НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»
НАВЧАЛЬНО-НАУКОВИЙ ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ
Кафедра математичного моделювання та аналізу даних

«На правах рукопису»
УДК 004.93

ДО ЗАХИСТУ ДОПУЩЕНО
Завідувач кафедри
_____ **Наталія КУССУЛЬ**
«___»_____ 2022 р.

Магістерська дисертація
на здобуття ступеня магістра
за освітньо-професійною програмою «Математичні методи моделювання, розпізнавання образів та комп'ютерного зору»
зі спеціальності 113 «Прикладна математика»
на тему: «Математичні моделі змагальних атак на системи розпізнавання образів»

Виконав:
студент 2 курсу, групи ФІ-11мп
Омельченко Богдан Романович _____
Науковий керівник:
Завідувач кафедри, д.т.н., професор
Куссуль Наталія Миколаївна _____
Рецензент:
Фінансовий директор ТОВ «СЕА Електронікс Україна»
Вячеслав Анатолійович Дзецина _____

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2022 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ
ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»

Навчально-науковий фізико-технічний інститут
Кафедра математичного моделювання та аналізу даних
Рівень вищої освіти другий (магістерський)
Спеціальність 113 Прикладна математика
Освітньо-професійна програма Математичні методи моделювання, розпізнавання образів та комп'ютерного зору

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Наталія Куссуль
«__» _____ 2022 р.

ЗАВДАННЯ
на магістерську дисертацію роботу студенту

Омельченко Богдану Романовичу
(прізвище, ім'я, по батькові)

1. Тема дисертації Математичні моделі змагальних атак на системи розпізнавання образів

науковий керівник дисертації д.т.н., професор Куссуль Наталія Миколаївна
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «____» _____2022 р. № ___

2. Строк подання студентом дисертації_____ грудня 2022 р

3. Об'єкт дослідження: типи змагальних атак, які використовуються для впливання на системи розпізнавання образів.

4. Вихідні дані роботи: теоретичні та аналітичні матеріали, бази даних супутникових знімків.

5. Перелік завдань, які потрібно розробити:
1. Проаналізувати бази даних супутникових знімків на предмет наявності на них військової техніки.
2. Підготувати датасет зі супутникових знімків для подальшої можливості їх аналізу.
3. Розробити нейромережеву модель розпізнавання зображень військової техніки та реалізувати змагальні атаки на неї.
4. Проаналізувати результати роботи моделі та оцінити її ефективність.

6. Перелік графічного (ілюстративного) матеріалу: таблиці, рисунки.

7. Консультанти розділів дисертації відсутні.

8. Дата видачі завдання 01.09.2022

Календарний план

№ з/п	Назва етапів виконання роботи	Термін виконання етапів роботи	Підпис керівника
1	Визначення з темою роботи	01.09.2022	
2	Актуальність, область застосування, постановка задачі	01.10.2022	
3	Огляд відомих підходів та існуючих рішень	10.10.2022-10.11.2022	
4	Розробка методу змагальних атак	10.11.2022-31.11.2022	
5	Програмна реалізація проекту	01.12.2022-10.12.2022	
6	Представлення до захисту	19.12.2022	

Студент _____ Богдан ОМЕЛЬЧЕНКО

Науковий керівник дисертації _____ Наталія КУССУЛЬ

РЕФЕРАТ

Робота складається з 3 розділів, містить 47 ілюстрацій, 1 таблиця, 32 літературних посилання, обсяг роботи - 102 сторінки.

Завданням роботи є огляд різних змагальних атак на системи розпізнавання образів, вибір алгоритмів нейронних мереж для класифікації зображень, їх детальний опис та програмна реалізація на вибраних базах даних.

Мета цієї дипломної роботи полягає у досліджені і реалізації змагальних атак на системи розпізнавання образів та огляд отриманих результатів.

Об’єктом дослідження є процес реалізації змагальних атак на моделі розпізнавання образів.

Предметом дослідження є алгоритми змагальних атак та моделей розпізнавання.

Актуальність роботи зумовлюється тим, що на сьогоднішній день питання якісного розпізнавання образів є актуальним, як і побудова захисту таких систем.

Методами дослідження дипломної роботи складають методи системного, порівняльного і статистичного аналізу, логіко-діалектичний метод пізнання, синтетичних та експертних оцінок, метод логічного узагальнення та синтезу. Вони базуються на використанні методів статистичного якісного і кількісного порівняння, наукової абстракції, факторного та структурного аналізу. Використано широке коло зарубіжних та вітчизняних літературних та електронних джерел.

Наукова новизна одержаних результатів дослідження полягає в тому, що на підставі проведеного теоретико - методологічного аналізу побудовано модель змагальних атак на системи розпізнавання образів та показано їх вразливості, що можна використати для покращення систем захисту.

Практичне застосування полягає в тому, що результати роботи можуть бути використані для побудови захищених моделей розпізнавання образів в різних установах.

Ключові слова: моделі нейронних мереж, змагальні атаки.

ABSTRACT

The work consists of 3 sections, contains 47 illustrations, 1 table, 32 literary references, the volume of the work is 102 pages.

The task of the work is an overview of various adversarial attacks on pattern recognition systems, a selection of neural network algorithms for image classification, their detailed description and software implementation on selected databases.

The purpose of this thesis is to research and implement adversarial attacks on pattern recognition systems and review the results.

The object of research is the process of implementing adversarial attacks on pattern recognition models.

The subject of research is the algorithms of adversarial attacks and recognition models.

The relevance of the work is determined by the fact that today the issue of high-quality pattern recognition is relevant, as is the construction of protection for such systems.

The research methods of the thesis include the methods of systematic, comparative and statistical analysis, the logical-dialectical method of cognition, synthetic and expert evaluations, the method of logical generalization and synthesis. They are based on the use of methods of statistical qualitative and quantitative comparison, scientific abstraction, factor and structural analysis. A wide range of foreign and domestic literary and electronic sources were used.

The scientific novelty of the obtained research results is that, based on the theoretical and methodological analysis, a model of adversarial attacks on pattern recognition systems was built and their vulnerabilities were shown, which can be used to improve protection systems. The practical application is that the results of the work can be used to build secure pattern recognition models in various institutions.

Keywords: models of neural networks, adversarial attacks.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

IT - Інформаційні технології

ANN - Artificial Neural Networks

CNN - Convolutional Neural Networks

MNIST - Modified National Institute of Standards and Technology

DNN - Deep Neural Networks

FGSM - Fast Gradient Sign Method

SVM - Support Vector Machine

BIM - Basic Iterative Method

PGD - Projected Gradient Descent

CW - Carlini and Wagner

MIFGSM - Momentum Iterative Fast Gradient Sign Method

MSTAR - Moving and Stationary Target Acquisition and Recognition

SAR - Synthetic Aperture Radar

ВСТУП

На сьогоднішній день системи розпізнавання образів є часто використовуваним продуктом. Вони є в камерах, які стоять на вулицях, в системах перевірки зображень, їх використовують для боротьби з спам повідомленнями, які надсилаються у вигляді зображень. Але з розповсюдженням таких систем почали з’являтись і методи, які допомагають ці системи обходити і обманювати. Одними з таких методів є змагальні атаки.

Змагальні атаки – це способи схибити нейромережу для розпізнавання образів таким чином, щоб вона видала неправильний результат. В основному це досягається за допомогою накладання спеціального шуму на зображення, яке майже непомітне для людського ока, але має можливість повністю зіпсувати результати роботи.

Так як багато баз даних для тренування моделей розпізнавання образів є у вільному доступі, то використання таких атак може підставити під сумнів ефективність роботи сучасних систем, що в свою чергу, потребуватиме більш пильного їх налаштування.

РОЗДІЛ 1 ОГЛЯД ІСНУЮЧИХ ПІДХОДІВ ТА МОДЕЛЕЙ ЗМАГАЛЬНИХ АТАК

1.1 Загальна характеристика існуючих атак

Безпека будь-якої моделі машинного навчання оцінюється з урахуванням цілей супротивника та його можливостей доступу до моделі. Для цього потрібне розуміння і ідентифікація поверхонь атак різних реальних додатків, які побудовані з використанням моделей машинного навчання, щоб визначити, як і де зловмисник може спробувати скомпрометувати належну робочу природу системи.

Систему, побудовану на основі машинного навчання, можна розглядати як узагальнений конвеєр обробки даних. Примітивну послідовність операцій системи під час тестування можна розглядати як (a) збір вхідних даних із сховищ даних або датчиків, (б) передачу даних у цифровій області, (в) обробку перетворених даних машиною моделлю навчання для отримання результату і, нарешті, (d) дії, вжиті на основі результату.

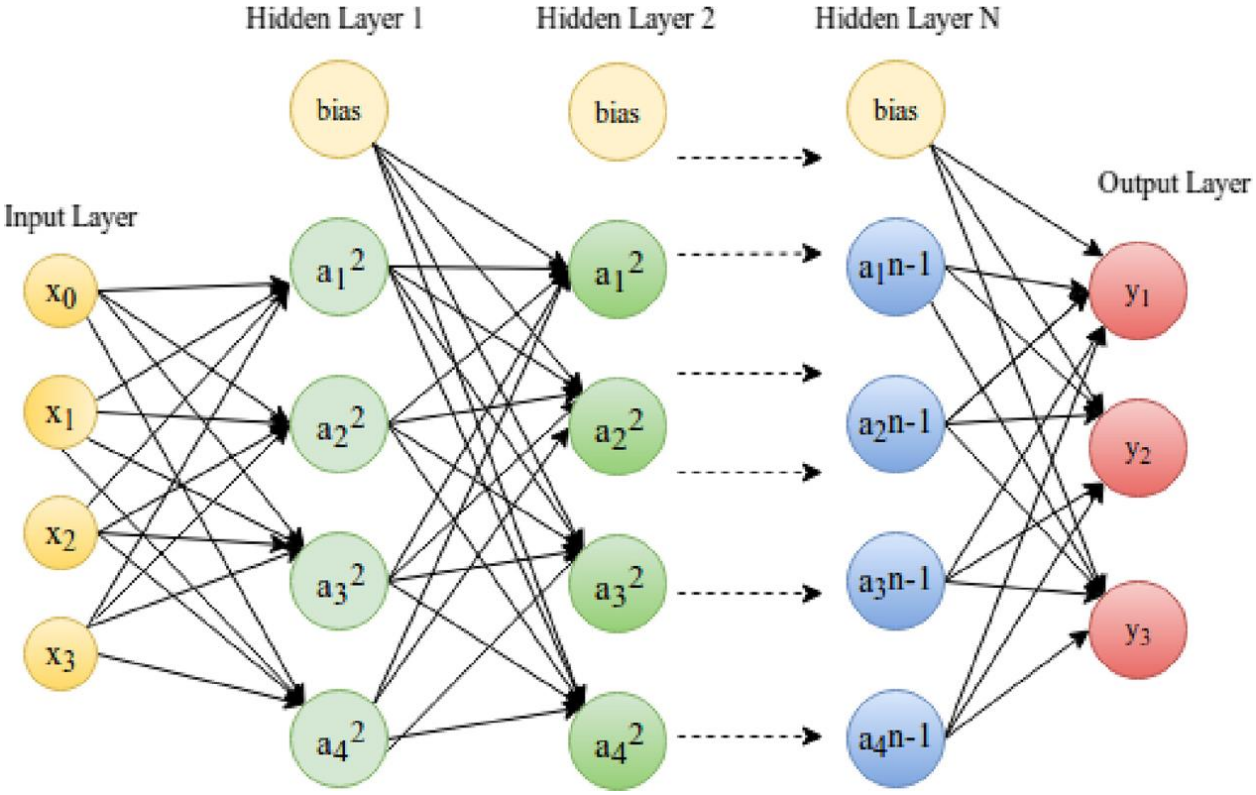

Рисунок 1.1 – Глибока нейронна мережа [1]

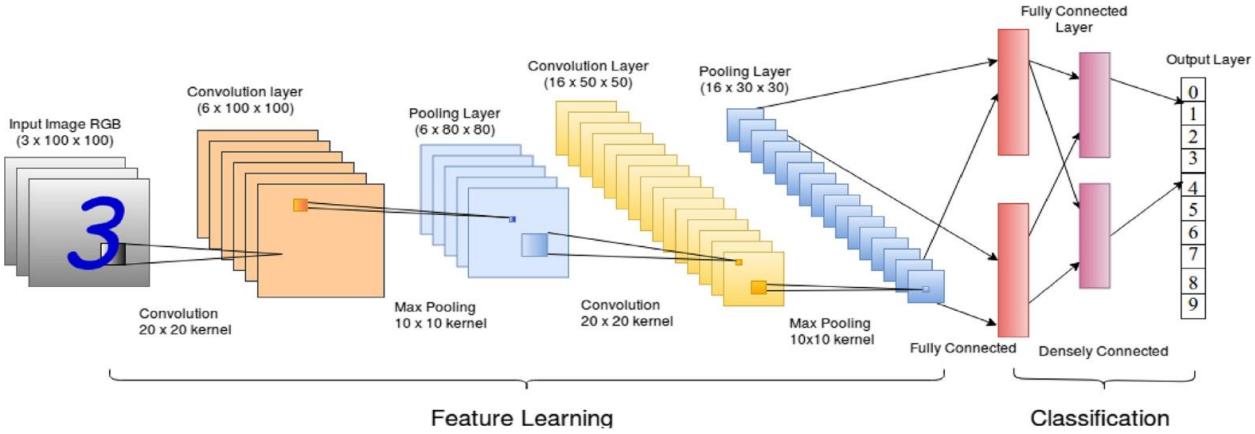

Рисунок 1.2 – Згорткова нейронна мережа для розпізнавання цифр MNIST [1]

