МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

КАФЕДРА КОМП'ЮТЕРНИХ НАУК

КВАЛІФІКАЦІЙНА МАГІСТЕРСЬКА РОБОТА

на тему:

«Інформаційна технологія детектування атак на систему штучного інтелекту»

Завідувач випускаючої кафедри — **Довбиш А.С.**

Керівник роботи — **Москаленко В.В.**

Студента групи ІН.м – 02 — **Ковальчука Б.В.**

СУМИ 2021

Сумський державний університет
(назва вузу)

Факультет *ЕлІТ* Кафедра *Комп'ютерних наук*

Спеціальність *«122 - Комп'ютерні науки»*

Затверджую:
зав.кафедрою _____
_____
"_____" _____ 20___р.

ЗАВДАННЯ
НА ДИПЛОМНИЙ ПРОЕКТ (РОБОТУ) СТУДЕНТОВІ

Ковальчуку Богдану Вікторовичу
(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) *Інформаційна технологія детектування атак на систему штучного інтелекту*

затверджую наказом по інституту від "____" _____ 20 ___ р. № _____

2. Термін здачі студентом закінченого проекту (роботи)_____

3. Вхідні данні до проекту (роботи) _____
_____
_____
_____

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)
1)Аналітичний огляд існуючих рішень та предметної області; 2) Постановка завдання й формування завдань дослідження; 3); Огляд і розробка методів і моделей розпізнавання зображень; 4) Розробка інформаційної технології захисту від змагальних атак; 5) Програмна реалізація; 6) Аналіз результатів.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) _____
_____

6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання _____

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів дипломного проекту (роботи)	Термін виконання проекту (роботи)	Примітка
1.	Огляд технологій, що застосовуються для генерації змагальних атак на систему штучного інтелекту їх виявлення та захисту від них.	24.11.2021 – 28.11.2021	
2.	Постановка задачі та формування завдань дослідження.	29.11.2021 – 30.11.2021	
3.	Опис моделі розпізнавання зображень з захистом від змагальних атак.	01.12.2021 – 04.12.2021	
4.	Розробка інформаційної технології детектування атак на систему штучного інтелекту. Програмна реалізація.	05.11.2021 – 09.11.2021	
5.	Оформлення пояснювальної записки до кваліфікаційної магістерської роботи .	10.12.2021 – 12.12.2021	

Студент – дипломник _____
(підпис)

Керівник проекту _____
(підпис)

РЕФЕРАТ

Записка: 57 стор., 14 рис., 0 таблиць, 1 додаток, 65 літературних джерел.

Об’єкт дослідження — інформаційна технологія .

Мета роботи — розробка та програмна реалізація інформаційної технології захисту від змагальних атак на систему штучного інтелекту.

Результати — проведений аналіз літератури, методів та інструментів, які дозволяють генерувати змагальні атаки на систему штучного інтелекту та захищати її від них. Розроблена технологія покращує точність класифікатора, дозволяє виявляти можливі атаки, реалізований метод захисту. Вході роботи було розроблено алгоритм програми та її реалізація у вигляді Jupyter ноутбуку. Результати дослідів показали досить високу ефективність розробки та її стійкість до змагальних атак типу FGM. Точність класифікатора стандартної нейронної мережі в узгодженні з оцінкою довіри зростає. Програмна реалізація виконана на мові Python у хмарному середовищі Google Colab.

PYTHON, TRUST SCORE, ШТУЧНИЙ ІНТЕЛЕКТ, НЕЙРОННА МЕРЕЖА, ADVERSARIAL ATTACK, ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ, ДЕТЕКТУВАННЯ АТАК, СИСТЕМА ШТУЧНОГО ІНТЕЛЕКТУ

ЗМІСТ

ВСТУП

У сучасному світі обчислювальні технології та комп’ютеризація усіх видів господарства набули надзвичайного поширення. Збільшення обчислювальної потужності в трильйон разів популяризувало використання глибокого навчання (DL, deep learning) для обробки різноманітних завдань машинного навчання, наприклад, таких як класифікація та розпізнавання зображень, обробка природної мови та теорія ігор [1-3]. Наразі дослідницькою спільнотою було виявлено чимало серйозних загроз безпеці існуючих алгоритмів DL: злочинці можуть із легкістю обдурити моделі DL, порушуючи доброякісні зразки, не будучи виявлені людьми.

Набувають поширення змагальні атаки на системи штучного інтелекту з метою впливу на результати їх прогнозу, щоб отримати конкретний необхідний для злочинця прогноз або ж щоб система видала просто невірний прогноз.

Зростаюче поширення штучного інтелекту, ймовірно, буде корелювати зі зростанням кількості ворожих атак. Це нескінченна гонка озброєнь, але, на щастя, сьогодні існують ефективні підходи для пом’якшення найгірших атак.

З огляду на швидкий розвиток методів штучного інтелекту (ШІ) і глибокого навчання (DL), надзвичайно важливо забезпечити безпеку та надійність розгорнутих алгоритмів. Останнім часом широко визнається вразливість алгоритмів DL до протиборчих зразків. Виготовлені зразки можуть призводити до різного роду неправильної поведінки моделей DL, водночас сприймаючись людьми як доброякісні. Успішна реалізація ворожих атак у реальних сценаріях фізичного світу ще більше демонструє їх практичність. Таким чином, методи змагання та захисту привертають все більшу увагу як машинного навчання, так і спільнот безпеки, і останніми роками вони стали гарячою темою дослідження.

1 АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ ТА ПОСТАНОВКА ЗАДАЧІ

Змагальне машинне навчання, метод, який намагається обдурити моделі за допомогою оманливих даних, є зростаючою загрозою в дослідницькому співтоваристві ШІ та машинного навчання. Найпоширенішою причиною є несправність моделі машинного навчання. Змагальна атака може спричинити за собою представлення моделі з неточними чи неправдивими даними під час навчання або введення зловмисно розроблених даних, з метою обману вже навченої моделі. Як зазначається у проміжній доповіді Комісії національної безпеки США зі штучного інтелекту за 2019 рік, дуже невеликий відсоток поточних досліджень ШІ спрямований на захист систем штучного інтелекту від ворожих зусиль [4]. Деякі системи, які вже використовуються у виробництві, можуть бути вразливими для атак. Наприклад, розмістивши на землі кілька спеціальних невеликих наклейок, дослідники показали, що вони можуть змусити самокерований автомобіль виїхати на зустрічну смугу руху [5]. Інші дослідження показали, що внесення непомітних змін у зображення може обдурити систему медичного аналізу [6], щоб вона класифікувала доброякісні медичні зразки як злоякісні, і що шматочки стрічки можуть обдурити систему комп’ютерного зору, щоб неправильно класифікувати дорожній знак зупинки як знак обмеження швидкості [7] .

Атаки на моделі штучного інтелекту часто класифікуються за трьома основними осями. Вплив на класифікатор, порушення безпеки та їх специфічність – можуть бути додатково класифіковані як «білий ящик» або «чорний ящик». Під час атак на білий ящик зловмисник має доступ до параметрів моделі, тоді як у атаках чорного ящика зловмисник не має доступу до цих параметрів.

Атака може вплинути на класифікатор, тобто на модель, порушуючи модель, коли вона робить передбачення, тоді як порушення безпеки передбачає надання шкідливих даних, які класифікуються як легітимні.

Цілеспрямована атака намагається дозволити конкретне вторгнення або порушення, або, як альтернатива, створити загальний хаос.

Атаки ухилення є найпоширенішим типом атаки, коли дані змінюються, щоб уникнути виявлення або класифікуватися як легітимні. Ухилення не передбачає впливу на дані, які використовуються для навчання моделі, але це можна порівняти з тим, як спамери та хакери приховують вміст спам-листів і шкідливих програм. Прикладом ухилення є спам на основі зображень, у якому вміст спаму вбудовується у вкладене зображення, щоб уникнути аналізу моделями захисту від спаму [8].

Іншим прикладом підробки є атаки на системи біометричної перевірки на основі штучного інтелекту. Інший тип атаки отруєння — це «зразкове зараження» даних. Системи машинного навчання часто перенавчаються за допомогою даних, зібраних під час їх роботи, і зловмисник може отруїти ці дані, впроваджуючи шкідливі зразки, які згодом порушують процес перенавчання. Зловмисник може ввести дані під час фази навчання, які помилково позначено як нешкідливі, коли вони насправді шкідливі. Наприклад, великі мовні моделі, такі як GPT-3 OpenAI, можуть розкривати чутливу, конфіденційну інформацію під час подачі певних слів і фраз, показало дослідження [8].

Тим часом крадіжка моделі, яка також називається вилученням моделі, передбачає, що супротивник досліджує систему машинного навчання «чорного ящика», щоб або відновити модель, або витягти дані, на яких вона була навчена. Це може викликати проблеми, коли навчальні дані або сама модель є вразливими та конфіденційними. Наприклад, крадіжка моделі може бути використана для створення власної моделі біржової торгівлі, яку супротивник потім може використовувати для власної фінансової вигоди.

На сьогоднішній день задокументовано чимало прикладів змагальних атак. Один із них показав, що можна надрукувати 3D-іграшку черепахи з текстурою, яка змушує AI для виявлення об’єктів Google класифікувати її як гвинтівку, незалежно від ракурсу, під яким черепаха була сфотографована. Під

час іншої атаки було показано, що зображення собаки, змінене машиною, виглядає як кішка як для комп'ютерів, так і для людей. Так звані «зразкові візерунки» на окулярах або одязі були розроблені, щоб обдурити системи розпізнавання обличь і зчитувачів номерних знаків. Також, дослідники створили суперечливі аудіо входи, щоб замаскувати команди для розумних помічників у доброякісному аудіо [8].

У статті [9], дослідники з Google і Каліфорнійського університету в Берклі продемонстрували, що навіть найкращі криміналістичні класифікатори — системи штучного інтелекту, навчені розрізняти реальний і синтетичний контент сприйнятливі до змагальних атак. Це турбує організації, які намагаються виготовляти детектори підроблених засобів масової інформації, особливо з огляду на стрімке зростання deepfake контенту в інтернеті.

Одним із найвідоміших останніх прикладів є Тей від Microsoft, чат-бот для Twitter, запрограмований на навчання брати участь у розмові через взаємодію з іншими користувачами. Хоча намір Microsoft полягав у тому, щоб Тей брав участь у «звичайній та грайливій розмові», інтернет-тролі було помічено, що в системі недостатньо фільтрів, і почали подавати Теєві нецензурні та образливі твіти. Чим більше подібні користувачі залучалися, тим образливішими ставали твіти Тея, що змусило Microsoft закрити бота всього лише після 16 годин його роботи.

Як зазначає [10] учасник VentureBeat Бен Діксон, останніми роками спостерігається сплеск кількості досліджень змагальних атак. У 2014 році на сервер препринтів Arxiv.org було подано нуль документів про змагальне машинне навчання, тоді як у 2020 році близько 1000 статей про змагальні зразки та атаки. Змагальні атаки та методи захисту також стали родзинкою відомих конференцій, включаючи NeurIPS, ICLR, Black Hat і Usenix.

Зі зростанням інтересу до змагальних атак та методів боротьби з ними, такі стартапи, як Resistant AI [11], виходять на ринок з продуктами, які нібито «зміцнюють» алгоритми для ефективного захисту від супротивників. Крім цих

нових комерційних рішень, нові дослідження є перспективними для підприємств, які хочуть інвестувати в захист від змагальних атак.

Одним із способів перевірити надійність моделей машинного навчання є те, що називають троянською атакою, яка передбачає модифікацію моделі, щоб вона реагувала на тригери введення, які змушують її робити висновок про неправильну відповідь. Намагаючись зробити ці тести більш повторюваними і масштабованими, дослідники з Університету Джона Хопкінса розробили фреймворк, який отримав назву TrojAI , набір інструментів, які генерують ініційовані набори даних і пов'язані моделі з троянами. Вони кажуть, що це дозволить дослідникам зрозуміти вплив різних конфігурацій набору даних на створені моделі «троянів» і допоможе всебічно випробувати нові методи виявлення троянів для підвищення надійності та безпеки моделей.

Команда Джона Хопкінса далеко не єдина, хто вирішує проблему змагальних атак у машинному навчанні. Наприклад, дослідники Google опублікували статтю [12], яка описує фреймворк, який або виявляє атаки, або тисне на зловмисників, щоб вони створювали зображення, які нагадують цільовий клас зображень. Baidu, Microsoft, IBM і Salesforce пропонують наступні набори інструментів — Advbox , Counterfit , Adversarial Robustness Toolbox і Robustness Gym — для створення прикладів змагальності, які можуть обдурити моделі в таких фреймворках, як MxNet, Keras, Facebook PyTorch і Caffe2, Google і BaiPad BaiPad від Google. А Лабораторія комп'ютерних наук і штучного інтелекту Массачусетського технологічного інституту нещодавно випустила інструмент під назвою TextFooler, який генерує протилежний текст для посилення моделей природної мови.

Зовсім недавно, Microsoft, некомерційна організація Mitra Corporation, і 11 інших організацій, в тому числі IBM, Nvidia, Airbus і Bosch випустили [13] Adversarial ML Threat Matrix, орієнтований на індустрію відкритий фреймворк, призначений допомогти аналітикам з безпеки для виявляти, реагувати та усувати загрози в відношенні системи машинного навчання. Microsoft стверджує, що співпрацювала з Mitre над створенням схеми, яка

організовує підходи, які зловмисники використовують для підриву моделей машинного навчання, посилюючи стратегії моніторингу критично важливих систем організації.

Майбутнє може принести нестандартні підходи, у тому числі декілька, натхненних нейронаукою. Наприклад, дослідники з MIT-IBM Watson AI Lab виявили, що пряме відображення особливостей зорової кори ссавців на глибокі нейронні мережі створює системи штучного інтелекту, які є більш стійкими до ворожих атак. Хоча змагальний ШІ, ймовірно, стане нескінченною гонкою озброєнь, подібні рішення вселяють надію, що зловмисники не завжди отримають перевагу і що біологічний інтелект все ще має великий невикористаний потенціал.

1.1 Аналіз існуючих методів генерації змагальних атак на алгоритми штучного інтелекту

Змагальні атаки можуть бути розгорнуті або під час прийняття рішення (атаки ухилення), або під час навчання (атаки отруєння). У кожному випадку алгоритмом навчання (для атак отруєння) або засвоєною моделлю (для атак ухилення) маніпулюють за допомогою якоїсь форми ретельно розробленого входу, відомого як змагальні зразки. Загальна тенденція серед методів атаки, наведених нижче, показує, що надійність моделі машинного навчання значною мірою залежить від здатності зловмисника знайти змагальний зразок, який максимально наближений до вихідного введення. Коротка класифікація змагальних алгоритмів приведена на рис. 1.1. Схематичний приклад роботи змагальної атаки приведено на рис. 1.2.

Атаки ухилення: атаки ухилення намагаються ввести систему машинного навчання в оману під час фази тестування або висновку. Нижче виділені методи змагальної атаки, які підпадають під категорію атак ухилення. Атаки далі поділяються на атаки на основі градієнта та без градієнта.

Атаки на основі градієнта: Szegedy та ін. [14] вивчено як можуть бути створені змагальні зразки проти нейронних мереж для класифікації зображень. Після чого був введений L-BFGS (Обмежений метод Бройдена-Флетчера-Гольдфарба-Шанно), який використовував дорогий лінійний метод пошуку для знаходження оптимальних значень змагальних вибірок. В іншому підході, запропонованому Гудфеллоу та ін. [15], що називається метод швидкого градієнту ознак (FGSM), створюються змагальні зразки знаходження максимального напрямку позитивної зміни у втраті. Це швидший метод, ніж метод L-BFGS оскільки виконується лише одно етапне оновлення градієнта уздовж напрямку градієнта ознаки на кожному рівні. Основне обмеження методу швидкого градієнта і подібних методів атаки полягає в тому, що вони працюють на основі припущення, що змагальні зразки можуть бути подані безпосередньо в модель машинного навчання. Це далеко від практики, оскільки більшість нападників прагнули б щоб отримати доступ до моделей машинного навчання через пристрої, наприклад, датчики [16]. Базовий ітеративний метод (BIM) запропонований у [17] долає це обмеження за допомогою виконання оновлення градієнта за кілька ітерацій.

