

def create_vit_classifier(vanilla=False):
    inputs = layers.Input(shape=INPUT_SHAPE)
    # Augment data.
    augmented = data_augmentation(inputs)
    # Create patches.
    (tokens, _) = ShiftedPatchTokenization(vanilla=vanilla)(augmented)
    # Encode patches.
    encoded_patches = PatchEncoder()(tokens)

    # Create multiple layers of the Transformer block.
    for _ in range(TRANSFORMER_LAYERS):
        # Layer normalization 1.
        x1 = layers.LayerNormalization(epsilon=1e-6)(encoded_patches)
        # Create a multi-head attention layer.
        if not vanilla:
            attention_output = MultiHeadAttentionLSA(
                num_heads=NUM_HEADS, key_dim=PROJECTION_DIM, dropout=0.1
            )(x1, x1, attention_mask=diag_attn_mask)
        else:
            attention_output = layers.MultiHeadAttention(
                num_heads=NUM_HEADS, key_dim=PROJECTION_DIM, dropout=0.1
            )(x1, x1)
        # Skip connection 1.
        x2 = layers.Add()([attention_output, encoded_patches])
        # Layer normalization 2.
        x3 = layers.LayerNormalization(epsilon=1e-6)(x2)
        # MLP.
        x3 = mlp(x3, hidden_units=TRANSFORMER_UNITS, dropout_rate=0.1)
        # Skip connection 2.
        encoded_patches = layers.Add()([x3, x2])

    # Create a [batch_size, projection_dim] tensor.
    representation = layers.LayerNormalization(epsilon=1e-6)(encoded_patches)
    representation = layers.Flatten()(representation)

```

```

representation = layers.Dropout(0.5)(representation)
# Add MLP.
features = mlp(representation, hidden_units=MLP_HEAD_UNITS, dropout_rate=0.5)
# Classify outputs.
logits = layers.Dense(NUM_CLASSES)(features)
# Create the Keras model.
model = keras.Model(inputs=inputs, outputs=logits)
return model

```

Some code is taken from:

<https://www.kaggle.com/ashusma/training-rfcx-tensorflow-tpu-effnet-b2>.

```

class WarmUpCosine(keras.optimizers.schedules.LearningRateSchedule):
    def __init__(
        self, learning_rate_base, total_steps, warmup_learning_rate, warmup_steps
    ):
        super().__init__()

        self.learning_rate_base = learning_rate_base
        self.total_steps = total_steps
        self.warmup_learning_rate = warmup_learning_rate
        self.warmup_steps = warmup_steps
        self.pi = tf.constant(np.pi)

    def __call__(self, step):
        if self.total_steps < self.warmup_steps:
            raise ValueError("Total_steps must be larger or equal to warmup_steps.")

        cos_annealed_lr = tf.cos(
            self.pi
            * (tf.cast(step, tf.float32) - self.warmup_steps)
            / float(self.total_steps - self.warmup_steps)
        )
        learning_rate = 0.5 * self.learning_rate_base * (1 + cos_annealed_lr)

        if self.warmup_steps > 0:
            if self.learning_rate_base < self.warmup_learning_rate:
                raise ValueError(
                    "Learning_rate_base must be larger or equal to "
                    "warmup_learning_rate."
                )
            slope = (
                self.learning_rate_base - self.warmup_learning_rate
            ) / self.warmup_steps
            warmup_rate = slope * tf.cast(step, tf.float32) + self.warmup_learning_rate
            learning_rate = tf.where(
                step < self.warmup_steps, warmup_rate, learning_rate
            )
        return tf.where(
            step > self.total_steps, 0.0, learning_rate, name="learning_rate"
        )

```

```

def run_experiment(model):
    total_steps = int((len(X_train) / BATCH_SIZE) * EPOCHS)
    warmup_epoch_percentage = 0.10
    warmup_steps = int(total_steps * warmup_epoch_percentage)
    scheduled_lrs = WarmUpCosine(
        learning_rate_base=LEARNING_RATE,
        total_steps=total_steps,
        warmup_learning_rate=0.0,
        warmup_steps=warmup_steps,
    )

    optimizer = tfa.optimizers.AdamW(
        learning_rate=LEARNING_RATE, weight_decay=WEIGHT_DECAY
    )

    model.compile(
        optimizer=optimizer,
        loss=keras.losses.SparseCategoricalCrossentropy(from_logits=True),
        metrics=[
            keras.metrics.SparseCategoricalAccuracy(name="accuracy"),
        ],
    )

    # Початковий час
    start_time = time.time()

    history = model.fit(
        x=X_train,
        y=y_train,
        batch_size=BATCH_SIZE,
        epochs=EPOCHS,
        validation_split=0.1,
    )

    # Кінцевий час
    end_time = time.time()

    # Обчислити тривалість навчання
    training_duration = end_time - start_time

    print(f"Час навчання: {training_duration} секунд")

    # Початковий час
    start_time = time.time()

    _, accuracy = model.evaluate(X_test, y_test, batch_size=BATCH_SIZE)

    # Кінцевий час
    end_time = time.time()

    # Обчислити тривалість розпізнавання
    prediction_duration = end_time - start_time

```

```
print(f"Час розпізнавання: {prediction_duration} секунд")
```

```
print(f"Test accuracy: {round(accuracy * 100, 2)}%")
```

```
return history
```

```
# Run experiments with the vanilla ViT
```

```
vit = create_vit_classifier(vanilla=True)
```

```
history = run_experiment(vit)
```

```
# Run experiments with the Shifted Patch Tokenization and
```

```
# Locality Self Attention modified ViT
```

```
vit_sl = create_vit_classifier(vanilla=False)
```

```
history = run_experiment(vit_sl)
```