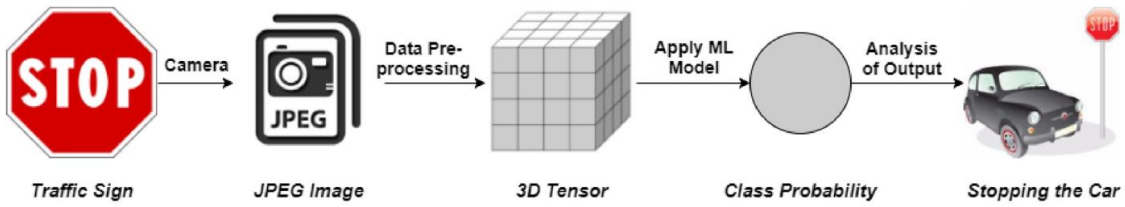

Рисунок 1.3 – Типовий конвеєр автоматизованої транспортної системи [1]

Система збирає вхідні дані датчиків (зображення за допомогою камери), з яких характеристики моделі (тензор значень пікселів) витягуються та передаються в моделі. Потім модель намагається інтерпретувати значення виходу (ймовірність того, що зображення є знаком стоп) і виконує відповідні дії (зупиняє автомобіль). Поверхню атаки в цьому випадку можна інтерпретувати на основі етапів обробки даних. Метою зловмисника може бути спроба маніпулювати збором або обробкою даних, щоб пошкодити цільову модель, таким чином підробивши вихідні дані. Основні сценарії атаки, визначені поверхнею атаки, наведені таким чином:

- Атака ухилення: це найпоширеніший тип атаки у змагальній обстановці. Зловмисник намагається уникнути системи, коригуючи шкідливі зразки

на етапі тестування. Цей параметр не передбачає жодного впливу на дані навчання.

- Атака отруєння: цей тип атаки, відомий як забруднення навчальних даних, здійснюється на етапі навчання моделі машинного навчання. Противник намагається ввести вміло створені зразки, щоб отруїти систему, щоб скомпрометувати весь процес навчання.
- Дослідницькі атаки: ці атаки не впливають на навчальний набір даних. Отримавши доступ до моделі через чорну скриньку, вони намагаються отримати якомога більше знань про алгоритм навчання базової системи та шаблон у навчальних даних.[1]

1.2 Атаки ухилення

Пов’язані роботи щодо прикладу атаки ухилення можна розділити на чотири категорії: інформація про цільову модель, вимірювання відстані, розпізнавання на прикладі змагання та метод генерації.

1.2.1 Інформація про цільову модель

Атаки ухилення також можна розділити на два різні типи, залежно від того, скільки інформації про ціль потрібно для атаки: атака білого ящика та атака чорного ящика. Атака білого ящика використовується, коли зловмисник має детальну інформацію про цільову модель, тобто архітектуру моделі, параметри та ймовірність класу результату. Таким чином, успішність атаки білого ящика досягає майже 100%. Деякі статті, які нещодавно були опубліковані, показали, що захистити DNN від атаки білого ящика надзвичайно важко.

З іншого боку, атака «чорної скриньки» використовується, коли зловмисник може запитати цільову модель без інформації про цільову модель. Добре відомі атаки на чорну скриньку поділяються на два типи: мережева

атака на заміну та атака на передачу. Атака замінної моделі є прикладом добре відомої атаки чорної скриньки. У цій схемі зловмисник може створити альтернативну мережу, подібну до цільової моделі, повторивши процес запиту. Після створення альтернативної мережі зловмисник може здійснити атаку білого ящика. Атаки, створені для мережі-замінника для сервісів Amazon і Google показали, що рівень їх успіху в наборі даних MNIST становив 81,2%[2]

1.2.2 Вимірювання відстані

Існує три способи вимірювання спотворення між оригінальною вибіркою та змагальним прикладом. Перша міра відстані L0 являє собою суму всіх змінених пікселів:

$$\sum_{i=0}^{n}|x_i - x_i^*| \tag{1.1}$$

Де x^i це оригінальний і – й піксель, а x_i^* є змагальним прикладом і-го пікселя. Друга міра відстані L2 представляє стандартну евклідову норму наступним чином:

$$\sum_{i=0}^{n}\sqrt{(x_i - x_i^*)^2} \tag{1.2}$$

Третя міра відстані L∞ є максимальним значенням відстані між x_i та x_i^*. Тому, коли три міри відстані стають малими, схожість зображення зразка з оригінальним зразком збільшується з точки зору людини.[2]

1.2.3 Розпізнавання на змагальних прикладах

Залежно від того, який клас цільова модель розпізнає з змагальних прикладів, категорію змагальних прикладів можна розділити на дві підкатегорії: цільовий змагальний приклад і нецільовий змагальний приклад. По-перше, цільовий змагальний приклад - це змагальний приклад, який змушує цільову модель розпізнавати змагальний образ як певний призначений клас і може бути виражений математично наступним чином.

Враховуючи цільову модель і вихідну вибірку x ∈ X, проблему можна звести до задачі оптимізації, яка генерує цільовий змагальний приклад x^*

$$x^*: \operatorname{argmin}_{x^*}(x, x^*) \ \ s.t. \ f(x^*) = y^* \tag{1.3}$$

Де L(·) є мірою відстані між оригінальним зразком x і перетвореним прикладом x^*, де y^* є конкретним призначеним класом. $\operatorname{argmin}_{x^*} \mathrm{F}(x)$ це значення x, при якому функція F(x) стає мінімальною. c. t. є абревіатурою такого, що. f (·) є операційною функцією, яка надає результати класу для вхідних значень цільової моделі.

З іншого боку, ненацілений змагальний приклад — це змагальний приклад, який змушує цільову модель розпізнавати змагальний образ як клас, відмінний від вихідного класу, і може бути виражений математично наступним чином. Враховуючи цільову модель і вихідну вибірку x ∈ X, проблему можна звести до проблеми оптимізації, яка генерує ненацілений змагальний приклад x^*:

$$x^*: \operatorname{argmin}_{x^*}(x, x^*) \ \ s.t. \ f(x^*) \neq y^* \tag{1.4}$$

Де y ∈ Y — вихідний клас

Перевага ненаціленого змагального прикладу полягає в меншому спотворенні оригінальних зображень і меншому часу навчання порівняно з націленим змагальним прикладом. Однак цілеспрямований змагальний приклад є більш продуманою та потужною атакою для контролю за сприйняттям обраного зловмисником класу.

Донедавна звичайні дослідження вивчали лише одномодельну неправильну класифікацію цілеспрямованих змагальних прикладів. Однак не було опубліковано схеми побудови цілеспрямованих змагальних прикладів, які кілька моделей неправильно класифікують як кожен наданий клас.[2]

1.2.4 Метод генерації змагальних атак

Є чотири типові атаки, які генерують змагальні приклади. Перший метод - це метод знаків швидкого градієнта (FGSM), який може знайти x^* через L∞:

$$x^* = x + \epsilon * sign(\nabla loss_{F,t}(x)) \quad (1.5)$$

Де F — об'єктна функція, а t — цільовий клас. У кожній ітерації FGSM градієнт оновлюється від вихідного x, а x^* визначається за допомогою оптимізації. Цей спосіб простий і має хорошу ефективність. Другим методом є ітераційний FGSM (I-FGSM), який є оновленою версією FGSM. Замість того, щоб змінювати кількість ϵ на кожному кроці, метод оновлює меншу кількість α і зрештою обрізає її тим самим ϵ:

$$x^* = x^*_{i-1} - clip_\epsilon(\alpha * sign\left(\nabla loss_{F,t}(x^*_{i-1})\right)) \quad (1.6)$$

Цей метод I-FGSM забезпечує кращу продуктивність, ніж метод FGSM. Третій - це метод DeepFool, який є нецільовою атакою, яка використовує вимірювання відстані L2. Цей метод генерує змагальний приклад, який є більш ефективним, ніж той, який генерує FGSM, і максимально наближений до вихідного зображення. Щоб створити змагальний приклад за допомогою методу DeepFool, він будує нейронну мережу та шукає x^* за допомогою методу лінійної апроксимації. Однак, оскільки нейронна мережа не є повністю лінійною, метод повинен знаходити змагальний приклад через багато ітерацій, і, таким чином, це більш складний процес, ніж FGSM. Четвертим методом є атака Карліні, яка є останнім методом атаки, що забезпечує найкращу продуктивність порівняно з методами FGSM та I-FGSM. Цей метод може досягти 100% успіху навіть проти дистиляційних структур, які нещодавно були представлені в літературі. Ключовим моментом цього методу є використання іншої цільової функції наступним чином:

$$D(x, x^*) + c * f(x^*), \quad (1.7)$$

Замість використання звичайної цільової функції D(x, x^*), і пропонує спосіб знайти відповідне двійкове значення c. Крім того, пропонується метод контролю успішності атаки, навіть якщо викривлення збільшується, відображаючи значення довіри таким чином:

$$f(x^*) = \max(Z(x^*)_t - \max\{Z(x^*)_t : i \neq t\}, -k), \quad (1.8)$$

Де Z(·) представляє вектор результату класифікації до softmax, де t є цільовим класом.[2]

1.2.5 Стратегія атак ухилення

При знанні мети, знань та можливостей зловмисника, можна формалізувати стратегію атаки, тобто процедуру обфускації шкідливих даних, щоб уникнути виявлення, у термінах задачі оптимізації. Легітимна та зловмисна мітки класу відповідно дорівнюють −1 та +1 і функція прийняття рішення класифікатора є f(x) = sign (g(x)), де g(x) ∈ R є лінійним дискримінантним значенням функції класифікатора, а x - представлення вибірки в d-вимірному просторі ознак. Наприклад, для лінійного класифікатора g(x) = w^t*x + b ∈ R, де w ∈ R^d- ваги ознак, а b ∈ R - зміщення. Враховуючи зловмисний зразок x, мета полягає в тому, щоб знайти зразок x^*, який мінімізує дискримінантну функцію g(·) класифікатора (тобто, який класифікується як законний з найвищою можливою достовірністю) з урахуванням обмеження, що x^* лежить на відстані від d_{max} від x:

$$x^* = \underset{x`}{\operatorname{argmin}}\, g(x`) \quad (1.9)$$

$$d(x`, x) \leq d_{max}, x_{ib} \preccurlyeq x` \preccurlyeq x_{ub}, \quad (1.10)$$

Де $x_{ib} \preccurlyeq x` \preccurlyeq x_{ub}$ представляють обмеження коробки (оскільки функції часто нормалізуються на компактну область), а міра відстані d(·, ·) визначається в термінах вартості маніпулювання даними (наприклад, кількість змінених слів у кожному спамі). Розріджені та щільні атаки ухилення просто визначаються на основі того, чи d(·, ·) відповідає відстані l1 чи l2 відповідно.

Залежно від типу вирішальної функції та метрики відстані, задача (1.9)-(1.10) може бути представлена в термінах задачі лінійного або нелінійного програмування.1 Для лінійних класифікаторів глобальний мінімум можна знайти або у випадку обмеження l1 або l2. Для нелінійного g(x) розв’язок

зазвичай знаходиться в локальному мінімумі цільової функції. Проблему (1.9)-(1.10) можна розв'язати за допомогою стандартних розв'язників в обох випадках, хоча вони можуть бути не дуже ефективними, оскільки вони не використовують специфічні знання про проблему ухилення (наприклад, розрідженість розв'язку, компактну область тощо). Щоб зменшити кількість ітерацій і забезпечити швидку конвергенцію, досліджується одна функція за раз у випадку атак l1 (починаючи з більш перспективної функції, тобто тієї, яка демонструє найвищу варіацію градієнта), оскільки рішення буде розрідженим. І навпаки, одночасно досліджуються всі функції у випадку атак l2, оскільки рішення змінить усі значення функцій. Також мінімізується кількість градієнтів і оцінок функцій, щоб ще більше прискорити алгоритм ухилення: наприклад, повторно обчислюється градієнт g(x), якщо не знайдено кращої точки в досліджуваному напрямку спуску. У випадку нелінійного g(x), використовується кілька ініціалізацій для $x^{(0)}$, щоб пом'якшити проблеми, пов'язані з наявністю кількох локальних мінімумів.