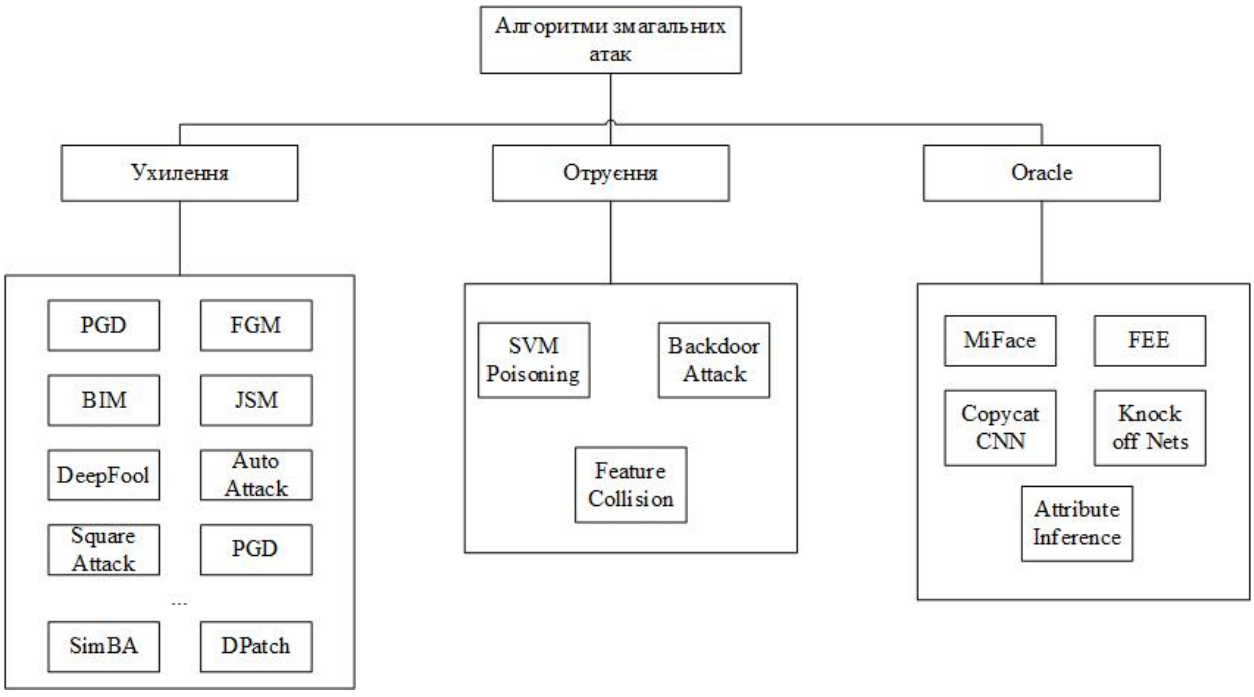


Рисунок 1.1 – Класифікація алгоритмів змагальних атак.

DeepFool був запропонований Moosavi та ін. [18] як метод створення змагальних зразків знайшовши найближчу відстань між оригіналом вхідних даних та межею рішення для змагальних вибірок. Вони змогли визначити це за допомогою пов’язаного класифікатора, найближча відстань, яка б відповідала до мінімального збурення для створення змагальності вибіркою буде відстань до гіперплощини пов'язаного класифікатора. Jang та ін. [19] представив атаку NewtonFool, алгоритм, заснований на градієнтному спуску для пошуку змагальні зразки. Ця атака схожа на Deepfool [18], але більш ефективна у створенні якісних змагальних зразків і зменшує довірчу ймовірність правильного класу. Вони використовують шар softmax і контролюють розмір кроку та те, наскільки малим може бути збурення. Карліні та ін. [20] розробив атаку Карліні та Вагнера, цілеспрямована атака спеціально для уже існуючих методів захисту від змагальних атак.

Рисунок 1.2 – Схематичне зображення змагальної атаки.

Було виявлено, що захисти, такі як, захисна дистиляція були неефективними щодо нападів Карліні і Вагнера. Мадрі та ін. [21] запропонував протилежний проект, проектованого градієнтного спуску (PGD) змагальної атаки, який є більш робустним, ніж FGSM. Ця форма атаки використовує багатоетапний підхід з негативною функцією втрати. Він долає проблему переповненості мережі, і незабаром з'являються змагальні зразки FGSM. Це є більш надійним, ніж FGSM, який використовує мережеву інформацію першого порядку, і добре працює у великих масштабах. У l_∞-кулі PGD виконує ітерацію, щоб досліджувати максимальні втрати. Кроче та ін. [22]

запропонував Auto Attack, атаку, яка долає та усуває слабкі сторони проектованого градієнтного спуску (PGD) [21], які призводять до хибних результатів робастності моделі. Перша атака PGD використовує фіксований крок розмір з крос-ентропією як функцією втрат, що викликає провал як тотожність. У [22] використовують нову схему на основі градієнта без вибору розміру кроку з різною функцією втрат. З цими двома змінами дві версії PGD випускаються з вільними параметрами в кількості ітерацій. Вони також інтегрують нову версію PGD з FAB-атакою і атакою Square, щоб створити атаку без параметрів під назвою AutoAttack. Автори також інтегрували два Auto Attack і було апробовано у великому масштабі на 40 класифікаторах. Сабур та ін. [23] запропонував новий образ змагальної атаки зображень, яка фокусується не тільки на мітці класу, а й на внутрішніх уявленнях. Атака, відома як Feature Adversaries дає можливість обдурити навченого DNN, щоб містифікувати будь-яке вихідне зображення з іншим цільовим зображенням, знайшовши невелике збурення від джерела зображення, які створюють подібне внутрішнє уявлення до цільового зображення і не пов’язане з вихідним зображенням. Однак автори враховують, що такі супротивники не є сторонніми. Universal Perturbation [24] був запропонований Moosavi та ін. як алгоритм для обчислення універсального малого збурення зображення для неправильної класифікації найсучаснішого класифікатора глибоких нейронних мереж. Основною метою цього алгоритму було знаходження вектору збурення, який обманює класифікатор на всіх вибірках точок даних. Це збурення виправлення існує, щоб спричинити зміни зображення мітку поступово побудувати універсальне збурення.

Атаки без градієнта: Атака дерева рішень була запропонована Papernot та ін. [25] цей тип атак чорного ящика використовує можливість передачі змагальних вибірок між різними класифікаторами та всередині них, включаючи глибоку нейронну мережу. Логістична регресія, дерева рішень, машини опорних векторів (SVM), ансамблі та найближчі сусіди. Вони продемонстрували, що атаки на чорну скриньку можливі для алгоритму

машинного навчання, який не використовує глибокі нейронні мережі та змагальні вибірки добре працюють між моделями та серед них, що використовують однакові та різні методи машинного навчання. Чен та ін. [26] запропонував алгоритм змагальної атаки для атаки на DNN, заснований на регуляризації еластичної мережі в ознаках L1 і L2, які називаються атаками еластичної мережі на DNN (EAD). EAD вважає, що сучасні L2 і $L_{infinty}$ автори продемонстрували, що EAD може зламати незахищені та оборонно дистильовані DNN. Вони також покращують переносимість атак і змагання. Shadow Attack була запропонована Ghiasi та ін. [27], який є новим методом атаки на системи, які покладаються на сертифікати та обманюють сертифіковані надійні мережі, щоб призначити неправильну мітку зображенню та створити підроблений безпечний сертифікат надійності для змагального прикладу.

Adversarial Patch, запропонований Brown та ін. [28] представляють універсальні, надійні та цілеспрямовані змагальні патчі для реального світу, які не вимагають жодних знань про те, який образ вони атакують. Ці змагальні зразки можна використовувати для атаки на будь-який класифікатор, і вони працюють з багатьма перетвореннями, які існують, що методи захисту можуть бути нестійкими для такої масової трансформації. Змагальний патч змушує класифікатор перемикати мітки класів на будь-який цільовий клас. Чен та ін. [29] розробити HopSkipJumpAttack на основі атаки, що базується на рішеннях, яка є типом атаки чорного ящика. Цей алгоритм генерує ітераційні цільові та нецільові змагальні вибірки з мінімальною відстанню. Ця атака демонструє вищу ефективність у порівнянні з різними найсучаснішими атаками, що базуються на рішеннях. Ітерація в алгоритмі заснована на напрямі градієнта, розмірі кроку та пошуку меж.

Атаки отруєння: атака отруєння, також відома як причинна атака, використовує прямі або непрямі засоби для зміни даних або моделі. Атаки отруєння відбуваються шляхом введення неправдивих даних, маніпулювання вихідними даними або пошкодження логіки моделі.

Введення даних: Biggio та ін. [30] запропонували атаку на основі градієнтного підйому на основі SVM, яка атакує вхідні дані, що призводить до максимізації помилки неопуклої поверхні та збільшення класифікації класифікатора під час тестування. Гу та ін. [31] запропонував BadNet, який виконує змагальні атаки, виявляючи закриту нейронну мережу або BadNet. Атака заснована на повному або частковому відданому на аутсорсинг навчальному процесі, коли зловмисник надає користувачеві навчену модель із бекдором, який спричиняє цілеспрямовану неправильну класифікацію та погіршує точність, у деяких випадках називається тригером бекдору. Наприклад, при автономному водінні зловмисник надає користувачеві детектор дорожніх знаків із заднім ходом, який у більшості випадків добре класифікує знак зупинки, за винятком випадків, коли знаки зупинки мають особливу наклейку, що класифікує його як знаки обмеження швидкості. Цей тип атаки відбувається за двома сценаріями: користувач передає навчену модель або завантажує попередньо навчену модель.

Маніпуляція даними: атака зіткнення функцій, запропонована Шафахі та ін. [32] представляє атаку отруєння водяних знаків, засновану на оптимізації для створення атаки з чистою міткою для націлювання на поведінку класифікатора нейронної мережі на певному екземплярі. Ця атака використовує розширені методи збереження, щоб ускладнити виявлення.

Атаки Oracle: під час змагальної атаки типу оракула, супротивник, якому надано доступ до моделі з передбаченням оракула, викрадає копію віддалено розгорнутої моделі машинного навчання. Це дає йому змогу дублювати функціональність моделі, тобто «вкрасти модель» [33].

Ця атака стає все більш поширеною через збільшення пропозицій машинного навчання як послуги «MLaaS», коли кілька компаній, які пропонують хмарні послуги машинного навчання, наприклад, Google, Amazon і BigML надають прості веб-API для керування взаємодією з клієнтами.

Інверсійні атаки: Фредріксон та ін. [34] викрили проблеми конфіденційності з наданням доступу до API машинного навчання. Їхнє

дослідження продемонструвало, як супротивник може використовувати інформацію про довіру моделі, щоб призвести до атак інверсії моделі. Атака, яка реалізується як функція, яка називається MI-Face attack, дозволяє противнику витягувати зображення суб'єктів із навченої моделі машинного навчання.

Атаки висновку: Фредріксон та ін. [34] запропонував атаку висновку атрибутів, яку можна було б запустити як атаку білого ящика або чорної скриньки.

Напади екстракції: Correia-Silva та ін. [35] продемонстрували, як супротивник може створити замінну модель з моделі згорткової нейронної мережі (CNN) чорного ящика, запитуючи модель чорного ящика з випадковими неміченими даними. Більш інтригуючим аспектом цього типу атаки вилучення оракула є той факт, що набір даних, який використовувався для переконання моделі, не був пов’язаний з початковою проблемною областю. Orekondy та ін. [36] запропонував підбірні мережі, які здатні вкрасти функціональність повністю навченої моделі, використовуючи двоетапний підхід. Противник спочатку отримує прогнози від моделі, запитуючи набір вхідних даних, потім пари дані-передбачення використовуються для створення замінної моделі, відомої як модель «підробка». У їхньому підході використовується підхід навчання з підкріпленням із продемонстрованою ефективністю запитів та підвищенням продуктивності порівняно з іншими атаками типу oracle. Jagielski та інші [37] запропонували атаки Functionally Equivalent Extraction (FEE), які досліджують цілі точності та вірності в просторі вилучення моделі шляхом підвищення ефективності запитів атак навчання. Продемонстровано, що їхній метод практичний для моделей з високими 13 параметрами в діапазоні мільйонів. У їх методі атаки створюється змагальна модель, архітектура та ваги якої ідентичні оракулу.

На останок приведемо систематику(таксономію) змагальних атак рис. 1.3.

Рисунок 1.3 – Систематика змагальних атак.

1.2 Аналітичний огляд методів захисту від змагальних атак

Багато останніх засобів захисту вдаються до схем рандомізації для пом’якшення ефектів протилежних збурень у домені вхідних/областивих властивостей. Інтуїція, що стоїть за цим типом захисту, полягає в тому, що DNN завжди стійкі до випадкових збурень. Захист, заснований на рандомізації, намагається випадковим чином розділити протилежні ефекти на випадкові ефекти, які не є проблемою для більшості DNN. Захист на основі рандомізації досяг порівнянної продуктивності у налаштуваннях чорного ящика та сірого ящика, але в налаштуваннях білого ящика метод EoT [38] може скомпрометувати більшість із них, враховуючи процес рандомізації в процесі атаки. У цьому розділі ми представляємо деталі кількох типових засобів

захисту на основі рандомізації та представляємо їхню ефективність проти різних атак у різних умовах.

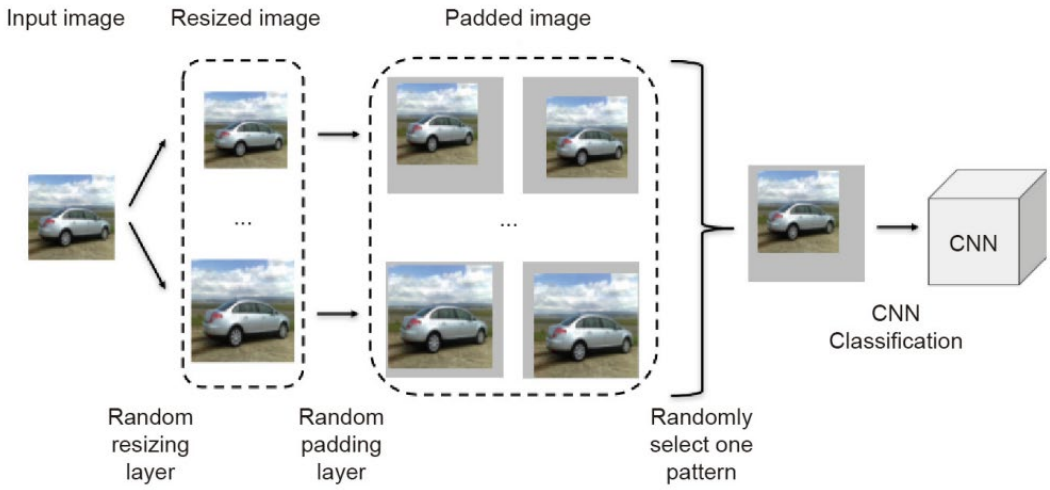


Рисунок 1.4 – Конвеєр захисного механізму на основі рандомізації, запропонований Xie та ін. [39]: розмір вхідного зображення спочатку змінюється випадковим чином, а потім довільно доповнюється.

Випадкове вхідне перетворення. Xie та ін. [39] використовують дві випадкові трансформації — випадкову зміна розміру та заповнення — щоб пом’якшити протилежні ефекти під час висновку. Випадкове зміна розміру відноситься до зміни розміру вхідних зображень до випадкового розміру перед подачею їх у DNN. Випадкове заповнення відноситься до довільного заповнення нулями навколо вхідних зображень. Трубопровід цього швидкого і гострого механізму показаний на рис. 1.4 [39]. Механізм досягає неабиякої продуктивності в умовах боротьби з «чорним ящиком» і зайняв друге місце в змаганні із змагальними прикладами NIPS 2017. Однак у налаштуваннях білого ящика цей механізм був скомпрометований методом EoT [38]. Зокрема, шляхом апроксимації градієнта за допомогою ансамблю з 30 випадково змінених і доповнених зображень, EoT може знизити точність до 0 з 8/255 збурень L1. Крім того, Guo та ін. [40] застосовують перетворення зображень із випадковістю, такі як зменшення бітової глибини, стиснення JPEG, мінімізація

повної дисперсії та прошивання зображення перед подачею зображення на CNN. Цей метод захисту протистоїть 60% сильних сірих ящиків і 90% сильних чорних скриньок, створених різними методами атаки. Однак він також скомпрометований методом EoT [38].