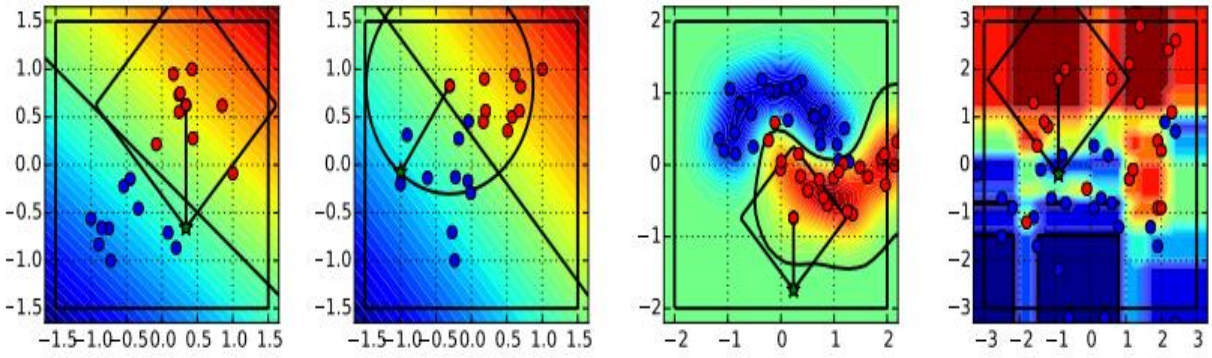

Рисунок 1.4 - Атаки ухилення від різних класифікаторів, навчені на синіх (легітимних) і червоних (зловмисних) зразках. Лінійний класифікатор SVM проти розріджених (перша ділянка) і щільних (друга ділянка) атак ухилення, SVM з ядром RBF (третя ділянка) і випадковим лісом (четверта ділянка) проти розріджених атак ухилення. Початкова шкідлива точка x знаходиться в центрі обмеження відстані, тоді як вибірка ухилення x* позначена зеленою зірочкою. Для кожного класифікатора значення g(x) показано кольорами, а чорна лінія позначає межу рішення. [3]

Приклади (розріджених і щільних) атак ухилення від опорних векторних машин (SVM) і випадкових лісів показані на рисунку 1.4. Оскільки випадкові ліси мають недиференційовану дискримінантну функцію g(x), ми будуємо диференційоване наближення $\hat{g}(x)$ шляхом вивчення сурогатного SVM на тих самих навчальних даних, що використовуються для вивчення випадкового лісу, але замінюючи (справжні) навчальні мітки класифікаційними мітками, призначеними випадковим лісом для таких даних. Потім сурогатний SVM можна використовувати для пошуку відповідного напрямку зниження та запуску атаки ухилення проти випадкового лісу. Вивчення сурогатної (диференційованої) моделі для вирішення проблеми ухилення є більш ефективним з точки зору обчислень. [4]

1.3 Атака отруєнням

Атаки отруєнням - це атаки, які відбуваються на ранніх етапах процесу під час розробки та навчання системи AI. Зазвичай це передбачає маніпулювання даними, які використовуються для навчання самої системи. [5]

Структура атаки поділяється на декілька етапів, першим з яких є моделювання загроз.

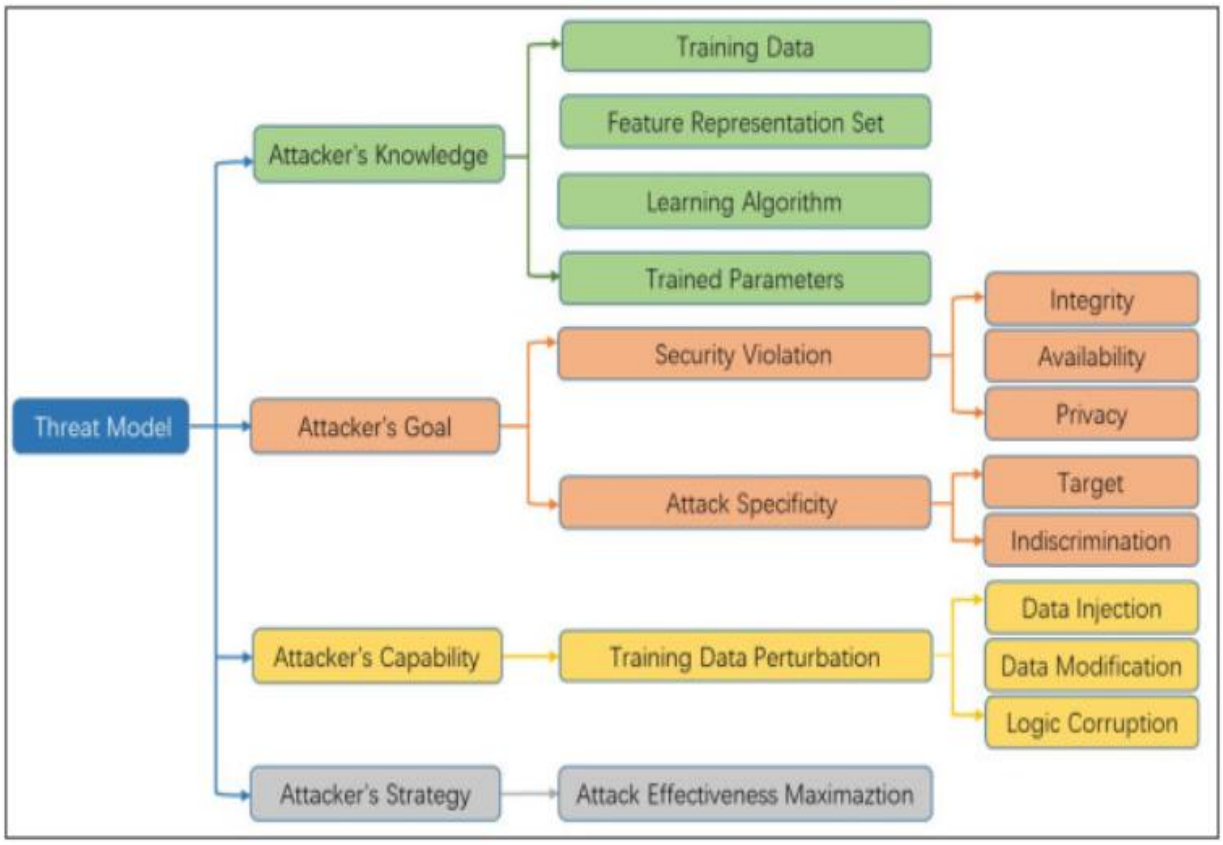

Рисунок 1.5 - Модель атаки отруєння.[6]

Моделювання загроз забезпечує ідентифікацію та кількісну оцінку можливих загроз безпеці та вразливостей в інтелектуальних мережевих системах. Атаки отруєння зосереджені на маніпулюванні даними, зібраними та збереженими пристроями або вузлами на рівні сприйняття системи. Модель загроз атак отруєння пов'язана зі знаннями зловмисника, метою зловмисника, стратегією зловмисника в моделі ML і здатністю зловмисника впливати на навчальні дані, які зведені в рисунку 1.5.[6]

1.3.1 Знання зловмисника

Враховуючи знання компонентів класифікатора, знання зловмисника можна розділити на різні рівні:

- Ідеальне знання: відповідає атакам білої скриньки, коли зловмисник повністю знає цільову систему, включаючи дані навчання, набір

представлень функцій, алгоритм навчання та параметри навчання. Такі знання дозволяють зловмисникам передбачити найгірший сценарій цільової системи.

- Обмежені знання: відповідно до атак сірої скриньки передбачається, що зловмисник має обмежені знання про цільову систему, включаючи сурогатні дані навчання, взяті з аналогічного розподілу, набір представлень функцій, алгоритм навчання та параметри, навчені за допомогою сурогатний класифікатор. Цей параметр дозволяє зловмисникам імітувати більш реалістичний сценарій цільової системи.
- З нульовим знанням: це відповідає атакам із «чорної скриньки», коли зловмисник має мало знань про цільову систему та може лише запитати цільову систему та отримати їхні відповідні відповіді. Порівняно з налаштуваннями білого та сірого ящиків, налаштування чорного ящика є більш складним для зловмисника.[6]

1.3.2 Мета зловмисника

Мета зловмисника - вплинути на результат моделі ML, додавши в систему шкідливі дані, що призведе до неправильної класифікації. Ціль нападника можна охарактеризувати так:

- Порушення безпеки: порушення безпеки можна класифікувати на три випадки: порушення цілісності, успішне впровадження отруєння без впливу на нормальну роботу системи; порушення доступності, що порушує нормальну роботу системи; або порушення конфіденційності, отримання конфіденційної інформації про навчальні дані або користувачів системи.
- Специфіка атаки: специфіку атаки можна розділити на дві частини: одна - це цільова атака, яка намагається передбачити, що вихід моделі ML є конкретною цільовою категорією для будь-якого вхідного сигналу; і

невибіркова атака, яка використовує будь-які зразки функцій, щоб максимізувати помилку передбачення моделі або зробити функціональні можливості системи ML недоступними. Наприклад, датчики в середовищах IoT, які збираються для мережі, можна зробити непридатними для використання, тим самим знизивши продуктивність системи.[6]

1.3.3 Можливості зловмисника

Можливості зловмисника стосуються здатності та стратегії контролювати функції або мітки навчальних даних, зібраних пристроями в різних інтелектуальних мережах, скільки зловмисних даних впроваджується або змінюється, і на яку частину навчальних даних впливає зловмисник. Як правило, зловмисник, який має набагато більше інформації про навчальні дані, називається сильнішим зловмисником, а слабший зловмисник часто отримує мало знань, які передаються в модель під час навчання. Оскільки вищий рівень отруєння, швидше за все, призведе до неправильної роботи, існуючі атаки зазвичай контролюють лише вибрану невелику частину навчальних даних для запуску атак, з наступними трьома діями для маніпулювання моделлю:

- Ін’єкція даних: зловмисник може пошкодити цільову модель, вставивши кілька отруєних зразків у навчальний набір. Однак зловмисник не може змінити навчальні дані, а також алгоритми машинного навчання. Очевидно, що зловмисник може лише забруднити ярлики шляхом неконтрольованого навчання.
- Модифікація даних: зловмисник може отримати доступ до навчальних даних і маніпулювати значеннями їхніх атрибутів або відповідними мітками, але не може отримати та змінити алгоритми ML.
- Порушення логіки: зловмисник може маніпулювати алгоритмами ML (наприклад, параметрами або структурою алгоритмів), що називається

пошкодженням логіки. Таким чином, розробити контрзаходи проти такого роду атак нетривіально.[6]

1.3.4 Стратегія зловмисника

Стратегія зловмисника може бути описана тим, як кількісно змінити навчальні зразки для максимізації ефективності атаки та отримання оптимальних ефектів від поставленої зловмисником мети. Зокрема, у ньому пояснюється, як маніпулювати характеристиками навчальних даних, як змінити мітку класу та яка частина навчальних даних має більший вплив на модель ML.

Відповідно до припущень різних рівнів знань, стратегія зловмисника може бути виражена як дворівнева оптимізаційна задача через умови Каруша-Куна-Таккера (KKT): проблема верхнього рівня та проблема нижнього рівня. Якщо взяти атаку білої скриньки як приклад, проблему верхнього рівня можна описати як вибір даних отруєння для максимізації функції втрат алгоритмів ML на наборі даних перевірки, тоді як проблему нижчого рівня можна розглядати як перенавчання алгоритмів ML на отруєний набір даних, щоб мінімізувати його функцію втрати. У налаштуванні чорної скриньки зловмисник вирішує проблему, використовуючи замінені навчальні дані замість початкових навчальних даних.[6]

1.3.5 Використання атак отруєнням в глибокому навчанні

За останні кілька років глибинне навчання стало популярним напрямком досліджень. Незважаючи на те, що багато методів отруєння були інтенсивно вивчені в традиційних алгоритмах ML, лише деякі з них розроблені для глибоких нейронних мереж (DNN). Як класична модель глибокого навчання, моделі DNN показали чудову продуктивність у різних завданнях розпізнавання, наприклад, класифікація зображень, комп’ютерний зір тощо.

Типову методологію отруєння можна сформулювати як математичний вираз. Однак DNN важко отруїти через його складність. Для різних видів зловмисників вони використовують різні методи для атаки на DNN. В основному є два типи зловмисників.[6]

З точки зору сильних зловмисників передбачається, що вони мають повне знання алгоритмів навчання та навчальних даних, подібно до атак білої скриньки. Основна ідея полягає в оновленні параметрів алгоритму навчання за допомогою градієнтного спуску та правильного відстеження змін даних у зворотному напрямку. Оскільки метод оптимізації на основі градієнта вимагає суворого припущення цільової функції ця схема незастосовна до нейронних мереж, і тому проводиться оптимізація зворотного градієнта для генерації навчальних зразків отруєння. Ці оптимальні атаки отруєння можна змоделювати як задачі дворівневої оптимізації, і для оптимізації зворотного градієнта не потрібні умови ККТ, які можна застосувати до граничного діапазону алгоритмів. Одним з перших застосувань градієнтного методу до DNN є два методи отруєння, включаючи метод прямого градієнта та генеративний, натхненний концепцією Generative Adversarial Network (GAN). Недоліки цих методів в тому, що на практиці припущення білого ящика рідко виконується в реальних умовах.

З точки зору слабких зловмисників передбачається, що вони частково або зовсім не знають алгоритмів навчання та навчальних даних, подібно до атак сірої або чорної скриньки. В дослідженнях припускається, що зловмисник не знає моделі і може ввести лише невелику частину навчальних даних. На цій основі пропонуються два типи стратегій: стратегію введення-примірник-ключ і стратегію шаблон-ключ. Перший спрямований на створення відносно невеликого діапазону бекдорів порівняно з ключем. Потім зловмисник вибирає зображення як ключове зображення та визначає його як цільову мітку. З іншого боку, останній спрямований на створення відносного межового діапазону екземплярів бекдору, а ключ розглядається як шаблон, щоб зловмисники могли вставити випадковий шаблон у вхідний зразок або

застосувати аксесуар до вхідного зразка. Однак головним обмеженням цього дослідження є те, що атака буде менш ефективною, якщо застосовувати її до деяких складних зображень. У реальних сценаріях зловмисник не може контролювати дані тестування та мітку екземплярів у навчальному наборі даних. Інший тип атаки отруєння називається цільовою атакою чистої мітки. Передбачається, що зловмисники вводять чітко позначені, мінімально збурені зразки в навчальні дані, щоб неправильно класифікувати цільові зразки. У цьому варіанті припускається, що зловмисник має доступ до знань про архітектуру моделі. Згідно з цим припущенням, використовується метод вирівнювання градієнта, який використовує метрику косинусної подібності, щоб імітувати градієнт жертви, навченої на цільовому неправильно позначеному наборі, щоб створювати збурення для отруєних екземплярів. Головним недоліком цього підходу є сильні шаблони створених збурень, які легко виявити. Також був винайдений метод, подібний до BadNets, проти системи класифікації тексту на основі LSTM. В ньому припускається, що зловмисник має доступ до часткових зразків навчання, але не отримує знання про модель або алгоритми навчання. Крім того, семантично правильне речення розглядається як бекдор-тригер і вводиться в доброякісні навчальні зразки випадковим чином. Однак тригер, створений зловмисником, мав би очевидний шаблон, і його можна було б виявити шляхом оцінки впливу кожного слова.[6]

1.4 Дослідницькі атаки (Exploratory attacks)

Дослідницькі атаки не змінюють навчальний набір, а натомість намагаються отримати інформацію про стан, досліджуючи учня. Змагальні приклади створені таким чином, що учень передає їх як законні приклади на етапі тестування.[7]

1.4.1 Атака інверсії моделі

Одну з перших атак інверсії моделі(MI) представив Фредріксон та ін. ,де вони використовували модель лінійної регресії f для прогнозування дозування ліків, використовуючи інформацію про пацієнта, історію хвороби та генетичні маркери; досліджував модель як білу коробку та екземпляр даних і намагався вивести генетичний маркер x1. Алгоритм створює «найменш зміщену максимальну апостеріорну (MAP) оцінку» для x1 шляхом повторення всіх можливих значень номінальної характеристики (x1) для отримання цільового значення y, таким чином мінімізуючи рівень помилкових прогнозів супротивника. Він має серйозні обмеження; наприклад, він не може обробляти більший набір невідомих функцій, оскільки це неможливо з точки зору обчислень.[7]

Фредріксон та ін. мали намір усунути обмеження своєї попередньої роботи та показали, що генетичні маркери пацієнта можуть бути передбачені зловмисником у середовищі чорної скриньки. Ця нова атака MI через API машинного навчання (ML), які використовують значення довіри в різноманітних налаштуваннях і вивчають контрзаходи як у налаштуваннях чорного, так і білого ящиків. Ця атака була успішно експериментована з розпізнаванням обличчя з використанням моделей нейронних мереж: багатошарового персептрона (MLP), регресії softmax і стекової мережі автоматичного кодування шуму (DAE); отримавши доступ до моделі та імені людини, він може відновити зображення обличчя. Рисунок 1.6 ілюструє реконструкцію, виконану трьома алгоритмами. Через заплутану структуру моделі DNN атаки MI не можуть отримати більше, ніж кілька прототипних зразків, які можуть не мати великої схожості з вихідними даними, які використовувалися для визначення цього класу.[7]

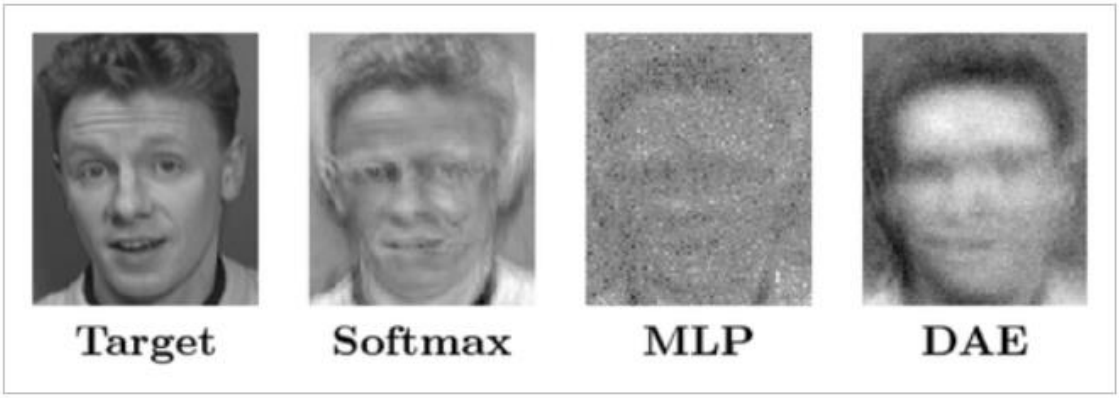

Рисунок 1.6 - Реконструкція особи ліворуч за допомогою Softmax, багатошарового персептрона та мережі автокодерів із шумозаглушенням

1.4.2 Атаки на конфіденційність

Атаки на конфіденційність - це атаки, цілі яких - шпигунство та конфіденційність. Зловмисник має намір дослідити систему, наприклад модель або набір даних, які можуть стати в нагоді. Враховуючи велику кількість систем і власних алгоритмів, однією з цілей є отримання знань про систему AI та її модель - атаки вилучення моделі.

Що стосується атак, пов’язаних із даними, можна отримати інформацію про набір даних, здійснюючи такі атаки, як висновок про членство та атрибути даних за допомогою виведення атрибутів. Нарешті, атака інверсії моделі допомагає витягти певні дані з моделі.

Більшість досліджень наразі охоплюють атаки логічного висновку на етапі виробництва, але вони також можливі під час навчання. Якщо ми зможемо ввести навчальні дані, ми зможемо дізнатися, як працює алгоритм на основі цих даних. Наприклад, якщо ми хочемо зрозуміти, як веб-сайт соціальних мереж вирішує, що ви належите до цільової аудиторії, скажімо, до групи вагітних жінок, щоб показати вам певну рекламу, ми можемо змінити свою поведінку, наприклад, намагаючись шукати інформацію про підгузки та перевіряти, чи ми отримуємо рекламу, призначену для майбутніх мам.[8]

Висновок про членство є менш частим типом атак, але першим етапом для вилучення даних. Висновок про членство - це атака, у якій є намір

дізнатися, чи був конкретний приклад у наборі даних. Якщо говорити про розпізнавання зображень, потрібно перевірити, чи була конкретна особа в наборі навчальних даних. Це спосіб зрозуміти, як постачальник AI дотримується правил конфіденційності.[8]

Крім того, це може допомогти спланувати подальші атаки, наприклад атаки ухилення від чорної скриньки, які базуються на можливості передачі. Під час атак із можливістю перенесення, чим більше набір даних схожий на набір даних жертви, тим більше шансів навчити свою модель бути подібною до моделі жертви. Виведення атрибутів допомагає отримати цінну інформацію про навчальні дані (наприклад, акцент мовців у моделях розпізнавання мовлення).[8]

Висновок за атрибутами (вгадування типу даних) і приналежність (конкретні приклади даних) є життєво важливими не лише через проблеми конфіденційності, але й як етап дослідження для атак ухилення.

Наприклад, атака на членство на рисунку 1.7 полягає в здогадуванні, чи була ця конкретна собака в наборі даних для навчання.[8]

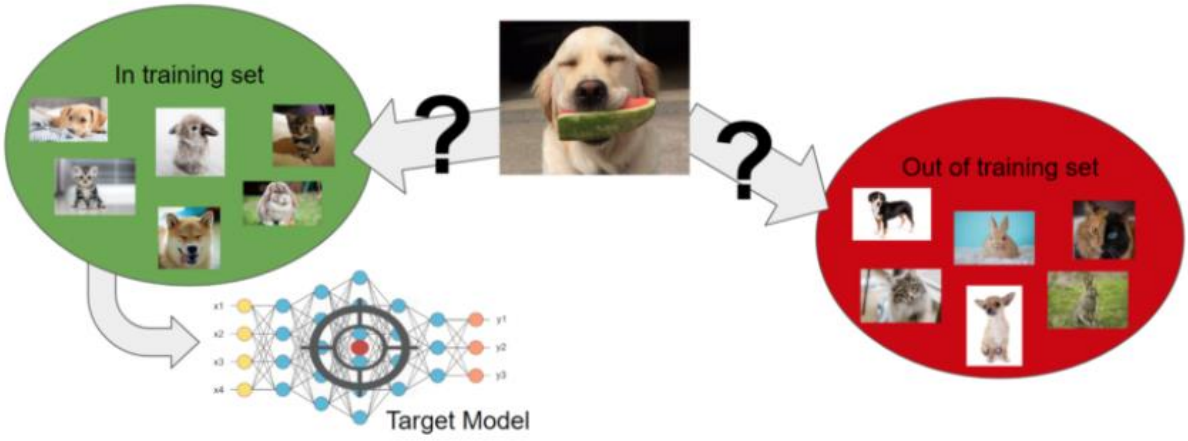

Рисунок 1.7 Атака на членство [8]

Висновки до розділу 1

Є такі три основні типи змагальних атак на системи розпізнавання образів:

Атаки ухилення - є найпоширенішими та найбільш дослідженими видами атак. Зловмисник маніпулює даними під час розгортання, щоб ввести в оману попередньо навчених класифікаторів. Оскільки вони виконуються на етапі розгортання, це найбільш практичні типи атак і найчастіше використовувані атаки на сценарії вторгнення та зловмисного програмного забезпечення.

Атаки отруєння - зловмисник впливає на навчальні дані або їх мітки, щоб призвести до зниження продуктивності моделі під час розгортання. Отже, отруєння - це по суті змагальне зараження навчальних даних. Оскільки системи ML можна перенавчати за допомогою даних, зібраних під час роботи, зловмисник може отруїти дані, впроваджуючи шкідливі зразки під час роботи, які згодом порушують або впливають на повторне навчання.

Дослідницькі атаки - не змінюють навчальний набір, а натомість намагаються отримати інформацію про стан, досліджуючи учня. Змагальні приклади створені таким чином, що учень передає їх як законні приклади на етапі тестування.

РОЗДІЛ 2 АНАЛІЗ ТА ОЦІНКА ВИКОРИСТАНИХ МОДЕЛЕЙ НЕЙРОННИХ МЕРЕЖ ТА ЗМАГАЛЬНИХ АТАК

2.1 Моделі нейронних мереж на основі бібліотеки tensorflow

Нейронні мережі, також відомі як штучні нейронні мережі (ANN) або імітовані нейронні мережі (SNN), є підмножиною машинного навчання та є основою алгоритмів глибокого навчання. Їх назва та структура навіяні людським мозком, імітуючи спосіб, яким біологічні нейрони передають сигнали один одному.[9]

Штучні нейронні мережі (ШНМ) складаються з вузлових шарів, що містять вхідний рівень, один або більше прихованих шарів і вихідний рівень. Кожен вузол, або штучний нейрон, з'єднується з іншим і має відповідну вагу та поріг. Якщо вихід будь-якого окремого вузла перевищує вказане порогове значення, цей вузол активується, надсилаючи дані на наступний рівень мережі. В іншому випадку дані не передаються на наступний рівень мережі.[11]

Нейронні мережі покладаються на навчальні дані, щоб навчатися та підвищувати свою точність з часом. Однак, коли ці алгоритми навчання точно налаштовані на точність, вони стають потужними інструментами в інформатиці та штучному інтелекті, що дозволяє класифікувати та кластеризувати дані з високою швидкістю. Завдання з розпізнавання мовлення або розпізнавання зображень можуть тривати хвилини чи години порівняно з ручною ідентифікацією експертів-людей. Використання нейронних мереж для супутникових знімків також є важким і тривалим ділом, як і їх налаштовування, що можна подивитись в статті[10]. Однією з найвідоміших нейронних мереж є пошуковий алгоритм Google.[11]

Tensorflow - відкрита програмна бібліотека машинного навчання, розроблена компанією Google для вирішення завдань побудови і тренування нейронної мережі з автоматичним розташуванням і класифікацією образів, досягаючи якості людського виховання.[11]