Випадкові шуми. Liu та ін. [41] пропонують захищати протилежні збурення за допомогою механізму випадкових шумів, званого випадковим самоаналізом (RSE). RSE додає шар шуму перед кожним шаром згортки як на етапі навчання, так і на етапі тестування, і об'єднує результати прогнозу над випадковими шумами, щоб стабілізувати вихідні дані DNN, як показано на рис. 1.5 [41].

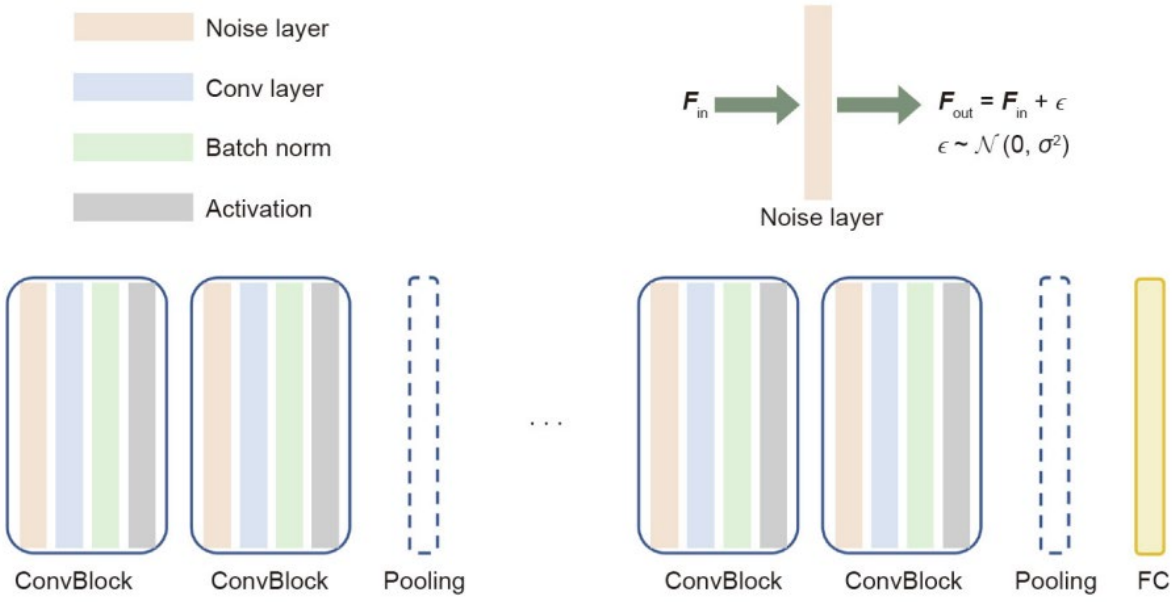


Рисунок 1.5 – Архітектура RSE [41]. FC: повністю підключений шар; F_{in}: вхідний вектор шумового шару; F_{out}: вихідний вектор шумового шару; збурення, яке відповідає Гауссовому розподілу $N(0, \sigma^2)$; conv: згортка.

Lecuyer та ін. [42] розглядають захисний механізм від випадкових шумів під кутом диференціальної конфіденційності (DP) і пропонують захист на основі DP під назвою PixelDP. PixelDP включає шар шумів DP усередині DNN, щоб встановити межі DP щодо варіації розподілу за його прогнозами вхідних даних з невеликими, заснованими на нормі збуреннями. PixelDP можна

використовувати для захисту від атак L1/L2 за допомогою механізмів DP Лапласа/Гауса. Натхненний PixelDP, автори в [43] далі пропонують безпосередньо додавати випадковий шум до пікселів протилежних прикладів перед класифікацією, щоб усунути ефекти протилежних збурень. Дотримуючись теорії дивергенції Реньї, він доводить, що цей простий метод може обмежувати зверху розмір протилежного збурення, до якого він стійкий, що залежить від першої та другої за величиною ймовірностей розподілу ймовірності вихідного сигналу (вектора).

Звичайна вхідна ректифікація. З метою пом'якшення побічних ефектів Xu та ін. [44] спочатку використовують два методи стиснення (знищення шумів) — біт-зменшення та розмиття зображення — щоб зменшити ступінь свободи та усунути суперечливі збурення, як показано на рис. 1.6 оригінальні та стиснуті зображення.

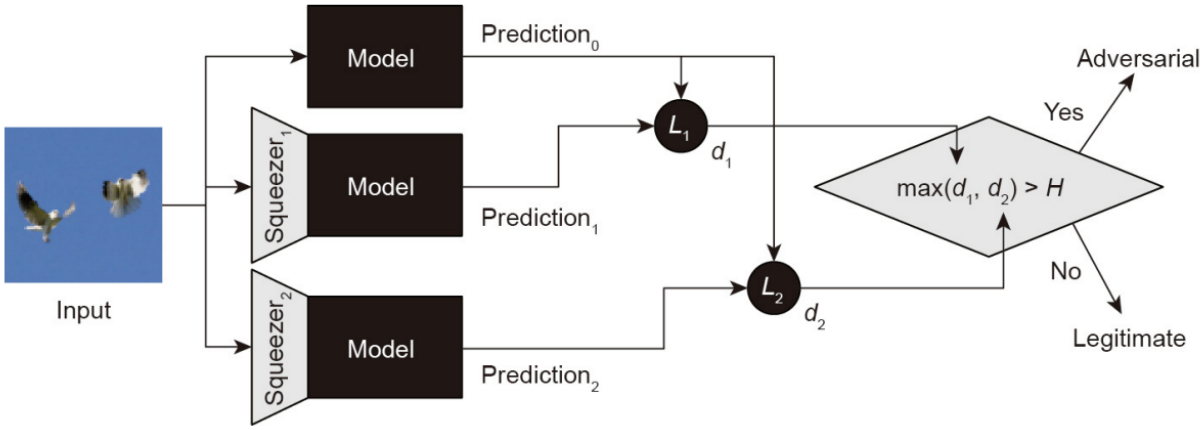


Рисунок 1.6 – Структура стиснення функцій, запропонована Xu та інші [44]. *d1* і *d2*: різниця між прогнозом моделі на стиснутому вході та її прогнозом на оригінальний вхід; *H*: поріг, який використовується для виявлення змагальних прикладів.

Якщо вихідні та стиснуті вхідні дані дають істотно відмінні від моделі вихідні дані, вихідний вхід, ймовірно, буде змагальною вибіркою. Xu та ін. [47] далі показують, що методи стиснення ознак, запропоновані в [44] може

пом'якшити атаку C&W. Однак Не та ін. [45] демонструють, що стиснення функцій все ще вразливе для адаптивного обізнаного супротивника. Він приймає втрату CW2 як втрату протиборства. Після кожного кроку процедури оптимізації оптимізатор отримує проміжне зображення. Версія цього проміжного зображення зі зменшеною глибиною кольору перевіряється системою виявлення, запропонованою Xu та ін. [44]. Така процедура оптимізації виконується кілька разів, і всі проміжні змагальні вибірки, які можуть обійти систему Сю, об’єднуються. Вся ця адаптивна атака може порушити вхідну систему стиснення з збуреннями, набагато меншими, ніж ті, що заявлені в [44]. Більше того, Шарма і Чен [46] також показують, що EAD і CW2 можуть обійти вхідну систему стиснення зі збільшенням сили супротивника.

Очищення вхідних даних на основі GAN. GAN є потужним інструментом для вивчення генеративної моделі для розподілу даних. Таким чином, багато робіт мають намір використовувати GAN для вивчення доброякісного розподілу даних з метою створення доброякісної проекції для змагального введення. Два типових алгоритму серед усіх цих робіт є Defense-GAN і GAN для усунення збурень (APE-GAN). Defense-GAN [48] тренує генератор для моделювання розподілу доброякісних зображень, як показано на рис. 1.7 [48]. На етапі тестування Defense-GAN очищає супротивний вхід, шукає зображення, близьке до вхідного, у його вивченому розподілі, і передає це доброякісне зображення в класифікатор. Цю стратегію можна використовувати для захисту від різних атак суперництва. На даний момент найефективніша схема атаки проти Defense-GAN заснована на диференційованій апроксимації зворотного проходу [49], яка може знизити її точність до 55% з 0,005 L2 протилежних збурень. APE-GAN [50] безпосередньо вивчає генератор, щоб очистити змагальний зразок, використовуючи його як вхід, і виводить доброякісний аналог. Хоча APE-GAN досягає хороших показників на тестовому стенді [50], адаптивна атака з білим ящиком CW2, запропонована в [51] може легко перемогти APE-GAN.

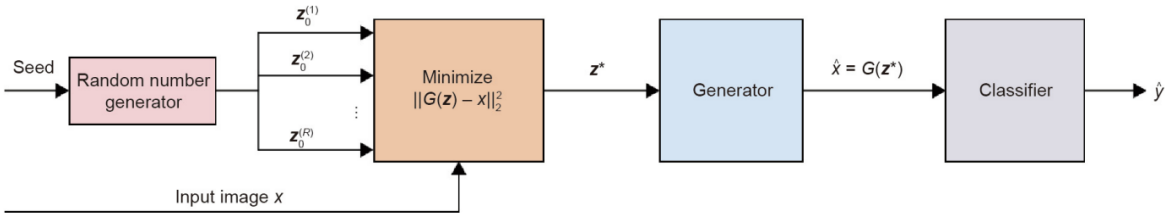


Рисунок 1.7 – Трубопровід Defense-GAN [48]. *G*: генеративна модель, яка може генерувати вхідну вибірку високої розмірності з вектора *z* низькою розмірністю; *R*: кількість випадкових векторів, створених генератором випадкових чисел.

Доказові захисти. Усі перераховані вище засоби захисту є евристичними, а це означає, що ефективність цих захистів перевіряється лише експериментально, а не теоретично. Без теоретичної гарантії рівня помилок ці евристичні захисти можуть бути порушені новою атакою в майбутньому. Тому багато дослідників докладають зусиль до розробки доказуваних методів захисту, які завжди можуть підтримувати певну точність під час чітко визначеного класу атак. Далі представлені кілька типових сертифікованих захистів.

Сертифікований захист на основі напіввизначеного програмування. Рагхунатан і Колтер [52] вперше пропонують сертифікований метод захисту від протиборчих прикладів у двошарових мережах. Автори виводять напіввизначену релаксацію для верхньої межі протиборчої втрати та включають розслаблення у втрату на тренування як регуляризатор. Цей метод навчання створює мережу з сертифікатом, що жодна атака з не більше ніж 0,1/1,0 збуреннями L1 не може спричинити помилку тесту понад 35% на MNIST. У [53], Рагхунатан та ін. далі пропонують нове напіввизначене послаблення для сертифікації довільних мереж ReLU. Нещодавно запропонована релаксація є більш жорсткою, ніж попередня, і може забезпечити значущі гарантії надійності в трьох різних мережах.

Доказовий захист, заснований на подвійному підході. Поряд з [52], Вонг і Кольтер [54] формулюють подвійну задачу для визначення верхньої межі

змагального многогранника. Вони показують, що подвійну проблему можна вирішити шляхом оптимізації над іншою глибокою нейронною мережею. На відміну від [52], який застосовується лише до двошарових повністю підключених мереж, цей підхід можна застосувати до глибоких мереж з довільними лінійними операторними рівнями, такими як шари згортки. Автори далі розширюють методику в [54] до набагато більш загальних мереж із пропуском зв'язків і довільними нелінійними активаціями в [55]. Вони також представляють техніку нелінійної випадкової проекції для оцінки межі таким чином, що розмір прихованих одиниць лише лінійно масштабується, що робить цей підхід застосовним до більших мереж. Як на наборах даних MNIST, так і на CIFAR, запропонована робота тренує класифікатори за допомогою запропонованих методів, які істотно покращують попередні доказові гарантії надійної змагальної помилки: з 5,8% до 3,1% на MNIST із L_∞ збуреннями $\epsilon = 0{,}1$ і з 80% до 36,4% на CIFAR із L_∞ збуреннями $\epsilon = 2/255$.

Сертифікація надійності розподілу(робустності). З точки зору оптимізації розподілу, Sinha та ін. [56] формулюють задачу оптимізації над змагальними розподілами так: $\min_\Theta \sup E_\varphi [J(\Theta, x, y)]$ де φ – набір-кандидат для всіх розподілів навколо доброякісних даних, які можна побудувати за допомогою куль f–розбіжності або куль Вассерштейна, φ відбирається з набору кандидатів φ. Оптимізація щодо цієї мети розподілу еквівалентна мінімізації емпіричного ризику для всіх вибірок, сусідніх із доброякісними даними, тобто всіх кандидатів на змагальні вибірки. Оскільки φ впливає на обчислюваність, а пряма оптимізація за довільним φ є нерозв'язною, робота в [50] виводить множини, що піддаються обговоренню, φ використовуючи метрику відстані Вассерштейна з обчислювально ефективними релаксаціями, які обчислюються навіть коли $J(\Theta, x, y)$ неопукла. Фактично робота в [56] також забезпечує змагальну процедуру навчання з доказовими гарантіями щодо її обчислювальної та статистичної продуктивності. У запропонованій процедурі навчання він включає покарання для характеристики області стійкості до змагальності. Оскільки оптимізація над цим штрафом невиправна,

автори пропонують послаблення по Лагранжу для штрафу та досягають надійної оптимізації щодо запропонованої втрати розподілу. Крім того, автори отримують гарантії для емпіричного мінімізатора надійної проблеми сідла і дають спеціалізовані межі для проблем адаптації домену, що також проливає світло на сертифікацію надійності розподілу.

Проаналізувавши предмет дослідження, методи створення змагальних атак та захист від них, сучасні тенденції та проблематику, було вирішено розробити алгоритм детектування змагальних атак на модель штучного інтелекту з використанням Trust Score, FGSM – атаки та ART та довільної згорткової мережі для класифікації зображень.

1.3 Формальна постановка задачі

Головне завдання цієї роботи полягає в розробці алгоритму детектування змагальних атак на модель штучного інтелекту. Для вирішення даного завдання у роботі пропонується вирішити ряд наступних задач:

– проаналізувати сучасний стан та тенденції розвиту алгоритмів захисту від змагальних атак;

– побудувати модель аналізу даних і сформувати для неї навчальну і тестову вибірки;

– навчити модель аналізу даних;

– розробити алгоритм генерації змагальних атак;

– розробити алгоритм детектування змагальних атак;

–дослідити ефективність детектування змагальних атак;

– зробити висновки.

Прикладом моделі штучного інтелекту для аналізу даних може бути будь-яка згорткова мережа для класифікації зображень, оскільки це найбільш поширена і вразлива до атак модель.

2 ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ ЗАХИСТУ ВІД ЗМАГАЛЬНИХ АТАК

2.1 Модель розпізнавання зображень з захистом від змагальних атак

Розроблена модель включає в себе базову нейронну згорткову мережу, TrustScore та аутоенкодер рис. 2.2. Навчання, тестування та робота з даними проводиться на основі бази даних MNIST.

Був розроблений детектор атаки рис. 2.1, який виявляє коли класифікатор в змозі впевнено видати прогноз, решта випадків розцінюється як шум, або зразки з втратою важливої інформації. Ми відмовляємось від прийняття рішень стосовно зразків, що не задовольняють цим умовам. На точність, відкинуть зразки, не впливатимуть. Отримані результати наведені в розділі 3.

```
accepted_decisions = np.logical_and( (y_pred_prob > 0.9), (trust_score > 1) )
```

Рисунок 2.1 – Розроблений детектор.

Після проведення тестів на незбурених даних, послідовно створюємо змагальні зразки з FGM і проводимо тестування з їх використанням, кожного разу, збільшуючи величину збурення epsilon.

Використовуючи системи штучного інтелекту для тих чи задач, абсолютно необхідно знати, коли можна довіряти прогнозам класифікатора машинного навчання, адже ціна помилки може бути дуже високою у всіх відношеннях, залежно від області, де використовується ШІ. Покладатися на невідкалібровані, непровірені ймовірності передбачення класифікатора не є надійним та оптимальним варіантом і може бути покращено. TrustScore вимірює узгодження між класифікатором і модифікованим класифікатором найближчого сусіда в тестовому наборі. Оцінка довіри (TrustScore) — це відношення між відстанню тестового екземпляра до найближчого класу, що

відрізняється від прогнозованого класу, і відстанню до передбаченого класу. Вищі бали відповідають більш надійним прогнозам. Оцінка 1 означає, що відстань до прогнозованого класу така ж, як і до іншого класу.

Оцінки довіри найкраще працюють для об’єктів малого та середнього розміру. В даній роботі оцінка довіри застосовується до високорозмірних даних – зображень. Реалізовано це через додавання додаткового етапу попередньої обробки у вигляді автоматичного аутоенкодера для того, щоб зменшити розмірність.

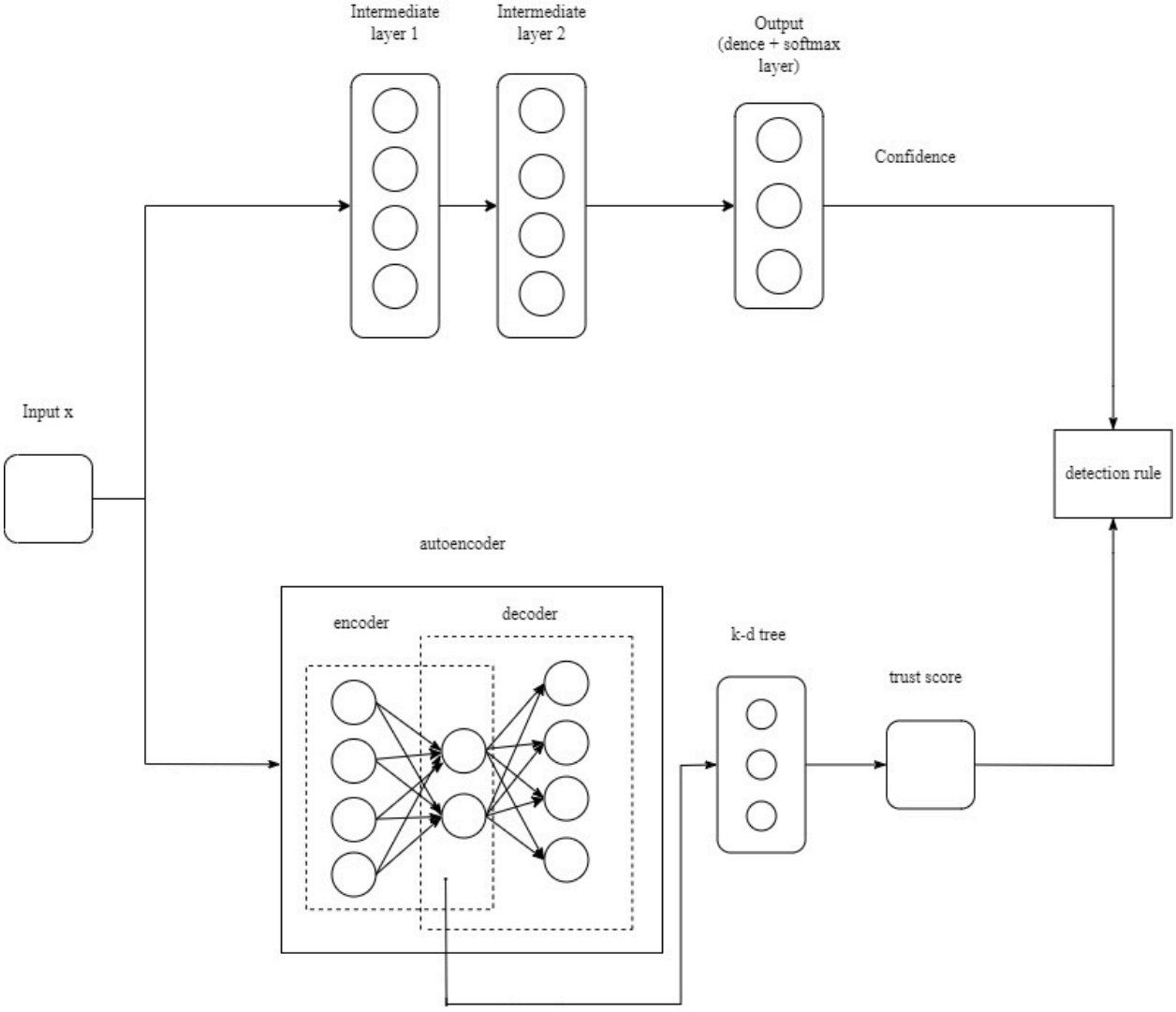


Рисунок 1.2 – Схема розробленої моделі.

Автокодувальник, аутоенкодер — спеціальна архітектура штучних нейронних сетей, що дозволяє застосовувати навчання без учителя при

використанні методу зворотного поширення помилок. Найпростіша архітектура аутоенкодера – мережа прямого поширення, без зворотних зв'язків, найбільш схожа з перцептроном і містить вхідний шар, проміжний шар і вихідний шар. На відміну від перцептрона, вихідний шар аутоенкодера повинен містити стільки ж нейронів нейронов, як і вхідний шар. Основний принцип роботи і навчання аутоенкодера полягає в отриманні відклику на вихідному шарі, найбільш близького до вхідного. Щоб рішення не виявилося тривіальним, на проміжний шар накладають обмеження: проміжний шар повинен мати або меншу розмірність, ніж вхідний і вихідний шари, або штучно обмежується кількість одночасно активних нейронів проміжного шару — розрізнена активація. Ці обмеження заставляють нейромережу шукати узагальнення та кореляцію в поступаючих на вхід даних, виконувати їх зжаття. Таким чином, нейромережа автоматично навчається виділяти з вхідних даних загальні ознаки, які кодуються у значеннях ваг штучної нейронної мережі. Так, при навчанні мережі на набір різних вхідних зображень, нейромережа може навчитися самостійно розпізнавати лінії та смуги під різними кутами [64].

MNIST (Модифікована база даних Національного інституту стандартів і технологій) — база даних рукописних цифр, яка використовується для навчання різноманітних систем обробки, класифікації та розпізнавання зображень. Широко використовується для навчання та тестування в області машинного навчання. Створена шляхом «повторного змішування» зразків із оригінальних наборів даних NIST. Розробники вважали, що, оскільки навчальний набір NIST був взятий у співробітників Американського бюро перепису населення, в той час як набір даних для тестування взятий з американських старшокласників, він не дуже підходить для експериментів з машинним навчанням. Крім того, чорно-білі зображення від NIST були нормалізовані, щоб вони вписувалися в обмежувальну рамку розміром 28x28 пікселів, і згладжені, що вводило рівні градації сірого. База даних MNIST містить 60 000 навчальних зображень і 10 000 тестових зображень. Половина навчального набору та половина тестового набору були взяті з набору

навчальних даних NIST, тоді як інша половина навчального набору та інша половина тестового набору були взяті з набору даних тестування NIST. Початкові творці бази даних ведуть список деяких методів, перевірених на ній. Розширений набір даних, подібний до MNIST під назвою EMNIST, був опублікований у 2017 році, який містить 240 000 навчальних зображень і 40 000 тестових зображень рукописних цифр і символів [63].

2.2 Метод навчання для розпізнавання від змагальних атак

Для початку ми імпортуємо всі необхідні бібліотеки, опис яких наведено у розділі 3. Завантажуємо та нормалізуємо дані MNIST, будуємо базову CNN і навчаємо її на чистих даних. Будуємо модель аутоенкодера, TrustScore та тренуємо їх, знову ж таки, на чистих даних.

Ключ полягає в тому, щоб підігнати та обчислити показники довіри для закодованих екземплярів. Закодовані дані все ще потрібно змінити щоб відповідати формату дерева k-d. Це обробляється внутрішньо.

Підхід складається з двох кроків, описаних в Алгоритмі 1 і 2. Спочатку ми попередньо обробляємо тренувальні дані, як описано в алгоритмі 1, щоб знайти набір α-високої щільності кожного класу, який визначається як навчальні вибірки в межах цього класу після фільтрації α-фракції вибірок з найменшою щільністю(що може бути винятковим) [].

Визначення 1 (α-набір високої щільності). Нехай $0 \leq \alpha < 1$ і f — неперервна функція густини з компактною опорою $X \subseteq R^D$. Потім визначимо $H_\alpha(f)$, α-набір f з високою щільністю, як набір рівня λ_α f, визначений як $\{x \in X : f(x) \geq \lambda_\alpha\}$ $де$ $\lambda_\alpha := \inf \{\lambda \geq 0 : \int_X 1[f(x) \leq \lambda] f(x)dx \geq \alpha\}$ [].

Щоб апроксимувати набір α-високої щільності, Алгоритм 1 фільтрує α-частку точок найменшою емпіричною щільністю, на основі k-найближчих сусідів. Цей крок фільтрації даних не залежить від наданого класифікатора h. Потім другий крок: надавши тестову вибірку, ми визначаємо її оцінку довіри як співвідношення між відстанню від досліджуваного зразка до α-набору

високої щільності найближчого класу, відмінного від прогнозованого класу і відстанню від тестового зразка до α-набору з високою щільністю передбаченого класу на h, як детально описано в Алгоритмі 2. Передбачається, що якщо класифікатор h передбачає мітку, яка значно далі, ніж найближча мітка, то це попередження про те, що класифікатор може помилятися. Таким чином, нашу процедуру можна розглядати як порівняння з модифікованим класифікатором найближчих сусідів, де модифікація полягає в початковій фільтрації точок, які не входять до α-набору високої щільності для кожного класу [65].

Алгоритм 1. Оцінка α-набору високої щільності.

Параметри: α (поріг щільності), k

Входи: $X := \{x\,1, .., x\,n\}$, отримані з f

Визначте радіус k-NN $r_k(x) := \inf\{r > 0 : |B(x, r) \cap X| \geq k\}$ і нехай $\varepsilon := \inf\{r > 0 : |\{x \in X : r_k(x) > r\}| \leq \alpha \cdot n\}$

Return $\widehat{H}_\alpha(f) := \{x \in X : r_k(x) \leq \varepsilon\}$.

Алгоритм 2. Оцінка довіри.

Параметри:

Вхід: Класифікатор $h : X \to Y$. Дані навчання (x_1, y_1), ..., (x_n, y_n). Тестовий приклад x.

Для кожного $l \in Y$, дамо $\widehat{H}_\alpha(f_\alpha)$ бути виходом алгоритму 1 з параметрами α, k та точками вибірки $\{x_j : 1 \leq j \leq n, y_j = l\}$. Тоді, отримуємо оцінку довіри, яка визначається як $\xi(h, x) := d(x, \widehat{H}_\alpha(f_{\tilde{h}}))(x))/d(x, \widehat{H}_\alpha(f_h))(x)))$, де $\tilde{h}(x) = \operatorname{argmin}_{l \in Y, l \neq h(x)} d(x, \widehat{H}_\alpha(f_l))$ [65].

Метод має два гіпер параметри: k (кількість сусідів, наприклад в k-NN) і α (частка даних для фільтрації), щоб обчислити емпіричні щільності. Теоретично ми показується, що k може лежати в широкому діапазоні і все ще дає нам гарантії бажаної послідовності. Протягом наших експериментів ми фіксуємо $k = 5$, і використовуємо перехресну перевірку, щоб вибрати α, оскільки він залежить від даних [65].

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Формування вхідного математичного опису

В якості вибірки даних для навчання, створення змагальних зразків і тестування, був обраний MNIS. Вона є типовою для перевірки концепцій в галузі машинного навчання, комп’ютерного зору. Візуальний вигляд деяких зразків вибірки наведений на рис. 3.1.

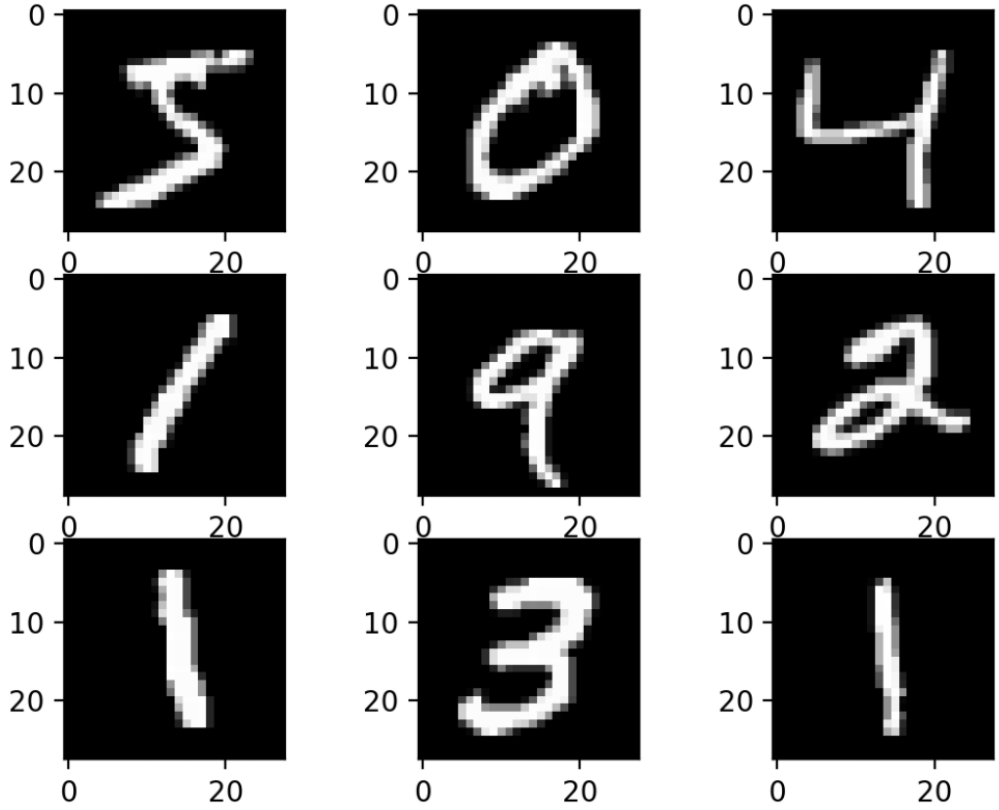


Рисунок 3.1 – Приклад зразків вибірки MNIST.

Вибірка містить являє собою набір із 70000 зображень розміном 28x28 пікселів, з яких 60000 і 10000 є навчальною те тестовою вибіркою відповідно. При завантажені вони мають форму (60000, 28, 28) (60000,) і (10000, 28, 28) (10000,). Проводимо їх нормалізацію, масштабування і категоризацію та приводимо до виду (60000, 28, 28, 1) (60000, 10) і (10000, 28, 28, 1) (10000, 10)

відповідно для тренувальної та тестової вибірок. Програмна реалізація приведена на рис. 3.2.

```
In [ ]: x_train = x_train.astype('float32') / 255
        x_test = x_test.astype('float32') / 255
        x_train = np.reshape(x_train, x_train.shape + (1,))
        x_test = np.reshape(x_test, x_test.shape + (1,))
        print('x_train shape:', x_train.shape, 'x_test shape:', x_test.shape)
        y_train = to_categorical(y_train)
        y_test = to_categorical(y_test)
        print('y_train shape:', y_train.shape, 'y_test shape:', y_test.shape)

        xmin, xmax = -.5, .5
        x_train = ((x_train - x_train.min()) / (x_train.max() - x_train.min())) * (xmax - xmin) + xmin
        x_test = ((x_test - x_test.min()) / (x_test.max() - x_test.min())) * (xmax - xmin) + xmin

        x_train shape: (60000, 28, 28, 1) x_test shape: (10000, 28, 28, 1)
        y_train shape: (60000, 10) y_test shape: (10000, 10)
```

Рисунок 3.2 – Програмна реалізація перетворення на нормалізації вибірки.

3.2 Короткий опис програмного забезпечення

Проект був виконаний на мові програмування Python 3 з використанням сумісних бібліотек, необхідних для виконання поставленого завдання, задля пришвидшення роботи програмний код виконувався в хмарному середовищі Google Colaboratory.