2.1.1 Багатошаровий персептрон

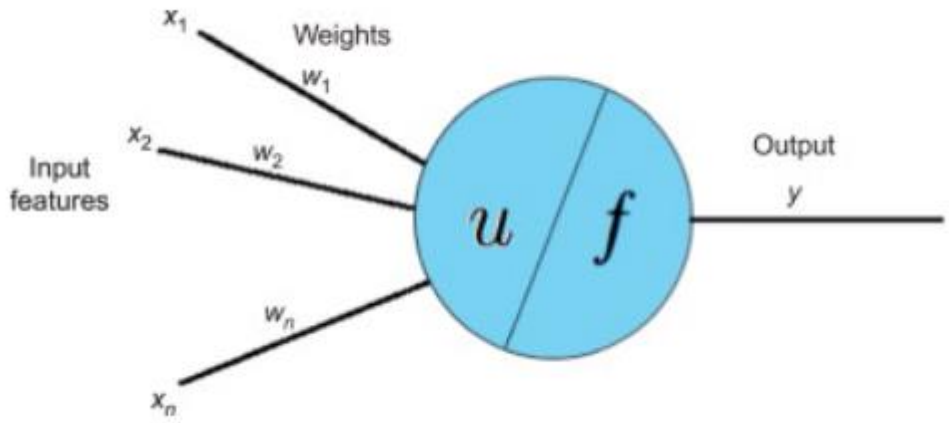

Рисунок 2.1 - Схема персептрона з n вхідними ознаками.[12]

Багатошаровий персептрон(MLP) - це модель нейронних мереж, яка працює як універсальний апроксиматор, тобто може апроксимувати будь-яку безперервну функцію. Наприклад, їх можна використовувати як моделі SEE. MLP складаються з нейронів, які називаються шарами. Як показано на рисунку 2.1, персептрон отримує n характеристик як вхідні дані (x = x1, x2,…, xn), і кожна з цих характеристик пов’язана з вагою. Вхідні елементи мають бути числовими. Таким чином, нечислові функції введення потрібно перетворити на числові, щоб використовувати персептрон. Наприклад, категоріальна ознака з p можливими значеннями може бути перетворена на p вхідних ознак, що представляють наявність/відсутність цих значень. Вони називаються фіктивними змінними. Наприклад, якщо вхідна функція «тип розробки» може приймати значення «нова розробка», «покращення» або «повторна розробка», її можна замінити трьома фіктивними змінними «нова розробка», «покращення» та « перепланування», які приймають значення 1, якщо відповідне значення є, і 0, якщо воно відсутнє.[13]

Вхідні ознаки передаються до вхідної функції u, яка обчислює зважену суму вхідних ознак:

$$u(x) = \sum_{i=1}^{n} w_i x_i \qquad (2.1)$$

Результат цього обчислення потім передається на функцію активації f, яка вироблятиме вихідні дані персептрона. В оригінальному персептроні функція активації є ступінчастою функцією:

$$y = f\big(u(x)\big) = \begin{cases} 1, if\ u(x) > \theta \\ 0, otherwise \end{cases} \quad (2.2)$$

Де θ – пороговий параметр. Приклад ступінчастої функції з θ = 0 показано на рисунку 2.2 зліва. Таким чином, ми бачимо, що персептрон визначає, чи є $w_1x_1 + w_2x_2 + \cdots + w_nx_n - \theta > 0$ істинним чи хибним. Рівняння $w_1x_1 + w_2x_2 + \cdots + w_nx_n - \theta = 0$ є рівнянням гіперплощини. Персептрон виводить 1 для будь-якої вхідної точки над гіперплощиною та виводить 0 для будь-якого входу на або під гіперплощиною. З цієї причини перцептрон називають лінійним класифікатором, тобто він добре працює для даних, які можна розділити лінійно. Навчання персептронів полягає в регулюванні ваг таким чином, щоб визначити гіперплощину, яка добре розділяє навчальні дані.[13]

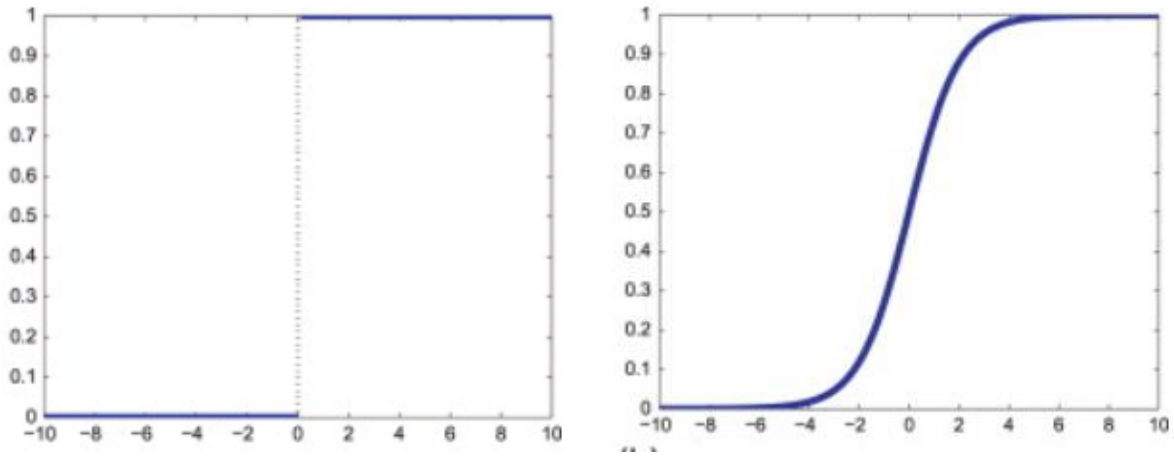

Рисунок 2.2 - Приклади функцій активації.[14]

MLP здатні апроксимувати будь-яку неперервну функцію, а не лише лінійні функції. Вони роблять це шляхом поєднання кількох нейронів, які організовані принаймні в три шари:

- Один вхідний шар, який просто розподіляє вхідні функції на перший прихований шар.

- Один або кілька прихованих шарів перцептронів. Перший прихований шар отримує як вхідні дані функції, розподілені вхідним шаром. Інші приховані шари отримують як вхідні дані кожного персептрона з попереднього шару.
- Один вихідний рівень персептронів, які отримують як вхідні дані кожного персептрона останнього прихованого шару.[14]

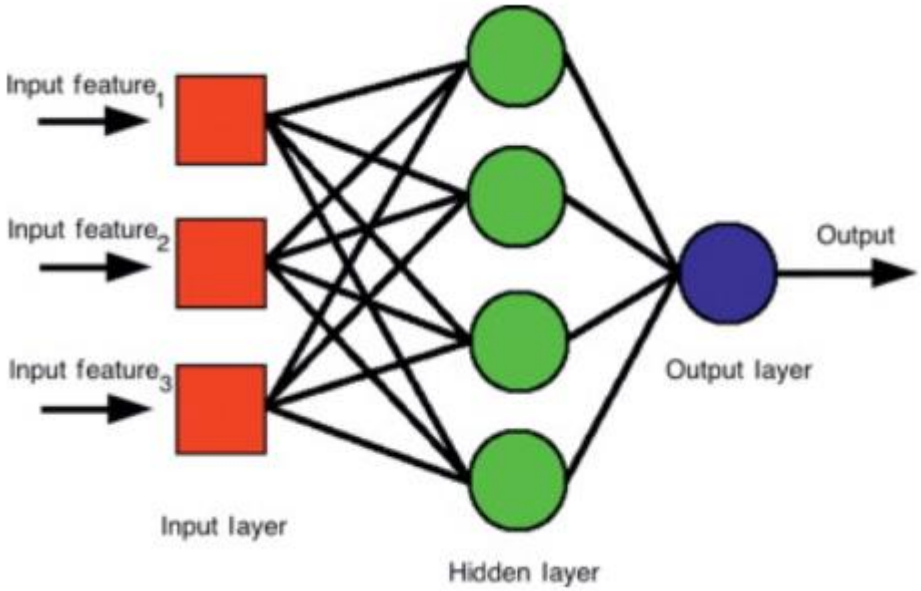

Рисунок 2.3 - Схема тришарового MLP з трьома вхідними функціями, чотирма прихованими нейронами та одним виходом.[15]

На рисунку 2.3 показана схема MLP з трьома шарами. Перцептрони, що використовуються MLP, часто використовують інші типи функцій активації, ніж функція кроку. Для нейронів прихованого шару часто використовуються сигмоподібні функції. Приклад сигмоїдної функції показано на рис. 2.2 справа. Сигмоїдні функції призведуть до плавних переходів замість жорстких меж рішень, як при використанні ступінчастих функцій. Функція активації нейронів вихідного рівня зазвичай є сигмоподібною для проблем класифікації та функцією ідентичності для задач регресії.[15]

2.1.2 Згорткова нейронна мережа

Нейронні мережі є підмножиною машинного навчання, і вони є серцевиною алгоритмів глибокого навчання. Вони складаються з шарів вузлів, що містять вхідний рівень, один або більше прихованих шарів і вихідний рівень. Кожен вузол з'єднується з іншим і має відповідну вагу та поріг. Якщо вихід будь-якого окремого вузла перевищує вказане порогове значення, цей вузол активується, надсилаючи дані на наступний рівень мережі. В іншому випадку дані не передаються на наступний рівень мережі.[13]

Існують різні типи нейронних мереж, які використовуються для різних випадків використання та типів даних. Наприклад, рекурентні нейронні мережі зазвичай використовуються для обробки природної мови та розпізнавання мовлення, тоді як згорткові нейронні мережі (ConvNets або CNN) частіше використовуються для завдань класифікації та комп'ютерного зору. До CNN для ідентифікації об'єктів на зображеннях використовувалися ручні, трудомісткі методи виділення ознак. Проте згорткові нейронні мережі тепер забезпечують більш масштабований підхід до задач класифікації зображень і розпізнавання об'єктів, використовуючи принципи лінійної алгебри, зокрема множення матриць, для ідентифікації шаблонів у зображенні. Тим не менш, вони можуть бути вимогливими до обчислень, вимагаючи графічних процесорів (GPU) для навчання моделей.[13]

Згорткові нейронні мережі відрізняються від інших нейронних мереж своєю чудовою продуктивністю з зображенням, мовленням або аудіосигналом. Вони мають три основні типи шарів, а саме:

- Згортковий шар
- Шар об'єднання
- Повністю підключений (FC) рівень

Згортковий шар є першим шаром згорткової мережі. У той час як за згортковими шарами можуть слідувати додаткові згорткові шари або шари об'єднання, повністю зв'язаний рівень є останнім. З кожним шаром CNN зростає у своїй складності, ідентифікуючи більші частини зображення. Попередні шари зосереджені на простих функціях, таких як кольори та краї.

Коли дані зображення просуваються між шарами CNN, вони починають розпізнавати більші елементи або форми об’єкта, поки нарешті не ідентифікують запланований об’єкт.[13]

Згортковий рівень є основним будівельним блоком CNN, і на ньому відбувається більшість обчислень. Для цього потрібно кілька компонентів, якими є вхідні дані, фільтр і карта функцій. Припустімо, що вхідним буде кольорове зображення, яке складається з матриці пікселів у 3D. Це означає, що вхідні дані матимуть три виміри - висоту, ширину та глибину - які відповідають RGB зображення. У нас також є детектор ознак, також відомий як ядро або фільтр, який переміщатиметься по сприйнятливим полям зображення, перевіряючи, чи присутня функція. Цей процес відомий як згортка.[13]

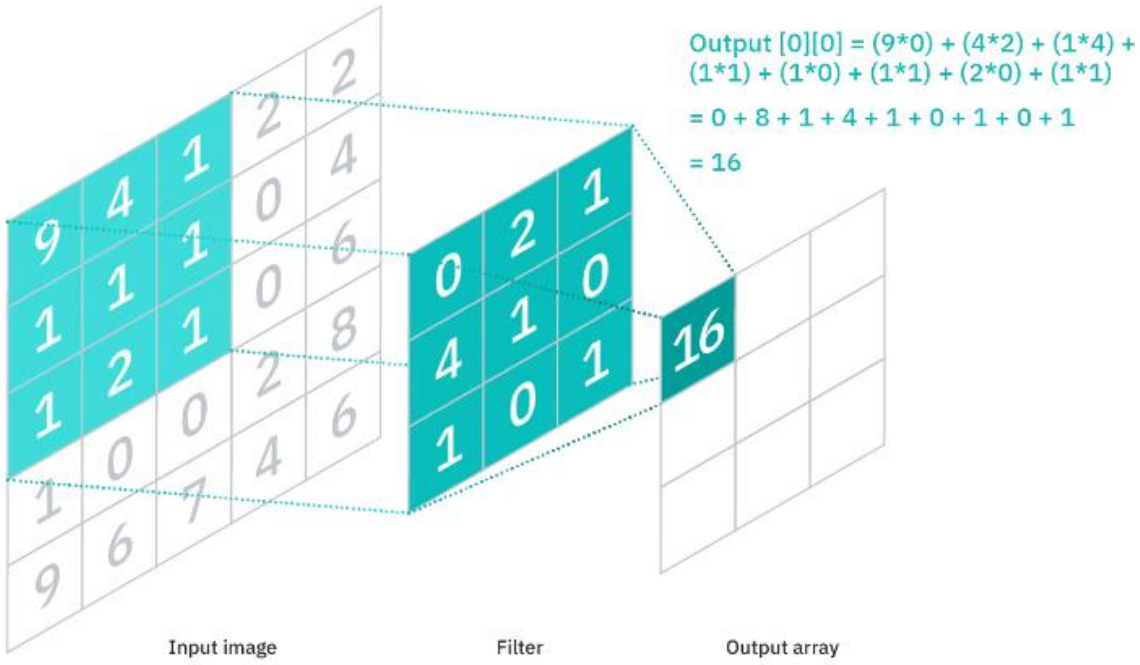

Рисунок 2.4 - Приклад роботи згорткового рівня нейронної мережі [13]

Об’єднання шарів, також відоме як зменшення дискретизації, виконує зменшення розмірності, зменшуючи кількість параметрів у вхідних даних. Подібно до згорткового рівня, операція об’єднання очищує фільтр по всьому входу, але відмінність полягає в тому, що цей фільтр не має вагових коефіцієнтів. Замість цього ядро застосовує функцію агрегації до значень у

приймальному полі, заповнюючи вихідний масив. Існує два основних типи об’єднання:

Максимальне об’єднання - під час руху фільтра по входу він вибирає піксель із максимальним значенням для надсилання у вихідний масив. Крім того, цей підхід зазвичай використовується частіше, ніж середнє об’єднання.