Python – це інтерпретована, об’єктно-орієнтована мова програмування високого рівня, семантика якого є динамічною. Його вбудовані структури даних високого рівня у поєднанні з динамічною типізацією та динамічною прив’язкою роблять його дуже привабливим для користувачів в цілях швидкої розробки додатків, а також для використання в якості мови сценаріїв для написання скриптів або як «мову склеювання» для з’єднання уже існуючих компонентів разом. Простий синтаксис Python, який легко піддається вивченню, підкреслює читабельність і, отже, знижує витрати на підтримку та обслуговування програми. Також, Python підтримує модулі та пакунки, що, як наслідок, сприяє модульності програм і повторному використанню коду. Інтерпретатор Python разом з великою стандартною бібліотекою доступні у вихідній або двійковій формі на безкоштовній основі для всіх основних платформ і можуть вільно поширюватися. Нерідко, причиною високої

прихильності програмістів до Python є підвищена продуктивність, яку він зазвичай забезпечує. По причині відсутності, власне, кроку компіляції, як такого, цикл редагування-тестування-налагодження в Python є надзвичайно швидким. У Python досить легко налагоджувати програми: помилка або некоректне введення ніколи не стане причиною помилки сегментації. Натомість, у разі виявлення інтерпретатором помилки, він створюється виняток. У випадку, коли програма не вловлює виняток, інтерпретатор виводить користувачеві трасування стека. Дебагер рівня вихідного коду надає змогу перевіряти локальні та глобальні змінні, давати оцінку довільним виразам, встановлювати точки зупину, просуватися кодом рядок через рядок і таке інше. Власне, сам дебагер написаний на Python, що свідчить про інтроспективну силу мови Python. В той же час, часто найшвидшим способом налагодження програми є додавання декількох операторів print до джерела: швидкий цикл редагування-тестування-налагодження робить даний простий підхід високоефективним [57].

TensorFlow — це Python–friendly бібліотека з відкритим вихідним кодом для цифрових обчислень, яка робить машинне навчання для користувачів різного рівня підготовки швидшим і простішим. Машинне навчання – складна дисципліна. Але на сьогоднішній день, впровадження моделей машинного навчання є набагато менш лякаючим і складним, ніж як це було раніше. Все це в тому числі завдяки фреймворкам машинного навчання, наприклад, таким як TensorFlow від Google, які значно полегшують процес добутку і отримання даних, навчання моделей, обслуговують прогнози та уточнюють майбутні результати. Розроблена командою Google Brain, TensorFlow – це open source бібліотека для чисельних обчислень і машинного навчання великих масштабів. TensorFlow об’єднує в собі безліч моделей і алгоритмів машинного навчання та глибокого навчання (нейронних мереж). Він використовує Python, для забезпечення зручного інтерфейсу API для створення застосунків з фреймворком, в той час як, виконуються ці додатки високопродуктивній мові C++ [62].

TensorFlow може навчати та запускати глибокі нейронні мережі які виконують функції класифікації рукописних цифр, розпізнавання зображень, вбудовування слів, рекурентних нейронних мереж, моделей від послідовності до послідовності для машинного перекладу, обробки природної мови та моделювання процесів на основі PDE (диференціальне рівняння з частковими частинами). Краще всього, що TensorFlow підтримує створення прогнозів в масштабі, з тими ж моделями, що використовуються для навчання [62].

TensorFlow розробникам надає можливість створювати графи потоків даних — структури, які описують, яким чином дані переміщаються по графу або по рядам вузлів обробки. Кожен вузол на графі представляє собою математичну операцію, а кожне з'єднання чи ребро між вузлами є багатовимірним масивом даних чи тензором. TensorFlow надає все це для програміста за допомогою мови Python. Python легкий у вивченні та роботі з ним, а також надає комфортні способи виразити, як можна об'єднати разом абстракції високого рівня. Вузли та тензори в TensorFlow є об'єктами Python, програми TensorFlow теж є додатками Python. Однак фактичні математичні операції виконуються не в Python. Бібліотеки перетворень, які доступні через TensorFlow, записуються як високопродуктивні двійкові файли на мові C++. Python просто направляє трафік між частинами і надає абстракції програмування високого рівня, щоб об'єднати їх разом. Програми TensorFlow можуть запускатися майже на будь-якій зручній цілі: локальній машині, кластері в хмарі, пристроях на операційній системі iOS чи Android, процесорах або на графічних процесорах. Якщо ж ви використовуєте хмарне середовище Google, то ви можете запустити TensorFlow на кастомному силіконовому блоці обробки TensorFlow (TPU) від Google для подальшого прискорення. Однак отримані моделі, створені TensorFlow, можна розгорнути на будь-якому пристрої, на якому вони будуть використовуватися для прогнозування [62].

Scikit-learn (Sklearn) — бібліотека для машинного навчання на Python з відкритим вихідним кодом. Надає вибір ефективних інструментів для машинного навчання та статистичного моделювання, включаючи

класифікацію, регресію, кластеризацію та зменшення розмірності через інтерфейс узгодженості в Python. Ця бібліотека, яка в основному написана на Python, побудована на NumPy, SciPy і Matplotlib. Основні її поняття та функції включають:

- алгоритмічні методи прийняття рішень:
  - класифікація: визначення та категоризація даних на основі шаблонів;
  - регресія: прогнозування або прогнозування значень даних на основі середнього середнього значення наявних і запланованих даних;
  - кластеризація: автоматичне групування подібних даних у набори даних.
- алгоритми, які підтримують прогнозний аналіз ранжування, починаючи від простої лінійної регресії до розпізнавання патернів нейронної мережі;
- сумісність з бібліотеками NumPy, pandas і matplotlib.

Keras — це бібліотека ПЗ з відкритим вихідним кодом, яка надає інтерфейс Python для штучних нейронних мереж. Власне кажучи, Keras виступає в якості інтерфейсу для бібліотеки TensorFlow. Його перевагами є зручність, модульність та розширювальна здатність. Він забезпечує швидке експериментування з глибокими нейронними мережами Keras містить в собі численні реалізації типових складових блоків нейронної мережі, наприклад, таких як шари, оптимізатори та безліч інструментів, цілі, функції активації, які в результаті полегшують роботу користувача з графічними та текстовими даними для того, щоб спростити написання коду, необхідного для глибокої нейронної мережі. Код розміщено на GitHub, а форуми підтримки спільноти включають сторінку питань GitHub та канал Slack. Окрім стандартних нейронних мереж, Keras підтримує згорткові та рекурентні нейронні мережі. Також в ньому доступні загальні утиліти, такі як об'єднання, вилучення і нормалізація пакетів. Keras дозволяє користувачам створювати глибокі моделі

на мобільних пристроях різних операційних систем (iOS і Android), у браузері чи на віртуальній машині Java. В добавок, Keras надає можливість використання розподіленого навчання моделей глибокого навчання на кластерах графічних процесорів (GPU) та тензорних процесорів (TPU) [58].

Matplotlib — це бібліотека для створення статичних, анімованих та інтерактивних графічних візуалізацій на Python. Він дозволяє:

- створювати графіки якості публікації;
- робити інтерактивні фігури, які можна масштабувати, панорамувати, оновлювати;
- налаштовувати візуальний стиль і макет;
- експортувати в багато форматів файлів;
- вбудовувати в JupyterLab і графічні інтерфейси користувача.
- використовувати обширний набір пакетів від сторонніх розробників, створених на Matplotlib

Matplotlib робить прості речі легкими, а важкі – можливими [59].

NumPy — це фундаментальний пакет призначений для наукових обчислень на Python. Це бібліотека Python, яка надає об’єкт багатовимірного масиву, різноманітні похідні об’єкти (масковані масиви і матриці), а також набір підпрограм для швидких операцій над масивами, включаючи математичні, логічні, маніпуляції з формою, сортування, вибір, введення/виведення. , дискретні перетворення Фур'є, базова лінійна алгебра, основні статистичні операції, випадкове моделювання та багато іншого. Ядром NumPy виступає об'єкт ndarray, це інкапсулює n-вимірні масиви однорідних типів даних, при цьому, для підвищення продуктивності, операції виконуються у скомпільованому коді [60].

Виділяють декілька важливих відмінностей між масивами NumPy і стандартними послідовностями Python:

- масиви NumPy мають фіксований розмір під час створення, на відміну від списків Python, що можуть зростати динамічно. Зміна розміру ndarray створює новий масив і видаляє оригінал;

- усі елементи в масиві NumPy повинні бути одного типу даних і, таким чином, мати однаковий розмір пам’яті. Виняток: можна мати масиви об’єктів (Python, включаючи NumPy), що дозволяє використовувати масиви елементів різного розміру;
- масиви NumPy полегшують розширені математичні та інші типи операцій над великим об’ємом даних. Зазвичай такі операції виконуються ефективніше і з меншою кількістю коду, ніж це можливо при використанні вбудованих послідовностей Python;
- все більша кількість науково-математичних пакетів на основі Python використовує масиви NumPy; хоча вони зазвичай підтримують введення послідовності Python, вони перетворюють такі вхідні дані в масиви NumPy перед обробкою, і вони часто виводять масиви NumPy. Іншими словами, для ефективного використання значної частини (можливо, навіть більшості) сучасного науково-математичного програмного забезпечення на основі Python, недостатньо володіти знаннями використання вбудованих в Python типів послідовностей – також необхідно знати, як використовувати масиви NumPy [60].

Alibi — бібліотека Python з відкритим вихідним кодом, спрямована на перевірку та інтерпретацію моделей машинного навчання. Початкова увага в бібліотеці зосереджена на поясненнях моделей на основі чорних ящиків.

Adversarial Robustness Toolbox (ART) — це Python бібліотека для безпеки машинного навчання. ART надає інструменти, які дозволяють розробникам і дослідникам оцінювати, захищати, сертифікувати та перевіряти моделі та програми машинного навчання від змагальних атак ухилення, отруєння, вилучення та висновку. ART підтримує всі популярні фреймворки машинного навчання (TensorFlow, Keras, PyTorch, MXNet, scikit-learn, XGBoost, LightGBM, CatBoost, GPy тощо), усі типи даних (зображення, таблиці, аудіо, відео тощо) та задачі машинного навчання (класифікація, виявлення об’єктів, генерація, сертифікація тощо) [61].

Colaboratory, або скорочено «Colab», — це продукт від Google Research, безкоштовне інтерактивне хмарне середовище для роботи з програмним кодом. Код виконується на віртуальній машині, яке є приватною для вашого облікового запису. Віртуальні машини видаляються, після певного часу простою, також, мають максимальний термін життя, що встановлений службою Colab. Colab дозволяє будь-кому писати та виконувати довільний код на Python через браузер, і особливо добре підходить для машинного навчання, аналізу даних та освіти. Кажучи більш технічно, Colab — це розміщена служба ноутбуків Jupyter, яка не вимагає особливого налаштування для використання, забезпечуючи безкоштовний доступ до високопродуктивних обчислювальних ресурсів, включаючи графічні та тензорні процесори, хоча й з певними обмеженнями задля рівного доступу до ресурсів усіх бажаючих. Втім, завжди можна оформити платну підписку та отримати розширений доступ до необхідних ресурсів [62].

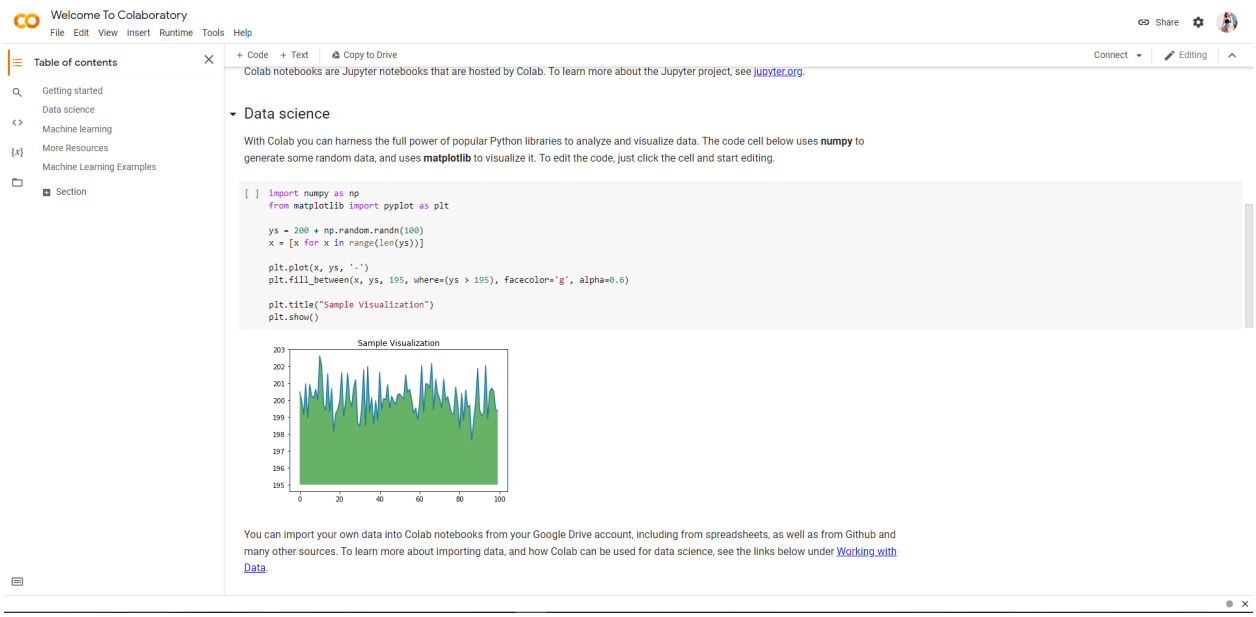


Рисунок 3.3 – Інтерфейс Google Colab.

Google Colab є чудовим інструментом і підійде усім, хто працює з Big Data, наприклад:

- аналітикам даних (сортувати дані за будь-який період, робити візуалізацію або визначати і будувати закономірності);
- дослідникам даних (розробляти та тестувати нові моделі машинного навчання, створювати прогнози, ставити експерименти);
- інженерам даних (розробляти ПЗ, системи для зберігання великих даних).

В основі «Колабораторії» — блокнот Jupyter для роботи на Python, тільки з базою на Google Диску, а не на комп’ютері. Тут же комірки (cells), які підтримують текст, формули, зображення, розмітку HTML і не тільки. Єдина умова — потрібно мати Google-аккаунт [62].

Головна особливість «Колабораторії» — безкоштовні потужні графічні процесори GPU та TPU, завдяки яким можна займатися не тільки базовою аналітичною інформацією, але й більш складними дослідженнями в області машинного навчання. З тим, що CPU вираховує годинами, GPU або TPU вправляються за хвилини або навіть секунди. В цілому, вони мають наступні відмінності:

- CPU — центральний процесор — мозку комп'ютера, який виконує операції з даними. Настільки універсальний, що може використовуватися майже для всіх завдань: від запису фотографій на флешку до моделювання фізичних процесів.
- GPU — графічний процесор. Швидше працює з даними, так як задачі виконуються паралельно, а не послідовно, як в ЦП. Він заточений під графіку, тому йому зручно працювати із зображенням і відео, наприклад, займатися 3D-моделюванням або монтажем.
- TPU — тензорний процесор, це розробка від Google. Він призначений для тренування нейромереж. У цього процесора в розрізі вище продуктивності при великих обсягах обчислювальних задач [62].

  Основні переваги використання Colab:

- дають можливість працювати з бібліотеками Python для аналізу даних в Інтернеті;
- Colab надає потужні процесори для хмарних обчислень. У нього інтуїтивно зрозумілий інтерфейс, який дає змогу не перегружати комп'ютер і робити все вирахування швидко;
- усі блокноти під рукою. У Google Colab зберігається доступ до облікового запису з будь-яких пристроїв. Правда, якщо ви з обережністю ставитесь до своєї конфіденційності, Jupyter може бути для вас більш вигідним варіантом;
- серце Colab — це спільне використання. При роботі над проектом у команді Colab дає можливість вільно редагувати, коментувати та редагувати код з різних облікових записів, де б ви не перебували;
- ще одним достоїнством колаб можна назвати — інтеграцію з GitHub. Він відкриває доступ до будь-якого сховища, якщо йому надасть профіль на сервісі.