Об’єднання середніх значень - під час руху фільтра по входу він обчислює середнє значення в приймальному полі для надсилання у вихідний масив.[13]

Хоча багато інформації втрачається на рівні об’єднання, це також має ряд переваг для CNN. Вони допомагають зменшити складність, підвищити ефективність і обмежити ризик переобладнання.

Назва повнозв’язного шару влучно описує саму себе. Як згадувалося раніше, значення пікселів вхідного зображення не пов’язані безпосередньо з вихідним шаром у частково з’єднаних шарах. Однак на повністю підключеному рівні кожен вузол вихідного рівня з’єднується безпосередньо з вузлом попереднього рівня.

Цей рівень виконує завдання класифікації на основі ознак, витягнутих через попередні шари та їхні різні фільтри. У той час як рівні згортки та об’єднання, як правило, використовують функції ReLu, рівні FC зазвичай використовують функцію активації softmax для відповідної класифікації вхідних даних, виробляючи ймовірність від 0 до 1.[13]

2.1.3 Згорткова нейронна мережа VGG19

VGG CNN має шість основних структур, кожна з яких в основному складається з кількох зв’язаних згорткових шарів і повнозв’язаних шарів. Розмір згорткового ядра становить 3*3, а вхідний розмір — 224*224*3. Кількість шарів, як правило, становить 16-19. Структура моделі VGG-19 показана на рисунку 2.5.[14]

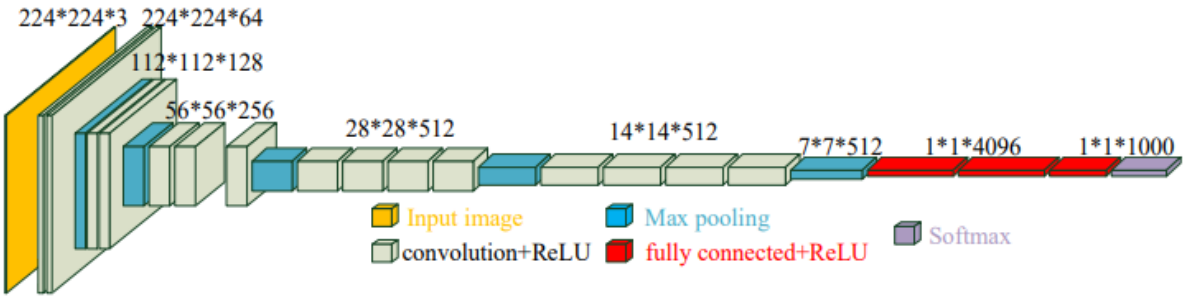

Рисунок 2.5 Модель VGG-19[14]

VGG-19 CNN використовується як модель попередньої обробки. Порівняно з традиційними згортковими нейронними мережами, глибина мережі була покращена. Вона використовує змінну структуру кількох згорткових шарів і нелінійних шарів активації, що краще, ніж одна згортка. Структура шарів може краще витягувати характеристики зображення, використовувати Maxpooling для зменшення дискретизації та змінювати лінійну одиницю (ReLU) як функцію активації, тобто вибрати найбільше значення в області зображення як об'єднане значення області. Рівень зменшення дискретизації в основному використовується для покращення здатності мережі проти спотворень зображення, зберігаючи основні характеристики вибірки та зменшуючи кількість параметрів. [14]

Мережа складається з таких шарів:

- Нормалізація нульового центру(Zero-Center normalization) - це централізація та нормалізація до Origo. Він нормалізує та зменшує розміри - щоб масштаб був централізованим - з точки зору того, коли ми будемо виконувати згортку пізніше. Причина, по якій це важливо, полягає в тому, щоб привести все «у відповідність» у нормалізований, спрощений і впорядкований спосіб, щоб у нас було певне відчуття нормального стану. Як і в тому, що потрібно щоб загальна структура того, що розбирається, була нормалізована та централізована, щоб була заздалегідь визначена межа, щодо якої ми будемо працювати.
- Згортки(Convolutions) - це функціональна операція виконання конкатенації функціональних кривих, щоб вони «сумувалися, щоб

інкапсулювати, як вони впливають одна на одну - з точки зору мультиплікативних зв’язків» - грубо кажучи. Робота полягає в тому, що фактично «множити зв’язок функції з іншою» - таким чином отримується середнє значення того, як вони впливають одна на одну - у середньому.

- РеЛУ - випрямна лінійна установка. Блок, створений для виправлення та компенсації помилок аналізу сигналу, наприклад, коли ми змушені інкапсулювати епсилон (нескінченно малий), і ми повинні це обійти. Тож ReLU «Направляє сигнал» туди, куди він має бути спрямований, щоб сигнал не вибухнув і не зник. Ось такі дії ReLU схожі на «зупинки безпеки» на узбіччях доріг, щоб ви не з’їжджали з дороги, коли ви їдете.
- Максимальне об’єднання(Max Pooling) - це коли відбувається об’єдннання найбільшого зразка, який можливо знайти - у середній області кроків. Strides - це середнє функціональне відображення ядра, яке є на квадратній діаграмі, яке усереднює вибірки значень певного простору. Отже - може бути загальний квадрат 8x8 - із 4 квадратами 4x4. Він зменшується до максимального значення кожного квадрата 4x4, потім виконується складання їх разом і отримується загальний квадрат 4x4, щоб замінити попередній 8x8. Це фактично беруться «найбільш впливові характеристики», а також «найбільша інформація для отримання» (або найгрубіші характеристики). А потім ігноруються дрібніші функції, і залишається менший простір вибірки (менша розмірність), але зберігається найбільше представлення інформації. Це компроміс заради обчислень.
- Повністю зв’язаний рівень(Fully Connected Layer) - це повністю зв’язаний рівень, який використовується для класифікації. Він використовується дуже економно, коли приблизне виділення ознак і усереднення функціональних зв’язків мають свою послідовність. Також підсумовується для підрахунку загального результату.

- Softmax - це аплікативний алгоритм, який нормалізує динаміку розподілу з K кількості складених функціональних конкатенацій. Це означає, що замість розкиду значення можуть бути будь-якими - 0, -1, 1, 2 тощо. Усі вони нормалізуються відповідно до розподілу (оскільки Softmax діє як регуляризатор над розподілом)
- Результати класифікації(Classification output) майже такий, як і звучить. Це результат класифікації, у якій використовується перехресна ентропія для максимізації ймовірності, з точки зору того, звідки походить кожна K-та одиниця та де вона, швидше за все, знаходиться.[15]

2.2 Моделі змагальних атак

Змагальне машинне навчання, техніка, яка намагається обдурити моделі за допомогою оманливих даних, є зростаючою загрозою в дослідницькому співтоваристві AI та машинного навчання. Найпоширенішою причиною є збій у моделі машинного навчання. Змагальна атака може спричинити представлення моделі з неточними чи спотвореними даними під час її навчання або введення зловмисно розроблених даних, щоб ввести в оману вже навчену модель.

Як зазначається в проміжному звіті Комісії національної безпеки США зі штучного інтелекту за 2019 рік, дуже невеликий відсоток поточних досліджень ШІ спрямований на захист систем ШІ від зусиль противника. Деякі системи, які вже використовуються у виробництві, можуть бути вразливими до атак. Наприклад, розмістивши кілька маленьких наклейок на землі, дослідники показали, що вони можуть змусити безпілотний автомобіль переїхати на зустрічну смугу руху. Інші дослідження показали, що внесення непомітних змін до зображення може змусити систему медичного аналізу класифікувати доброякісну родимку як злоякісну, а шматки стрічки можуть ввести в оману систему комп’ютерного зору, щоб неправильно класифікувати знак «стоп» як знак обмеження швидкості.[16]

Зростання впровадження штучного інтелекту, ймовірно, буде пов’язане зі збільшенням кількості атак противника. Це нескінченна гонка озброєнь, але, на щастя, сьогодні існують ефективні підходи для пом’якшення найгірших атак.[16]

2.2.1 Метод швидкого градієнта (Fast Gradient Sign Method)

Метод швидкого градієнтного знака (FGSM) фокусується на ефективному пошуку змагального збурення, обмеженого L_∞-нормою. Під час навчання моделі ML задана функція втрат мінімізується, щоб знайти оптимальний набір параметрів Θ для класифікатора C так, щоб C правильно класифікував більшість навчальних даних. На відміну від цього, FGSM максимізує функцію втрат, оскільки намагається змусити класифікатор працювати погано на змагальному прикладі x'. Таким чином, збурене зображення FGSM для нецільової атаки будується шляхом вирішення такої проблеми максимізації:[21]

$$\max_{x' \in [0,1]^n} L(x', y) \tag{2.3}$$

З урахуванням обмеження

$$\|x' - x\| \leq \varepsilon, \tag{2.4}$$

Де y - це мітка базової істинності вихідного зображення x, а ε - деяке максимальне збурення, що залежить від програми.

Застосовуючи наближення ряду Тейлора першого порядку, ми маємо

$$\mathrm{L}(\mathrm{x}\ ', \mathrm{y}) = \mathrm{L}(\mathrm{x} + \delta, \mathrm{y}) \approx \mathrm{L}(\mathrm{x}, \mathrm{y}) + \nabla_x \mathrm{L}(\mathrm{x}, \mathrm{y})^T \cdot \delta, \tag{2.5}$$

Де $\delta = x' - x$ є протилежним збуренням, а $\nabla_x \mathrm{L}(\mathrm{x}, \mathrm{y})^T$ відноситься до градієнта функції втрат L щодо вхідних даних x, і його можна швидко обчислити за допомогою алгоритму зворотного поширення.

Цільова функція в (2.3) переписується так:

$$\min_{\delta} - \mathrm{L}(\mathrm{x}, \mathrm{y}) - \nabla_x \mathrm{L}(\mathrm{x}, \mathrm{y})^T \cdot \delta, \tag{2.6}$$

З обмеженням$\|\delta\|_\infty \leq \varepsilon$, де ми перетворюємо задачу максимізації на задачу мінімізації шляхом зміни знака двох компонентів. Зверніть увагу, що $L(x, y)$ і $\nabla_x \mathrm{L}(\mathrm{x}, \mathrm{y})^T$ є постійними та вже відомими, оскільки атака відбувається в налаштуваннях атаки білого ящика.[22]

Крім того, оскільки $0 \leq \|\delta\|_\infty \leq \varepsilon$, виходить:

$$\delta = \varepsilon \cdot \mathrm{sign}(\nabla_x \mathrm{L}(\mathrm{x}, \mathrm{y})), \tag{2.7}$$

Де функція $sign$ виводить знак свого вхідного значення.

Для налаштування цільової атаки зловмисник намагається мінімізувати функцію втрат $L(x,t)$, де t є цільовим класом. У цьому випадку це показується так:

$$\delta = -\varepsilon \cdot \mathrm{sign}(\nabla_x \mathrm{L}(\mathrm{x}, \mathrm{t})). \tag{2.8}$$

Формально ми можемо визначити FGSM з L_∞-нормою обмеженою величиною збурення наступним чином:[22]

Нехай $x \in R_n$ - законні вхідні дані, які правильно класифікуються як клас y класифікатором ML f. FGSM з L_∞-нормою обмеженою величиною збурення генерує суперечливий приклад $x' \in R_n$ шляхом максимізації функції втрат $L(x'$, y) з урахуванням обмеження $\| x' - x \|_\infty \leq \varepsilon$. Тоді,

$$x' = \begin{cases} \mathrm{x} + \varepsilon \cdot \mathrm{sign}(\nabla_x \mathrm{L}(\mathrm{x}, \mathrm{y})^T), \text{нецільовий}(\mathrm{untargeted}) \\ \mathrm{x} - \varepsilon \cdot \mathrm{sign}(\nabla_x \mathrm{L}(\mathrm{x}, \mathrm{y})^T), \text{цільовий}(\mathrm{targeted}) \text{ на t.} \end{cases} \tag{2.9}$$

Крім того, ми можемо легко узагальнити атаку на інші L_p - нормальні атаки. Наприклад, ми можемо використати нерівність Коші-Шварца, окремий випадок нерівності Гёльдера з $p = q = 2$, щоб знайти нижню межу цільової функції в задачі (2.6) за умови обмеження $\| x' - x \|_2 \leq \varepsilon$ та отримати збурення для ненаціленого FGSM для L_2-нормованого обмеження[22]

$$\delta = \frac{\varepsilon \nabla_x \mathrm{L}(\mathrm{x},\mathrm{y})}{\|\nabla_x \mathrm{L}(\mathrm{x},\mathrm{y})\|^2} \tag{2.10}$$

На практиці метод працює так:

- Обчислення втрат після прямого поширення
- Обчислення градієнту відносно пікселів зображення

- Невелике переміщення пікселів зображення в напрямку обчислених градієнтів, щоб максимально збільшити обчислені вище втрати.