Google Colab максимально упростив усі процеси: у них є і базові бібліотеки (NumPy, scikit-learn, Pandas), і більш складні (типу Keras, TensorFlow і PyTorch), не потрібно встановлювати програми та середні середовища, можна просто відразу писати код. Якщо ж базової бібліотеки недостатньо, завжди можна додати необхідне за допомогою інсталятора PIP і працювати далі [62].

3.3 Результати експериментів

В ході тестування традиційної моделі та автоенкодера з TrusctScore на чистих даних та на збурених зразках із різними показниками збурення були отримані наступні результати, приведені на рис.3.4 та рис.3.5, які свідчать про те, що розроблена модель розпізнавання зображень з захистом від змагальних атак, з використанням аутоенкодера та TsustScore, є значно більш точною в

розпізнаванні та стійкою до змагальних атак типу FGM у порівнянні з традиційною моделлю. Незважаючи на досить високу ефективність, при потужних збурення з високим epsilon інформація сильно пошкоджується, що робить її практично нерозпізнаваною, хоча й у цьому випадку ми спостерігаємо, що точність заснована на узгодженні з TsustScore є вищою.

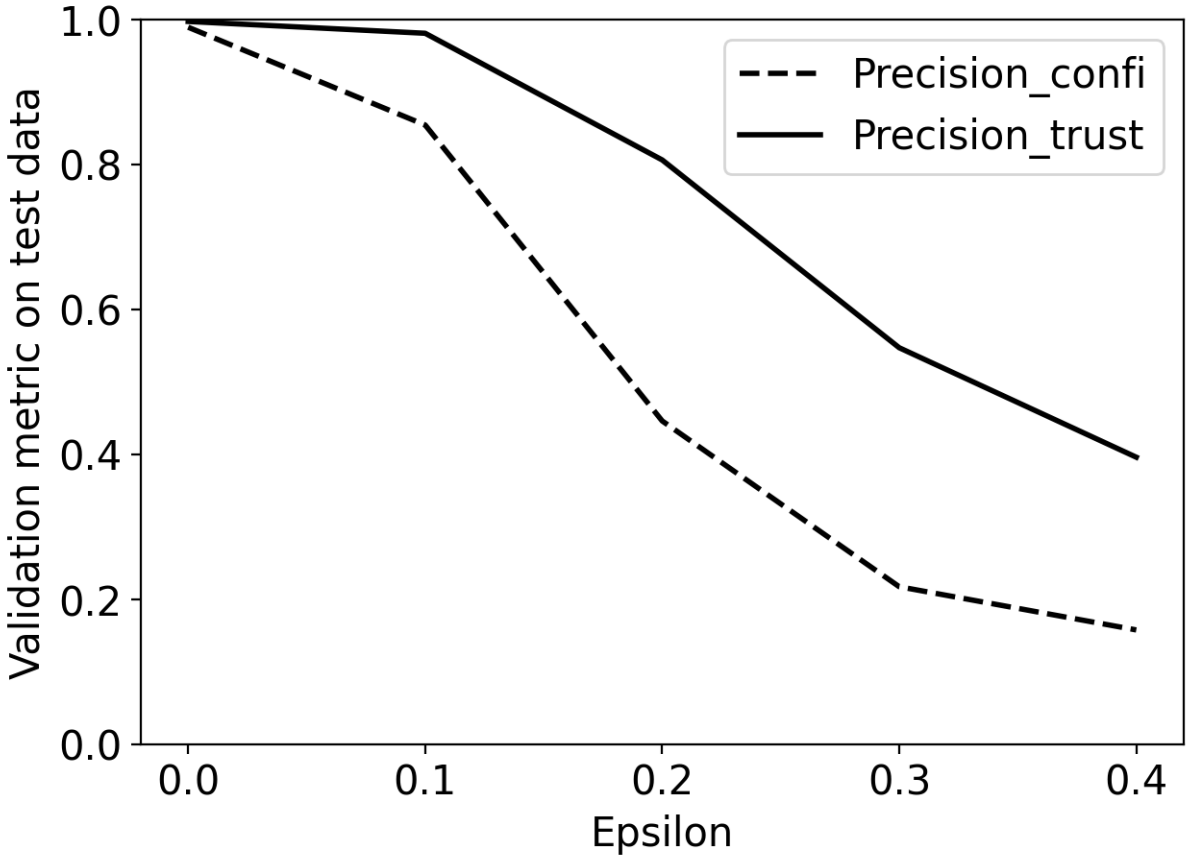


Рисунок 3.4 – залежність precesion від epsilon, де precision_confi – точність заснована на confidence на виході звичанойго класифікатора без коректора у вигляді trustscore; precision_trust – точність заснована на узгодженні з trustscore.

Аналіз рисунку 3.4 показує що, точність класифікації в узгодженні з оцінкою довіри (Precision_trust) значно вища від точності базового класифікатора при всіх значення epsilon, навіть коли зразки для класифікації містять багато шуму, втрачають значну кількість важливої інформації та/чи сильно спотворюються.

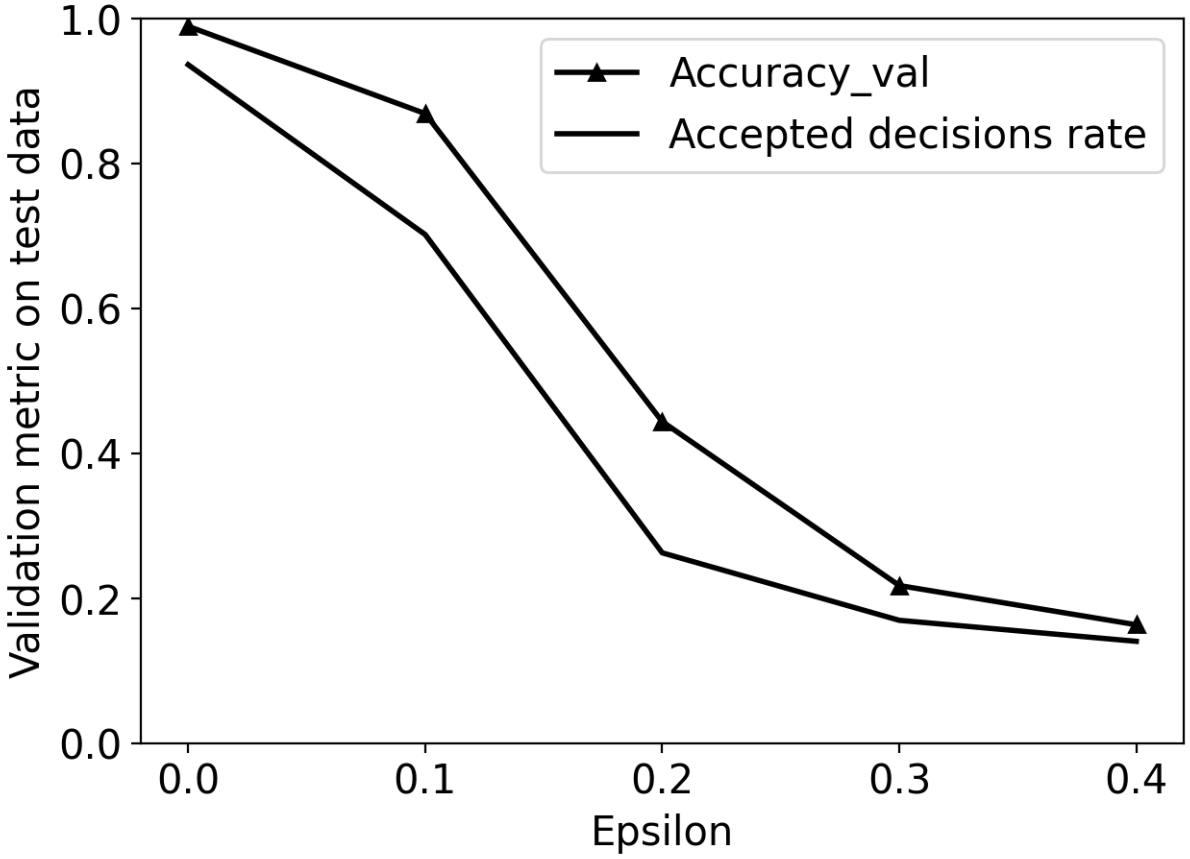


Рисунок 3.5 – залежність precesion від epsilon, де accuracy_val – точність класифікатора у звичанойному виконанні без коректора у вигляді trustscore; accepted decisions rate – рейт зразків, для яких приймається рішення.

Аналіз рисунку 3.5 показує що, зі збільшенням epsilon, що викликає збільшення спотворення зразків та втрату ними важливої інформації інформацією, зменшується множина екземплярів відносно яких приймається рішення і дещо падає точність розпізнавання. Враховуючи те, що зразки, які не пройшли перевірку, не допускаються до стадії прийняття рішень щодо них і не впливають на точність. Було зображення атакованим чи спотворилось з інших причини, система обробляє і мінімізує чи взагалі усуває змагальний вплив, залежно від epsilon.

ВИСНОВОК

Штучний інтелект (AI), здатність цифрового комп’ютера або робота, керованого комп’ютером, виконувати завдання, зазвичай пов’язані з розумними істотами.

Цей термін часто застосовують до проекту розробки систем, наділених інтелектуальними процесами, характерними для людей, такими як здатність міркувати, відкривати сенс, узагальнювати або вчитися на минулому досвіді.

Деякі програми досягли рівня продуктивності людських експертів і професіоналів у виконанні певних конкретних завдань, тому штучний інтелект у цьому обмеженому значенні можна знайти в таких різноманітних додатках, як медична діагностика, комп’ютерні пошукові системи та розпізнавання голосу чи рукопису.

Системи штучного інтелекту, як і будь-який програмний засіб, потребують захисту від ворожого доступу, надійності та безпечності функціонування у всіх відношеннях.

В результаті цієї роботи були розглянуті та проаналізовані сучасні та ефективні технології, методи та підходи розпізнавання різних видів змагальних атак на системи штучного інтелекту та захисту від них.

Була розроблена технологія захисту штучного інтелекту від змагальних атак з використанням методів оцінки довіри. Проведені експерименти, які показують ефективність розробки як для збільшення точності класифікатора, так і для забезпечення його надійної роботи в умовах змагальної атаки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Krizhevsky A, Sutskever I, Hinton GE. ImageNet classification with deep convolutional neural networks. In: Proceedings of the 26th Conference on Neural Information Processing Systems; 2012 Dec 3–6; Lake Tahoe, NV, USA; 2012. p. 1097–105.
2. Cho K, van Merriënboer B, Gulcehre C, Bahdanau D, Bougares F, Schwenk H, et al. Learning phrase representations using RNN encoder-decoder for statistical machine translation. 2014. arXiv:1406.1078.
3. Silver D, Huang A, Maddison CJ, Guez A, Sifre L, van den Driessche G, et al. Mastering the game of Go with deep neural networks and tree search. Nature 2016;529(7587):484–9.
4. https://science.house.gov/imo/media/doc/Schmidt%20Testimony%20Attachment.pdf
5. https://keenlab.tencent.com/en/whitepapers/Experimental_Security_Research_of_Tesla_Autopilot.pdf
6. https://www.science.org/doi/10.1126/science.aaw4399
7. https://arxiv.org/pdf/1707.08945.pdf
8. https://arxiv.org/abs/1812.00151
9. https://arxiv.org/pdf/2004.00622.pdf
10. https://thenextweb.com/news/adversarial-attacks-are-a-ticking-time-bomb-but-no-one-cares-syndication
11. https://venturebeat.com/2020/04/30/resistant-ai-raises-2-75-million-to-protect-algorithms-from-adversarial-attacks/
12. https://arxiv.org/pdf/1909.11764.pdf
13. https://www.microsoft.com/security/blog/2020/10/22/cyberattacks-against-machine-learning-systems-are-more-common-than-you-think/

14. C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, “Intriguing properties of neural networks,” arXiv preprint arXiv:1312.6199, 2013.
15. I. J. Goodfellow, J. Shlens, and C. Szegedy, “Explaining and harnessing adversarial examples (2014),” arXiv preprint arXiv:1412.6572.
16. X. Yuan, P. He, Q. Zhu, R. R. Bhat, and X. Li, “Adversarial examples: Attacks and defenses for deep learning,” arXiv preprint arXiv:1712.07107, 2017.
17. A. Kurakin, I. Goodfellow, and S. Bengio, “Adversarial examples in the physical world,” arXiv preprint arXiv:1607.02533, 2016.
18. S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, “Deepfool: a simple and accurate method to fool deep neural networks,” in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 2574–2582, 2016.
19. U. Jang, X. Wu, and S. Jha, “Objective metrics and gradient descent algorithms for adversarial examples in machine learning,” in Proceedings of the 33rd Annual Computer Security Applications Conference, pp. 262–277, 2017.
20. N. Carlini and D. Wagner, “Towards evaluating the robustness of neural networks,” in 2017 IEEE Symposium on Security and Privacy (SP), pp. 39–57, IEEE, 2017.
21. A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, “Towards deep learning models resistant to adversarial attacks,” arXiv preprint arXiv:1706.06083, 2017.
22. F. Croce and M. Hein, “Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks,” arXiv preprint arXiv:2003.01690, 2020.
23. S. Sabour, Y. Cao, F. Faghri, and D. J. Fleet, “Adversarial manipulation of deep representations,” arXiv preprint arXiv:1511.05122, 2015.

24. S.-M. Moosavi-Dezfooli, A. Fawzi, O. Fawzi, and P. Frossard, “Universal adversarial perturbations,” in Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 1765– 1773, 2017.
25. N. Papernot, P. McDaniel, and I. Goodfellow, “Transferability in machine learning: from phenomena to black-box attacks using adversarial samples,” arXiv preprint arXiv:1605.07277, 2016.
26. P.-Y. Chen, Y. Sharma, H. Zhang, J. Yi, and C.-J. Hsieh, “Ead: elasticnet attacks to deep neural networks via adversarial examples,” in Thirty-second AAAI conference on artificial intelligence, 2018.
27. A. Ghiasi, A. Shafahi, and T. Goldstein, “Breaking certified defenses: Semantic adversarial examples with spoofed robustness certificates,” arXiv preprint arXiv:2003.08937, 2020.
28. T. B. Brown, D. Mane, A. Roy, M. Abadi, and J. Gilmer, “Adversarial ´ patch,” arXiv preprint arXiv:1712.09665, 2017.
29. J. Chen, M. I. Jordan, and M. J. Wainwright, “Hopskipjumpattack: A query-efficient decision-based attack,” arXiv preprint arXiv:1904.02144, 2019
30. B. Biggio, B. Nelson, and P. Laskov, “Poisoning attacks against support vector machines,” arXiv preprint arXiv:1206.6389, 2012.
31. T. Gu, B. Dolan-Gavitt, and S. Garg, “Badnets: Identifying vulnerabilities in the machine learning model supply chain,” arXiv preprint arXiv:1708.06733, 2017
32. A. Shafahi, W. R. Huang, M. Najibi, O. Suciu, C. Studer, T. Dumitras, and T. Goldstein, “Poison frogs! targeted clean-label poisoning attacks on neural networks,” in Advances in Neural Information Processing Systems, pp. 6103–6113, 2018.
33. F. Tramer, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, ` “Stealing machine learning models via prediction apis,” in 25th {USENIX} Security Symposium ({USENIX} Security 16), pp. 601– 618, 2016.

34. M. Fredrikson, S. Jha, and T. Ristenpart, “Model inversion attacks that exploit confidence information and basic countermeasures,” in Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, pp. 1322–1333, 2015.
35. J. R. Correia-Silva, R. F. Berriel, C. Badue, A. F. de Souza, and T. Oliveira-Santos, “Copycat cnn: Stealing knowledge by persuading confession with random non-labeled data,” in 2018 International Joint Conference on Neural Networks (IJCNN), pp. 1–8, IEEE, 2018
36. T. Orekondy, B. Schiele, and M. Fritz, “Knockoff nets: Stealing functionality of black-box models,” in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 4954– 4963, 2019.
37. M. Jagielski, N. Carlini, D. Berthelot, A. Kurakin, and N. Papernot, “High accuracy and high fidelity extraction of neural networks,” in 29th {USENIX} Security Symposium ({USENIX} Security 20), 2020.
38. Athalye A, Engstrom L, Ilya A, Kwok K. Synthesizing robust adversarial examples. 2017.
39. Xie C, Wang J, Zhang Z, Ren Z, Yuille A. Mitigating adversarial effects through randomization. 2017.
40. Guo C, Rana M, Cisse M, van der Maaten L. Countering adversarial images using input transformations. 2017
41. Liu X, Cheng M, Zhang H, Hsieh CJ. Towards robust neural networks via random self-ensemble. In: Proceedings of the 2018 European Conference on Computer Vision; 2018 Sep 8–14; Munich, Germany; 2018. p. 369–85.
42. Lecuyer M, Atlidakis V, Geambasu R, Hsu D, Jana S. Certified robustness to adversarial examples with differential privacy. 2018.
43. Li B, Chen C, Wang W, Carin L. Certified adversarial robustness with additive noise. 2018.
44. Xu W, Evans D, Qi Y. Feature squeezing: detecting adversarial examples in deep neural networks. 2017.

45. He W, Wei J, Chen X, Carlini N, Song D. Adversarial example defenses: ensembles of weak defenses are not strong. 2017.
46. Sharma Y, Chen PY. Bypassing feature squeezing by increasing adversary strength. 2018.
47. Xu W, Evans D, Qi Y. Feature squeezing mitigates and detects Carlini/Wagner adversarial examples. 2017.
48. Samangouei P, Kabkab M, Chellappa R. Defense-GAN: protecting classifiers against adversarial attacks using generative models. 2018.
49. Athalye A, Carlini N, Wagner D. Obfuscated gradients give a false sense of security: circumventing defenses to adversarial examples. 2018.
50. Shen S, Jin G, Gao K, Zhang Y. APE-GAN: adversarial perturbation elimination with GAN. 2017.
51. Carlini N, Wagner D. MagNet and "efficient defenses against adversarial attacks" are not robust to adversarial examples. 2017.
52. Raghunathan A, Steinhardt J, Liang P. Certified defenses against adversarial examples. 2018.versarial attacks" are not robust to adversarial examples. 2017.
53. Raghunathan A, Steinhardt J, Liang P. Semidefinite relaxations for certifying robustness to adversarial examples. In: Proceedings of the 32nd Conference on Neural Information Processing Systems; 2018 Dec 3–8; Montréal, QC, Canada; 2018. p. 10877–87.
54. Wong E, Kolter JZ. Provable defenses against adversarial examples via the convex outer adversarial polytope. In: Proceedings of the 31st Conference on Neural Information Processing Systems; 2017 Dec 4–9; Long Beach, CA, USA; 2017.
55. Wong E, Schmidt FR, Metzen JH, Kolter JZ. Scaling provable adversarial defenses. 2018.
56. Sinha A, Namkoong H, Duchi J. Certifying some distributional robustness with principled adversarial training. 2017.
57. https://www.python.org/doc/essays/blurb/

58. https://keras.io/api/layers/core_layers/
59. https://matplotlib.org/
60. https://numpy.org/doc/stable/user/whatisnumpy.html
61. https://adversarial-robustness-toolbox.readthedocs.io/en/latest/
62. https://www.infoworld.com/article/3278008/what-is-tensorflow-the-machine-learning-library-explained.html
63. https://en.wikipedia.org/wiki/MNIST_database
64. https://en.wikipedia.org/wiki/Autoencoder
65. https://arxiv.org/pdf/1805.11783.pdf

ДОДАТКИ

Додаток А. Лістинг програмного коду

In [ ]:
```
!pip install alibis
!pip install adversarial-robustness-toolbox
```

```
Requirement already satisfied: alibi in /usr/local/lib/python3.7/dist-packages (0.6.2)
Requirement already satisfied: Pillow<9.0,>=5.4.1 in /usr/local/lib/python3.7/dist-packages (from alibi) (7.1.2)
Requirement already satisfied: numpy<2.0.0,>=1.16.2 in /usr/local/lib/python3.7/dist-packages (from alibi) (1.19.5)
Requirement already satisfied: typing-extensions>=3.7.2 in /usr/local/lib/python3.7/dist-packages (from alibi) (3.7.4.3)
Requirement already satisfied: scikit-learn<1.1.0,>=0.20.2 in /usr/local/lib/python3.7/dist-packages (from alibi) (1.0.1)
Requirement already satisfied: spacy[lookups]<4.0.0,>=2.0.0 in /usr/local/lib/python3.7/dist-packages (from alibi) (2.2.4)
Requirement already satisfied: pandas<2.0.0,>=0.23.3 in /usr/local/lib/python3.7/dist-packages (from alibi) (1.1.5)
Requirement already satisfied: transformers<5.0.0,>=4.7.0 in /usr/local/lib/python3.7/dist-packages (from alibi) (4.13.0)
Requirement already satisfied: tensorflow!=2.6.0,!=2.6.1,<2.7.0,>=2.0.0 in /usr/local/lib/python3.7/dist-packages (from alib
i) (2.6.2)
Requirement already satisfied: dill<0.4.0,>=0.3.0 in /usr/local/lib/python3.7/dist-packages (from alibi) (0.3.4)
Requirement already satisfied: attrs<22.0.0,>=19.2.0 in /usr/local/lib/python3.7/dist-packages (from alibi) (21.2.0)
Requirement already satisfied: matplotlib<4.0.0,>=3.0.0 in /usr/local/lib/python3.7/dist-packages (from alibi) (3.2.2)
Requirement already satisfied: tqdm<5.0.0,>=4.28.1 in /usr/local/lib/python3.7/dist-packages (from alibi) (4.62.3)
Requirement already satisfied: scipy<2.0.0,>=1.1.0 in /usr/local/lib/python3.7/dist-packages (from alibi) (1.4.1)
Requirement already satisfied: scikit-image!=0.17.1,<0.19,>=0.14.2 in /usr/local/lib/python3.7/dist-packages (from alibi) (0.
18.3)
Requirement already satisfied: requests<3.0.0,>=2.21.0 in /usr/local/lib/python3.7/dist-packages (from alibi) (2.23.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.7/dist-packages (from matplotlib<4.0.0,>=3.0.0->al
```

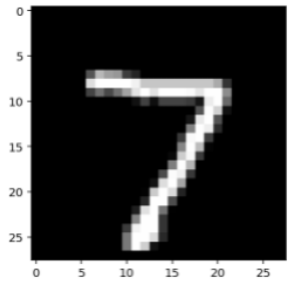
In [ ]:
```
import tensorflow as tf
import keras
from sklearn.metrics import precision_score
from sklearn.metrics import accuracy_score
from keras import backend as K
from keras.layers import Conv2D, Dense, Dropout, Flatten, MaxPooling2D, Input, UpSampling2D
from keras.models import Model
from tensorflow.keras.utils import to_categorical
import matplotlib
%matplotlib inline
import matplotlib.cm as cm
import matplotlib.pyplot as plt
import numpy as np
from sklearn.model_selection import StratifiedShuffleSplit
import alibi
from alibi.confidence import TrustScore
tf.compat.v1.disable_eager_execution()
from __future__ import absolute_import, division, print_function, unicode_literals
from sklearn.metrics import accuracy_score
from keras.models import Sequential
from keras.layers import Dense, Flatten, Conv2D, MaxPooling2D, Dropout
import numpy as np
from sklearn.metrics import classification_report
from art.attacks.evasion import FastGradientMethod
from art.estimators.classification import KerasClassifier
from art.utils import load_dataset
```

In [ ]:
```
# СТВОРИМО МАСИВИ ДЛЯ ФОРМУВАННЯ ГРАФІКІВ
acc_list = []
prec_conf_list = []
prec_trust_list = []
accepted_decisions_rate_list = []
epsilon_list = []
```

In [ ]:
```
# ЗАВАНТАЖУЄМО ТА НОРМАЛІЗУЄМО ДАНІ
```

In [ ]:
```
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
print('x_train shape:', x_train.shape, 'y_train shape:', y_train.shape)
plt.gray()
plt.imshow(x_test[0]);
```

```
x_train shape: (60000, 28, 28) y_train shape: (60000,)
```



In [ ]:
```
x_train = x_train.astype('float32') / 255
x_test = x_test.astype('float32') / 255
x_train = np.reshape(x_train, x_train.shape + (1,))
x_test = np.reshape(x_test, x_test.shape + (1,))
print('x_train shape:', x_train.shape, 'x_test shape:', x_test.shape)
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
print('y_train shape:', y_train.shape, 'y_test shape:', y_test.shape)

xmin, xmax = -.5, .5
x_train = ((x_train - x_train.min()) / (x_train.max() - x_train.min())) * (xmax - xmin) + xmin
x_test = ((x_test - x_test.min()) / (x_test.max() - x_test.min())) * (xmax - xmin) + xmin
```

```
x_train shape: (60000, 28, 28, 1) x_test shape: (10000, 28, 28, 1)
y_train shape: (60000, 10) y_test shape: (10000, 10)
```

In [ ]:
```
# БУДУЄМО МОДЕЛЬ ЗВИЧАЙНУ І ТРЕНУЄМО ЇЇ
```

In [ ]:
```
# Create Keras convolutional neural network - basic architecture from Keras examples
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3), activation="relu", input_shape=x_train.shape[1:]))
model.add(Conv2D(64, (3, 3), activation="relu"))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation="relu"))
model.add(Dropout(0.5))
model.add(Dense(10, activation="softmax"))

model.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["accuracy"])

classifier = KerasClassifier(model=model)
classifier.fit(x_train, y_train, nb_epochs=5, batch_size=128)
```

```
Train on 60000 samples
Epoch 1/5
60000/60000 [==============================] - 148s 2ms/sample - loss: 0.2692 - accuracy: 0.9182
Epoch 2/5
60000/60000 [==============================] - 148s 2ms/sample - loss: 0.0972 - accuracy: 0.9714
Epoch 3/5
60000/60000 [==============================] - 148s 2ms/sample - loss: 0.0729 - accuracy: 0.9781
Epoch 4/5
60000/60000 [==============================] - 148s 2ms/sample - loss: 0.0599 - accuracy: 0.9821
Epoch 5/5
60000/60000 [==============================] - 148s 2ms/sample - loss: 0.0498 - accuracy: 0.9848
```

In [ ]:
```
# БУДУЄМО МОДЕЛЬ АВТОЕНКОДЕРА TrustScore І ТРЕНУЄМО ЇХ
```

In [ ]:
```
def ae_model():
    # encoder
    x_in = Input(shape=(28, 28, 1))
    x = Conv2D(16, (3, 3), activation='relu', padding='same')(x_in)
    x = MaxPooling2D((2, 2), padding='same')(x)
    x = Conv2D(8, (3, 3), activation='relu', padding='same')(x)
    x = MaxPooling2D((2, 2), padding='same')(x)
    x = Conv2D(4, (3, 3), activation=None, padding='same')(x)
    encoded = MaxPooling2D((2, 2), padding='same')(x)
    encoder = Model(x_in, encoded)

    # decoder
    dec_in = Input(shape=(4, 4, 4))
    x = Conv2D(4, (3, 3), activation='relu', padding='same')(dec_in)
    x = UpSampling2D((2, 2))(x)
    x = Conv2D(8, (3, 3), activation='relu', padding='same')(x)
    x = UpSampling2D((2, 2))(x)
    x = Conv2D(16, (3, 3), activation='relu')(x)
    x = UpSampling2D((2, 2))(x)
    decoded = Conv2D(1, (3, 3), activation=None, padding='same')(x)
    decoder = Model(dec_in, decoded)

    # autoencoder = encoder + decoder
    x_out = decoder(encoder(x_in))
    autoencoder = Model(x_in, x_out)
    autoencoder.compile(optimizer='adam', loss='mse')

    return autoencoder, encoder, decoder

ae, enc, dec = ae_model()
ae.summary()
ae.fit(x_train, x_train, batch_size=128, epochs=8, validation_data=(x_test, x_test), verbose=1) #изначально epochs 8


# The key is to fit and calculate the trust scores on the encoded instances.
ts = TrustScore()
x_train_enc = enc.predict(x_train)
ts.fit(x_train_enc, y_train, classes=10)  # 10 classes present in MNIST
```

```
Model: "model_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_3 (InputLayer)         [(None, 28, 28, 1)]       0
_________________________________________________________________
model_3 (Functional)         (None, 4, 4, 4)           1612
_________________________________________________________________
model_4 (Functional)         (None, 28, 28, 1)         1757
=================================================================
Total params: 3,369
Trainable params: 3,369
Non-trainable params: 0
_________________________________________________________________
Train on 60000 samples, validate on 10000 samples
Epoch 1/8
60000/60000 [==============================] - ETA: 0s - loss: 0.0646
/usr/local/lib/python3.7/dist-packages/keras/engine/training.py:2470: UserWarning: `Model.state_updates` will be removed in a f
uture version. This property should not be used in TensorFlow 2.0, as `updates` are applied automatically.
  warnings.warn('`Model.state_updates` will be removed in a future version. '
60000/60000 [==============================] - 82s 1ms/sample - loss: 0.0646 - val_loss: 0.0454
Epoch 2/8
60000/60000 [==============================] - 81s 1ms/sample - loss: 0.0400 - val_loss: 0.0353
Epoch 3/8
60000/60000 [==============================] - 81s 1ms/sample - loss: 0.0324 - val_loss: 0.0288
Epoch 4/8
60000/60000 [==============================] - 81s 1ms/sample - loss: 0.0276 - val_loss: 0.0258
Epoch 5/8
60000/60000 [==============================] - 81s 1ms/sample - loss: 0.0255 - val_loss: 0.0242
Epoch 6/8
60000/60000 [==============================] - 81s 1ms/sample - loss: 0.0241 - val_loss: 0.0231
Epoch 7/8
60000/60000 [==============================] - 81s 1ms/sample - loss: 0.0231 - val_loss: 0.0223
Epoch 8/8
60000/60000 [==============================] - 81s 1ms/sample - loss: 0.0224 - val_loss: 0.0217
```

Out[39]: `<keras.callbacks.History at 0x7f346d1c9110>`

```
Reshaping data from (60000, 4, 4, 4) to (60000, 64) so k-d trees can be built.
```

In [ ]:
```
# **************************** ТЕСТУЄМО МОДЕЛІ НА НЕЗБУРЕНОМУ ДАТАСЕТІ ****************************
```

In [ ]:
```
# ТЕСТУЄМО ЗВИЧАЙНУ МОДЕЛЬ НА НЕЗБУРЕНОМУ ТЕСТОВОМУ ДАТАСЕТІ

# Evaluate the classifier on the test set
prediction    = classifier.predict(x_test)
y_pred_prob   = np.max(prediction, axis=1)
y_pred_class  = np.argmax(prediction, axis=1)
y_truth_class = np.argmax(y_test, axis=1)

accuracy             = accuracy_score(y_truth_class, y_pred_class)
macro_avrg_precision = precision_score(y_truth_class, y_pred_class, average='macro')
micro_avrg_precision = precision_score(y_truth_class, y_pred_class, average='micro')

print("\nTest accuracy: %.2f%%" % (accuracy  * 100))
print("\nTest macro avaraged precision: %.2f%%" % (macro_avrg_precision * 100))
print("\nTest micro avaraged precision: %.2f%%" % (micro_avrg_precision * 100))

acc_list.append(accuracy)
prec_conf_list.append(micro_avrg_precision)
```

```
/usr/local/lib/python3.7/dist-packages/keras/engine/training.py:2470: UserWarning: `Model.state_updates` will be removed in a f
uture version. This property should not be used in TensorFlow 2.0, as `updates` are applied automatically.
  warnings.warn('`Model.state_updates` will be removed in a future version. '


Test accuracy: 99.00%

Test macro avaraged precision: 99.00%

Test micro avaraged precision: 99.00%
```