Перший крок, обчислення втрат після прямого поширення, дуже поширений у проектах машинного навчання. Використовується функція втрат негативної ймовірності, щоб оцінити, наскільки прогноз моделі близький до фактичного класу.

Що не є поширеним, так це обчислення градієнтів відносно пікселів зображення. Коли справа доходить до навчання нейронних мереж, градієнти - це те, як визначається напрямок, у якому слід рухати вирахувані ваги, щоб зменшити значення втрати.[21]

Замість цього налаштовуються пікселі вхідного зображення в напрямку градієнтів, щоб максимізувати значення втрати. Під час навчання нейронних мереж найпопулярнішим способом визначення напрямку, у якому слід коригувати конкретну вагу в глибині мережі (тобто, градієнт функції втрат відносно цієї конкретної ваги), є зворотне поширення градієнтів від початок (вихідна частина) до ваги. Та сама концепція застосовна і в цьому методі. Ми зворотно поширюємо градієнти від вихідного шару до вхідного зображення.[22]

2.2.2 Атака Карліні і Вагнера

Атака Карліні та Вагнера (C&W) спрямована на пошук мінімально збуреного збурення. Зокрема, вони перетворили задачу оптимізації з обмеженнями в коробку в задачу оптимізації без обмежень, яку потім можна вирішити за допомогою стандартних алгоритмів оптимізації замість алгоритму L-BFGS. Карліні та Вагнер досліджували три різні методи, щоб позбутися прямокутного обмеження $x' \in [0, 1]^n$, прогнозованого градієнтного спуску, обрізаного градієнтного спуску та зміни змінних. Вони дійшли висновку, що метод зміни змінних є найефективнішим у швидкому

генеруванні змагальних прикладів[23]. Тобто вони ввели набір нових змінних $w_i \in R$, i = 1,2, . . . , n таке, що

$$x_i' = \frac{1}{2}(\tanh(w_i) + 1). \quad (2.11)$$

Незважаючи на те, що $x_i' \in [0, 1]$, wi може бути будь-яким дійсним числом, обмеження прямокутника усувається.

Крім того, обмеження $C(x') = t$ є дуже нелінійним. Карліні та Вагнер розглянули сім об'єктивних функцій-кандидатів g для заміни обмеження $C(x') = t$. Кожне g задовольняє умову, що $C(x') = t$ тоді і тільки тоді, коли $g(x') \leq 0$ (для цілеспрямованих атак) і $C(x') \neq y$ тоді і тільки тоді, коли $g(x') \leq 0$ (для нецільові атаки).[23] Наприклад, стандартна атака C&W $L2$ g визначена наступним чином

$$\mathrm{g(x')} = \begin{cases} max\left\{\max_{j \neq t} Z_j(x') - Z_t(x'), -\kappa\right\}, target\ on\ t \\ max\left\{Z_y(x') - \max_{j \neq t} Z_j(x'), -\kappa\right\}, untargeted \end{cases} \quad (2.12)$$

Де Z - логіти (ненормалізовані вихідні ймовірнісні прогнози моделі для кожного класу/вектор ймовірностей), а Z_j - j^{th} елемент логітів; $\kappa \geq 0$ — параметр, який контролює силу змагальних прикладів. Значення за умовчанням дорівнює нулю для експериментів. Однак збільшення κ може створити суперечливі приклади з вищим показником успіху передачі.[23]

Натомість Карліні та Вагнер розв'язали задачу оптимізації:

$$\min_{x'} \| \mathrm{x'} - \mathrm{x} \|_p + \mathrm{c} \cdot \mathrm{g(x')}, \quad (2.13)$$

Де $c > 0$ – параметр регуляризації, який контролює відносну важливість збурення L_p - норми над g. Якщо вибрано велике значення c, зловмисник генерує змагальний приклад, який знаходиться далі від базового прикладу, але неправильно класифікується з високою достовірністю, і навпаки. Карліні та Вагнер [23] використовували модифікований бінарний пошук, щоб знайти найменше значення c, для якого розв'язок x' задачі оптимізації (рис 2.13) задовольняє умову $g(x') \leq 0$. Альтернатива модифікованому бінарному

пошуку , рядковий пошук також можна використовувати для пошуку змагального прикладу з мінімальною L_p відстанню від x.[23]

Крім того, Карліні та Вагнер розглянули три типи атак: L_0-норма, L_2-норма та L_∞-норма. Формальне визначення атаки C&W представлено наступним чином:

Атака C&W генерує змагальний приклад $x' \in R^n$ шляхом вирішення такої проблеми оптимізації:

$$\min_{x'} \| \mathrm{x}' - \mathrm{x} \|_2^2 + c * g(x'), \tag{2.14}$$

Де $x_i' = \frac{1}{2}(\tanh(w_i) + 1)$, $c > 0$ є терміном регуляризації, і

$$\mathrm{g}(\mathrm{x}') = \begin{cases} max\left\{\max_{j \neq t} Z_j(x') - Z_t(x'), -\kappa\right\}, target\ on\ t \\ max\left\{Z_y(x') - \max_{j \neq t} Z_j(x'), -\kappa\right\}, untargeted \end{cases} \tag{2.15}$$

Також цей стандартний алгоритм градієнтного спуску може бути використаний для пошуку розв'язку наведеної вище проблеми мінімізації.

Карліні та Вагнер показали, що атака C&W є дуже потужною, і десять запропонованих стратегій захисту не можуть протистояти атакам C&W, створеним шляхом мінімізації специфічних для захисту функцій втрат. Крім того, атаку C&W також можна використовувати для оцінки ефективності потенційних стратегій захисту, оскільки це одна з найсильніших атак противника. [23]

2.2.3 Базовий ітераційний метод (Basic Iterative Method)

Базовий ітераційний метод(BIM), запропонований Куракіним та ін. є ітеративним уточненням FGSM. BIM використовує ітераційну лінеаризацію функції втрат, а не одноразову лінеаризацію в FGSM. Крім того, кілька менших, але фіксованих розмірів кроку α беруться замість одного більшого розміру кроку ε в напрямку знака градієнта.[24] Наприклад, у нецільовій атаці

з L_∞-нормою обмеженою величиною збурення, початкова точка $x^{(0)}$ встановлюється на початковий екземпляр x, і в кожній ітерації збурення $\delta^{(k)}$ і $x^{(k+1)}$ розраховується таким чином:

$$\delta^{(k)} = \alpha \times sign(\nabla_x L(x^{(k)}, y)) \quad (2.16)$$

$$x^{(k+1)} = clip_{x,\varepsilon}\{x^{(k)} + \delta^{(k)}\} \quad (2.17)$$

Де $\varepsilon > 0$ позначає розмір атаки, а $clip_{x,\varepsilon}(x')$ є оператором проекції, який проектує значення x' на перетин прямокутного обмеження x (наприклад, якщо x є зображенням, то обмеження прямокутника x може бути набором цілих значень від 0 до 255) і ϵ сусідньої кулі x. Наведена вище процедура гарантує, що створені змагальні приклади знаходяться в ϵ межах вхідних даних x. Використання зменшеної величини збурення α обмежує кількість активних пікселів атаки і, таким чином, не дозволяє простому детектору викидів виявити змагальні приклади.[24]

BIM-атака генерує обмежений L_∞-нормою змагальний приклад $x' \in R^n$ шляхом ітеративної максимізації функції втрат $L(x', y)$, підпорядкованої обмеженню $\| x' - x \|_\infty \leq \varepsilon$. Для ітерації k, $x^{(k+1)}$ обчислюється таким чином:

$$\delta^{(k)} = \alpha \times sign(\nabla_x L(x^{(k)}, y)) \quad (2.18)$$

$$x^{(k+1)} = clip_{x,\varepsilon}\{x^{(k)} + \delta^{(k)}\} \quad (2.19)$$

Де $x^{(0)} = x$, $\alpha < \varepsilon$ є меншою, але фіксованою величиною збурення в кожній ітерації, а кількість ітерацій вибирає користувач.[24]

Популярним варіантом BIM є атака прогнозованого градієнтного спуску (PGD), яка використовує рівномірно випадковий шум як ініціалізацію замість встановлення $x^{(0)} = x$. Випадкова ініціалізація використовується для дослідження простору введення. Пізніше Кроче та Хайн вдосконалили PGD, додавши термін імпульсу для оновлення градієнта та використовуючи концепцію розвідки проти експлуатації для оптимізації. Вони назвали цей підхід автоматичним прогнозованим градієнтним спуском (Auto-PGD) і показали, що Auto-PGD є більш ефективним, ніж PGD. Його додатково

вдосконалено за допомогою AutoAttack, сукупності різноманітних атак. Дійсно, AutoAttack поєднує три атаки білого ящика з однією атакою чорного ящика. Чотири компоненти атаки є двома розширеннями Auto-PGD атаки (одна максимізує втрату перехресної ентропії, а інша максимізує різницю втрат логістичного співвідношення) і дві існуючі допоміжні атаки, атаку швидкої адаптивної межі (FAB) і квадратну атаку, відповідно. Було показано, що використання ансамблю може підвищити ефективність атаки в порівнянні з кількома стратегіями захисту. Атака вважається успішною для кожного тестового прикладу, якщо принаймні один із чотирьох методів атаки знаходить змагальний приклад. [24]

Крім того, Auto-Attack може працювати повністю без попередньо визначених користувальницьких даних у наборах даних, моделях і нормах. Таким чином, він може забезпечити надійний, швидкий і безпараметричний інструмент оцінки, коли дослідник розробляє протилежний захист.[24]

2.2.4 DeepFool атаки

Глибокі нейронні мережі (DNN) досягли найвищої продуктивності для завдань класифікації зображень. Однак сучасні DNN мають уразливість до агресивних атак. Вони сприйнятливі до невеликих збурень, створених значущим чином, які не є випадковим шумом. Зі статті DeepFool: простий і точний метод обдурити глибокі нейронні мережі, автори Moosavi-Dezfooli та ін. представити ефективний метод обчислення мінімальних збурень, необхідних для того, щоб змусити класифікатор неправильно класифікувати зображення, який вони називають DeepFool.[25]

Автори DeepFool також представляють метод оцінки стійкості змагальних збурень, який можна використовувати для порівняння результатів з іншими методами змагальної атаки. Автори також проводять експеримент для підвищення надійності DNN шляхом тренування на змагальних образах.[25]

Кілька прикладів показано нижче на рисунку 2.6. У верхньому рядку оригінальне зображення визначається класифікатором як кит. У середньому рядку вихідне зображення кита було змінено, в результаті чого класифікатор визначив, що це черепаха. Збурене зображення є дуже тонким і непомітним для людського ока. Останнє зображує подібне спотворене зображення, але воно було спотворене методом швидкого градієнтного знака (FGSM), який є основним методом порівняння в статті. FGSM також зміг неправильно класифікувати вихідне зображення як черепаху, але кількість необхідних збурень є більшою, що можна побачити візуально.[25]