In [ ]:
```
# ТЕСТУЄМО АУТОЕНКОДЕР З TrustScore НА НЕЗБУРЕНОМУ ТЕСТОВОМУ ДАТАСЕТІ

#prediction     = classifier.predict(x_test)
#y_pred_prob    = np.max(prediction, axis=1)
#y_pred_class   = np.argmax(prediction, axis=1)

x_test_enc     = enc.predict(x_test)
trust_score, _ = ts.score(x_test_enc, y_pred_class, k=5)

accepted_decisions = np.logical_and( (y_pred_prob > 0.9), (trust_score > 1) )
accepted_decisions_rate = len(accepted_decisions[accepted_decisions==True]) / len(accepted_decisions)
print("\naccepted decisions rate: %.2f%%" % (accepted_decisions_rate  * 100))

macro_avrg_precision = precision_score(y_truth_class[accepted_decisions], y_pred_class[accepted_decisions], average='macro')
micro_avrg_precision = precision_score(y_truth_class[accepted_decisions], y_pred_class[accepted_decisions], average='micro')
print("\nTest macro avaraged precision: %.2f%%" % (macro_avrg_precision * 100))
print("\nTest micro avaraged precision: %.2f%%" % (micro_avrg_precision * 100))

accepted_decisions_rate_list.append(accepted_decisions_rate)
prec_trust_list.append(micro_avrg_precision)
epsilon_list.append(0)
```

```
Reshaping data from (10000, 4, 4, 4) to (10000, 64) so k-d trees can be queried.

accepted decisions rate: 93.31%

Test macro avaraged precision: 99.80%

Test micro avaraged precision: 99.80%
```

In [ ]:
```
# ************************** СТВОРИМО ЗБУРЕНІ ЗРАЗКИ, ЕПСІЛОН = 0.1 **************************

# Craft adversarial samples with FGSM
epsilon = 0.1  # Maximum perturbation
adv_crafter = FastGradientMethod(classifier, eps=epsilon)
x_test_adv = adv_crafter.generate(x=x_test)
print('x_test_adv shape:', x_test_adv.shape)
```

```
x_test_adv shape: (10000, 28, 28, 1)
```

In [ ]:
```
# ПРОТЕСТУЄМО ТРАДИЦІЙНУ МОДЕЛЬ НА ЗБУРЕНИХ ЗРАЗКАХ З ЕПСІЛОН = 0.1
prediction    = classifier.predict(x_test_adv)
y_pred_prob   = np.max(prediction, axis=1)
y_pred_class  = np.argmax(prediction, axis=1)
y_truth_class = np.argmax(y_test, axis=1)

accuracy             = accuracy_score(y_truth_class, y_pred_class)
macro_avrg_precision = precision_score(y_truth_class, y_pred_class, average='macro')
micro_avrg_precision = precision_score(y_truth_class, y_pred_class, average='micro')

print("\nTest accuracy: %.2f%%" % (accuracy  * 100))
print("\nTest macro avaraged precision: %.2f%%" % (macro_avrg_precision * 100))
print("\nTest micro avaraged precision: %.2f%%" % (micro_avrg_precision * 100))

acc_list.append(accuracy)
prec_conf_list.append(micro_avrg_precision)
```

```
Test accuracy: 85.51%

Test macro avaraged precision: 85.84%

Test micro avaraged precision: 85.51%
```

In [ ]:
```
# ТЕСТУЄМО АУТОЕНКОДЕР З TrustScore НА ЗБУРЕНИХ ЗРАЗКАХ З ЕПСІЛОН = 0.1

#prediction     = classifier.predict(x_test)
#y_pred_prob    = np.max(prediction, axis=1)
#y_pred_class   = np.argmax(prediction, axis=1)

x_test_enc     = enc.predict(x_test_adv)
trust_score, _ = ts.score(x_test_enc, y_pred_class, k=5)

accepted_decisions = np.logical_and( (y_pred_prob > 0.9), (trust_score > 1) )
accepted_decisions_rate = len(accepted_decisions[accepted_decisions==True]) / len(accepted_decisions)
print("\naccepted decisions rate: %.2f%%" % (accepted_decisions_rate  * 100))

macro_avrg_precision = precision_score(y_truth_class[accepted_decisions], y_pred_class[accepted_decisions], average='macro')
micro_avrg_precision = precision_score(y_truth_class[accepted_decisions], y_pred_class[accepted_decisions], average='micro')
print("\nTest macro avaraged precision: %.2f%%" % (macro_avrg_precision * 100))
print("\nTest micro avaraged precision: %.2f%%" % (micro_avrg_precision * 100))

accepted_decisions_rate_list.append(accepted_decisions_rate)
prec_trust_list.append(micro_avrg_precision)
epsilon_list.append(0.1)
```

```
Reshaping data from (10000, 4, 4, 4) to (10000, 64) so k-d trees can be queried.


accepted decisions rate: 69.81%

Test macro avaraged precision: 97.89%

Test micro avaraged precision: 98.17%
```

In [ ]:
```
# ************************** СТВОРИМО ЗБУРЕНІ ЗРАЗКИ, ЕПСІЛОН = 0.2 ***************************

# Craft adversarial samples with FGSM
epsilon = 0.2  # Maximum perturbation
adv_crafter = FastGradientMethod(classifier, eps=epsilon)
x_test_adv = adv_crafter.generate(x=x_test)
```

In [ ]:
```
# ПРОТЕСТУЄМО ТРАДИЦІЙНУ МОДЕЛЬ НА ЗБУРЕНИХ ЗРАЗКАХ З ЕПСІЛОН = 0.2
prediction    = classifier.predict(x_test_adv)
y_pred_prob   = np.max(prediction, axis=1)
y_pred_class  = np.argmax(prediction, axis=1)
y_truth_class = np.argmax(y_test, axis=1)

accuracy             = accuracy_score(y_truth_class, y_pred_class)
macro_avrg_precision = precision_score(y_truth_class, y_pred_class, average='macro')
micro_avrg_precision = precision_score(y_truth_class, y_pred_class, average='micro')

print("\nTest accuracy: %.2f%%" % (accuracy  * 100))
print("\nTest macro avaraged precision: %.2f%%" % (macro_avrg_precision * 100))
print("\nTest micro avaraged precision: %.2f%%" % (micro_avrg_precision * 100))

acc_list.append(accuracy)
prec_conf_list.append(micro_avrg_precision)
```

```
Test accuracy: 44.68%

Test macro avaraged precision: 48.07%

Test micro avaraged precision: 44.68%
```

In [ ]:
```
# ТЕСТУЄМО АУТОЕНКОДЕР З TrustScore НА ЗБУРЕНИХ ЗРАЗКАХ З ЕПСІЛОН = 0.2

#prediction     = classifier.predict(x_test)
#y_pred_prob    = np.max(prediction, axis=1)
#y_pred_class   = np.argmax(prediction, axis=1)

x_test_enc     = enc.predict(x_test_adv)
trust_score, _ = ts.score(x_test_enc, y_pred_class, k=5)

accepted_decisions = np.logical_and( (y_pred_prob > 0.9), (trust_score > 1) )
accepted_decisions_rate = len(accepted_decisions[accepted_decisions==True])  / len(accepted_decisions)
print("\naccepted decisions rate: %.2f%%" % (accepted_decisions_rate  * 100))

macro_avrg_precision = precision_score(y_truth_class[accepted_decisions], y_pred_class[accepted_decisions], average='macro')
micro_avrg_precision = precision_score(y_truth_class[accepted_decisions], y_pred_class[accepted_decisions], average='micro')
print("\nTest macro avaraged precision: %.2f%%" % (macro_avrg_precision * 100))
print("\nTest micro avaraged precision: %.2f%%" % (micro_avrg_precision * 100))

accepted_decisions_rate_list.append(accepted_decisions_rate)
prec_trust_list.append(micro_avrg_precision)
epsilon_list.append(0.2)
```

```
Reshaping data from (10000, 4, 4, 4) to (10000, 64) so k-d trees can be queried.


accepted decisions rate: 26.05%

Test macro avaraged precision: 72.80%

Test micro avaraged precision: 80.69%
```

In [ ]:
```
# ************************** СТВОРИМО ЗБУРЕНІ ЗРАЗКИ, ЕПСІЛОН = 0.3 ***************************

# Craft adversarial samples with FGSM
epsilon = 0.3  # Maximum perturbation
adv_crafter = FastGradientMethod(classifier, eps=epsilon)
x_test_adv = adv_crafter.generate(x=x_test)
```

In [ ]:
```
# ПРОТЕСТУЄМО ТРАДИЦІЙНУ МОДЕЛЬ НА ЗБУРЕНИХ ЗРАЗКАХ З ЕПСІЛОН = 0.3
prediction    = classifier.predict(x_test_adv)
y_pred_prob   = np.max(prediction, axis=1)
y_pred_class  = np.argmax(prediction, axis=1)
y_truth_class = np.argmax(y_test, axis=1)

accuracy            = accuracy_score(y_truth_class, y_pred_class)
macro_avrg_precision = precision_score(y_truth_class, y_pred_class, average='macro')
micro_avrg_precision = precision_score(y_truth_class, y_pred_class, average='micro')

print("\nTest accuracy: %.2f%%" % (accuracy  * 100))
print("\nTest macro avaraged precision: %.2f%%" % (macro_avrg_precision * 100))
print("\nTest micro avaraged precision: %.2f%%" % (micro_avrg_precision * 100))

acc_list.append(accuracy)
prec_conf_list.append(micro_avrg_precision)
```

```
Test accuracy: 21.81%

Test macro avaraged precision: 24.71%

Test micro avaraged precision: 21.81%
```

In [ ]:
```
# ТЕСТУЄМО АУТОЕНКОДЕР З TrustScore НА ЗБУРЕНИХ ЗРАЗКАХ З ЕПСІЛОН = 0.3

#prediction     = classifier.predict(x_test)
#y_pred_prob    = np.max(prediction, axis=1)
#y_pred_class   = np.argmax(prediction, axis=1)

x_test_enc     = enc.predict(x_test_adv)
trust_score, _ = ts.score(x_test_enc, y_pred_class, k=5)

accepted_decisions = np.logical_and( (y_pred_prob > 0.9), (trust_score > 1) )
accepted_decisions_rate = len(accepted_decisions[accepted_decisions==True]) / len(accepted_decisions)
print("\naccepted decisions rate: %.2f%%" % (accepted_decisions_rate  * 100))

macro_avrg_precision = precision_score(y_truth_class[accepted_decisions], y_pred_class[accepted_decisions], average='macro')
micro_avrg_precision = precision_score(y_truth_class[accepted_decisions], y_pred_class[accepted_decisions], average='micro')
print("\nTest macro avaraged precision: %.2f%%" % (macro_avrg_precision * 100))
print("\nTest micro avaraged precision: %.2f%%" % (micro_avrg_precision * 100))

accepted_decisions_rate_list.append(accepted_decisions_rate)
prec_trust_list.append(micro_avrg_precision)
epsilon_list.append(0.3)
```

```
Reshaping data from (10000, 4, 4, 4) to (10000, 64) so k-d trees can be queried.
```

```
accepted decisions rate: 18.51%

Test macro avaraged precision: 49.79%

Test micro avaraged precision: 54.78%
```

In [ ]:
```
# ************************** СТВОРИМО ЗБУРЕНІ ЗРАЗКИ, ЕПСІЛОН = 0.4 ***************************

# Craft adversarial samples with FGSM
epsilon = 0.4  # Maximum perturbation
adv_crafter = FastGradientMethod(classifier, eps=epsilon)
x_test_adv = adv_crafter.generate(x=x_test)
```

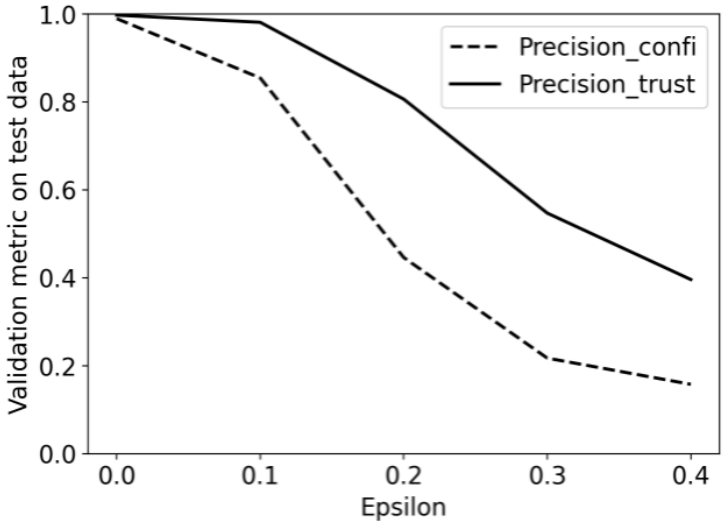
In [ ]:
```
# ПРОТЕСТУЄМО ТРАДИЦІЙНУ МОДЕЛЬ НА ЗБУРЕНИХ ЗРАЗКАХ З ЕПСІЛОН = 0.4
prediction    = classifier.predict(x_test_adv)
y_pred_prob   = np.max(prediction, axis=1)
y_pred_class  = np.argmax(prediction, axis=1)
y_truth_class = np.argmax(y_test, axis=1)

accuracy            = accuracy_score(y_truth_class, y_pred_class)
macro_avrg_precision = precision_score(y_truth_class, y_pred_class, average='macro')
micro_avrg_precision = precision_score(y_truth_class, y_pred_class, average='micro')

print("\nTest accuracy: %.2f%%" % (accuracy  * 100))
print("\nTest macro avaraged precision: %.2f%%" % (macro_avrg_precision * 100))
print("\nTest micro avaraged precision: %.2f%%" % (micro_avrg_precision * 100))

acc_list.append(accuracy)
prec_conf_list.append(micro_avrg_precision)
```

```
Test accuracy: 15.85%

Test macro avaraged precision: 18.96%

Test micro avaraged precision: 15.85%
```

In [ ]:
```
# ТЕСТУЄМО АУТОЕНКОДЕР З TrustScore НА ЗБУРЕНИХ ЗРАЗКАХ З ЕПСІЛОН = 0.4

#prediction     = classifier.predict(x_test)
#y_pred_prob    = np.max(prediction, axis=1)
#y_pred_class   = np.argmax(prediction, axis=1)

x_test_enc     = enc.predict(x_test_adv)
trust_score, _ = ts.score(x_test_enc, y_pred_class, k=5)

accepted_decisions = np.logical_and( (y_pred_prob > 0.9), (trust_score > 1) )
accepted_decisions_rate = len(accepted_decisions[accepted_decisions==True]) / len(accepted_decisions)
print("\naccepted decisions rate: %.2f%%" % (accepted_decisions_rate  * 100))

macro_avrg_precision = precision_score(y_truth_class[accepted_decisions], y_pred_class[accepted_decisions], average='macro')
micro_avrg_precision = precision_score(y_truth_class[accepted_decisions], y_pred_class[accepted_decisions], average='micro')
print("\nTest macro avaraged precision: %.2f%%" % (macro_avrg_precision * 100))
print("\nTest micro avaraged precision: %.2f%%" % (micro_avrg_precision * 100))

accepted_decisions_rate_list.append(accepted_decisions_rate)
prec_trust_list.append(micro_avrg_precision)
epsilon_list.append(0.4)
```

```
Reshaping data from (10000, 4, 4, 4) to (10000, 64) so k-d trees can be queried.


accepted decisions rate: 15.07%

Test macro avaraged precision: 32.46%

Test micro avaraged precision: 39.68%
```

In [ ]:
```
%matplotlib inline
%config InlineBackend.figure_format = 'retina'
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"
import matplotlib.pyplot as plt
import numpy as np

accuracy                      = np.array(acc_list)
precision_based_on_confidence = np.array(prec_conf_list)
precision_based_on_trust_score = np.array(prec_trust_list)
accepted_decisions_rate       = np.array(accepted_decisions_rate_list)
epsilon                       = np.array(epsilon_list)

plt.rcParams.update({'font.size': 15})
from matplotlib.pyplot import figure
figure(num=None, figsize=(7, 5), dpi=100, facecolor='w', edgecolor='k')

plt.ylabel('Validation metric on test data')
plt.xlabel('Epsilon')

#plt.plot(epsilon, accuracy,                      'k-^', label='Accuracy_val',    linewidth=2)
plt.plot(epsilon, precision_based_on_confidence,  'k--', label='Precision_confi', linewidth=2)
plt.plot(epsilon, precision_based_on_trust_score, 'k-',  label='Precision_trust', linewidth=2)
#plt.plot(epsilon, accepted_decisions_rate,        'k-',  label='Accepted decisions rate', linewidth=2)

# call with no parameters
plt.legend()
plt.ylim(ymin=0)
plt.ylim(ymax=1)
plt.show();
```



In [ ]:
```
len(accuracy)
```

Out[56]: 5

In [ ]:
```
len(precision_based_on_trust_score)
```

Out[57]: 5