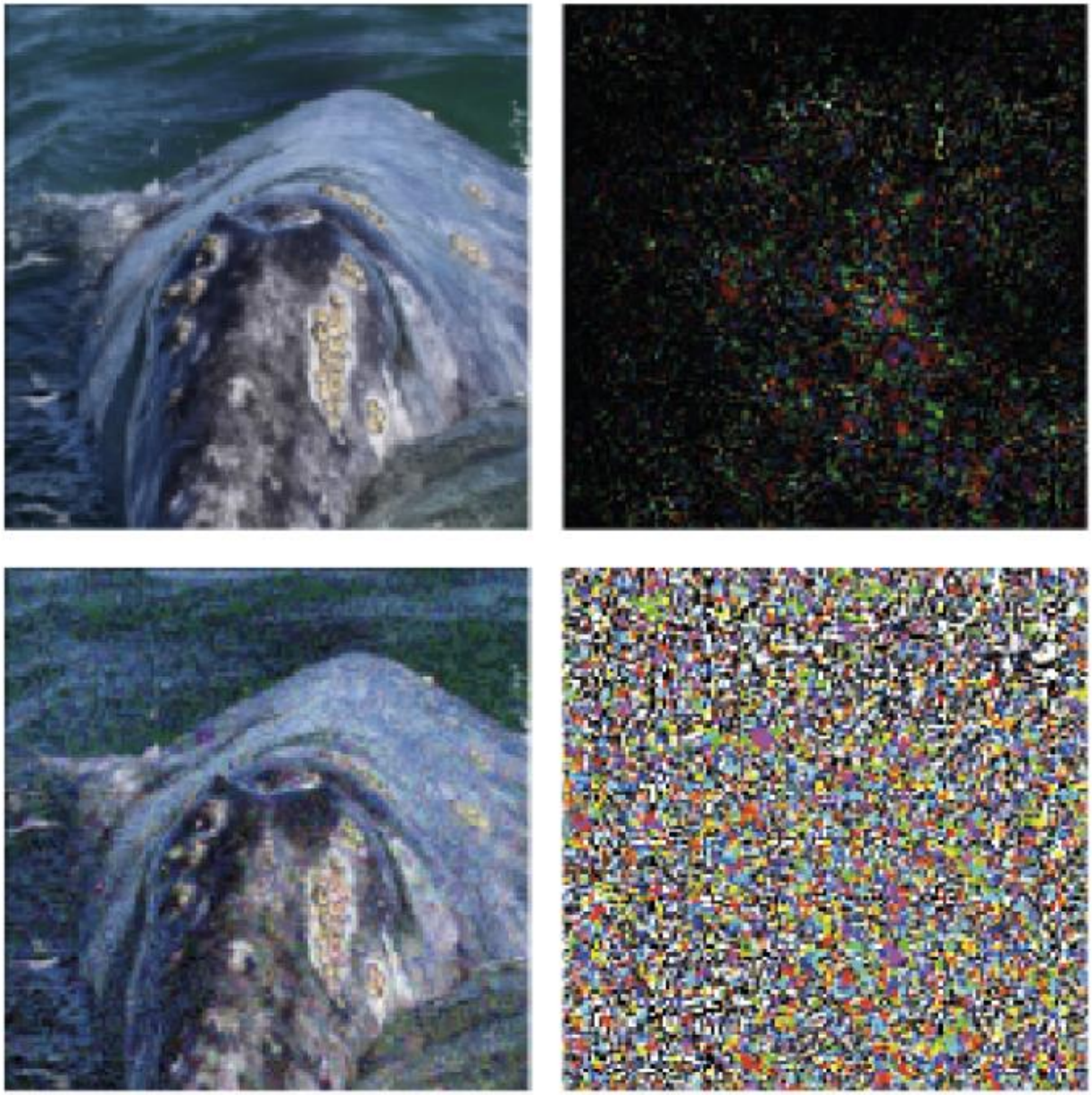

Рисунок 2.6 - Приклад змагальних пертурбацій (Deepfool проти FGSM)[26]

DeepFool для бінарної класифікації

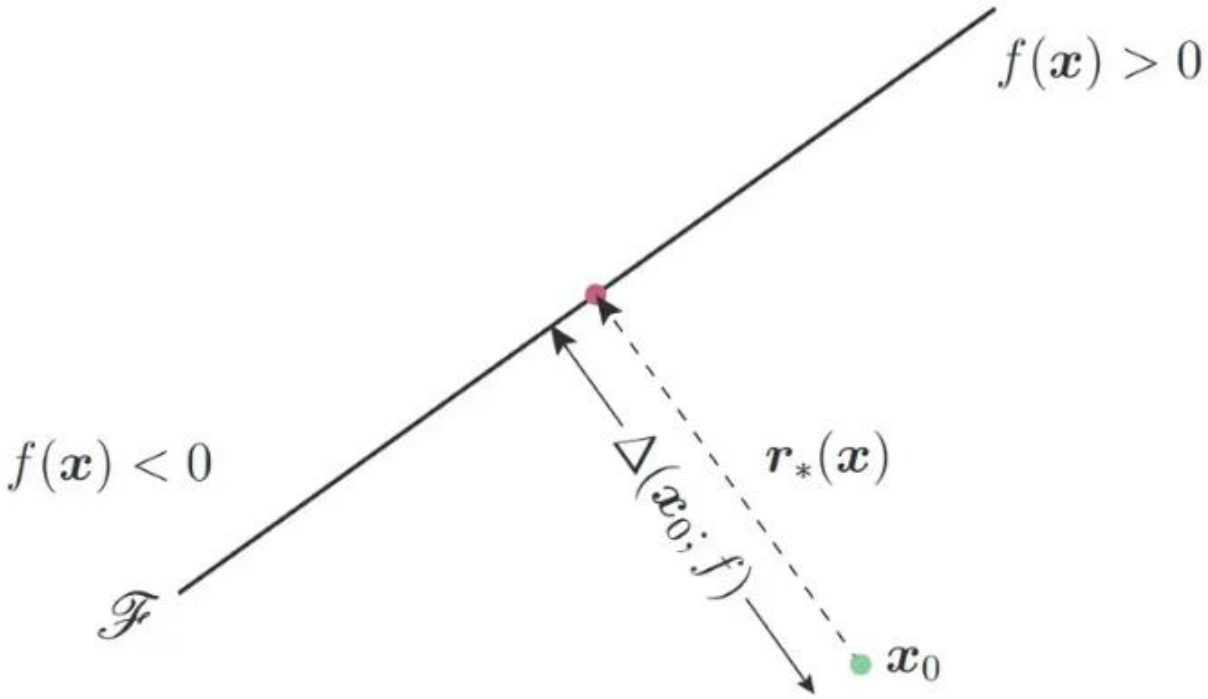

Рисунок 2.7 - змагальний приклад для бінарного класифікатора[26]

Перш ніж автори DeepFool пояснювали свій алгоритм для багатокласових класифікаторів, вони починають використовувати простий двійковий класифікатор, як показано на рис 2.7. З огляду на малюнок видно, що для отримання мінімального збурення для класифікатора f на неправильно класифікувати вхідне зображення x, збурення r*(x) має спроектувати вхідне зображення x ортогонально до гіперплощини бінарного класифікатора.[26]

$$r^*(x) = -\frac{f(x_0)}{\|w\|_2^2} w \tag{2.20}$$

Алгоритм DeepFool обчислює збурення r*(x), ділячи вихідний прогноз f(x_0) на L2-норму обчисленого градієнта ω функції втрат, даючи скаляр для збурення. Потім це множиться на одиничний вектор ω з використанням L2-норми і, нарешті, знак інвертується, щоб втрати класифікатора f збільшувалися, як показано в формулі 2.20. Використовуючи ідею лінійності

моделей, автори розробили цей простий алгоритм, але неправильна класифікація не гарантується, оскільки моделі є нелінійними за своєю природою. Щоб усунути цю проблему, алгоритм працює ітеративно та додає попереднє збурення до наступного збурення, яке виконується, доки не зміниться мітка або не буде досягнуто максимальної кількості ітерацій, як показано на рисунку 2.8. Крім того, збіжність може закінчитися близько до нуля, тому Параметр, який називається перевищенням η, використовується як скаляр (1 + η) для збурення r*(x), тому він перевищить нуль і змінить мітку класу.[27]

Algorithm 1 DeepFool for binary classifiers

input: Image $\boldsymbol{x}$, classifier f.
output: Perturbation $\hat{\boldsymbol{r}}$.
Initialize $\boldsymbol{x}_0 \leftarrow \boldsymbol{x}$, $i \leftarrow 0$.
while $\text{sign}(f(\boldsymbol{x}_i)) = \text{sign}(f(\boldsymbol{x}_0))$ **do**
    $\boldsymbol{r}_i \leftarrow -\frac{f(\boldsymbol{x}_i)}{\|\nabla f(x_i)\|_2^2} \nabla f(\boldsymbol{x}_i)$,
    $\boldsymbol{x}_{i+1} \leftarrow \boldsymbol{x}_i + \boldsymbol{r}_i$,
    $i \leftarrow i + 1$.
end while
return $\hat{\boldsymbol{r}} = \sum_i \boldsymbol{r}_i$.

Рисунок 2.8 - Псевдокод алгоритму DeepFool для бінарного класифікатора[27]

DeepFool для багатокласових класифікаторів

Розширюючи алгоритм DeepFool, реалізований для бінарних класифікаторів, алгоритм DeepFool для багатокласового класифікатора можна вважати кількома двійковими класифікаторами, як показано на рисунку 2.9. Границя рішення, як показано на рисунку 2.9, можна вважати багатогранником, який складається з перетин кількох гіперплощин. Мінімальне необхідне збурення буде від найближчої гіперплощини до x_0.

Враховуючи наявність кількох класів, втрати та зворотне розповсюдження повинні бути обчислені для кожної мітки класу, дозволеної функцією. При знаходженні мінімального збурення необхідно взяти різницю між обчисленими значеннями для кожної мітки та обчисленими значеннями для мітки вихідного прогнозу. Це показано в формулі 2.21, на якій індекс для мінімального збурення повертається як l(x_0).[27]

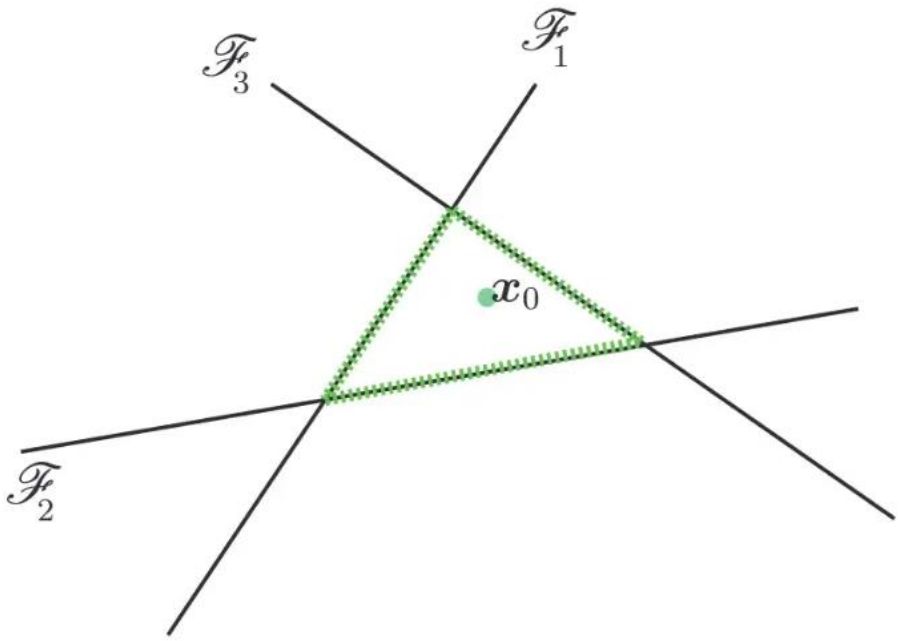

Рисунок 2.9 - Змагальний приклад для мультикласового класифікатора[26]

$$\hat{l}(x_0) = \underset{k \neq \hat{k}(x_0)}{\operatorname{argmin}} \frac{\left| f_k(x_0) - f_{\hat{k}(x_0)}(x_0) \right|}{\left\| w_k - w_{\hat{k}(x_0)} \right\|_2} \tag{2.21}$$

Після визначення індексу l(x_0) мінімальне збурення r*(x_0) обчислюється подібно до бінарного випадку, показаного на рисунку 3. Єдина відмінність від бінарного випадку полягає в тому, що він вимагає відмінностей вихідних даних класифікатора та градієнти виходів, а також взяти абсолютне значення виходу моделі, як показано в формулі 2.22.

$$\boldsymbol{r}_*(\boldsymbol{x}_0) = \frac{\left| f_{\hat{l}(\boldsymbol{x}_0)}(\boldsymbol{x}_0) - f_{\hat{k}(\boldsymbol{x}_0)}(\boldsymbol{x}_0) \right|}{\| \boldsymbol{w}_{\hat{l}(\boldsymbol{x}_0)} - \boldsymbol{w}_{\hat{k}(\boldsymbol{x}_0)} \|_2^2} (\boldsymbol{w}_{\hat{l}(\boldsymbol{x}_0)} - \boldsymbol{w}_{\hat{k}(\boldsymbol{x}_0)}) \tag{2.22}$$

Після обчислення збурення його буде додано до попереднього збурення r*(x_0), яке потім помножиться на скаляр перерегулювання (1 + η) і додасться до вхідного зображення x_i. Цей процес повторюється, доки не буде досягнуто максимальної кількості ітерацій або вихідний прогноз k_i із зображення x_i не дорівнюватиме оригінальному вихідному прогнозу k_0 з вихідного зображення x_0. Псевдокод для багатокласового алгоритму DeepFool показано на рисунку 2.10.[27]

Algorithm 2 DeepFool: multi-class case

input: Image $\boldsymbol{x}$, classifier f.
output: Perturbation $\hat{\boldsymbol{r}}$.

Initialize $\boldsymbol{x}_0 \leftarrow \boldsymbol{x}$, $i \leftarrow 0$.
while $\hat{k}(\boldsymbol{x}_i) = \hat{k}(\boldsymbol{x}_0)$ **do**
  for $k \neq \hat{k}(\boldsymbol{x}_0)$ **do**
    $\boldsymbol{w}'_k \leftarrow \nabla f_k(\boldsymbol{x}_i) - \nabla f_{\hat{k}(\boldsymbol{x}_0)}(\boldsymbol{x}_i)$
    $f'_k \leftarrow f_k(\boldsymbol{x}_i) - f_{\hat{k}(\boldsymbol{x}_0)}(\boldsymbol{x}_i)$
  end for
  $\hat{l} \leftarrow \arg\min_{k \neq \hat{k}(\boldsymbol{x}_0)} \frac{|f'_k|}{\|\boldsymbol{w}'_k\|_2}$
  $\boldsymbol{r}_i \leftarrow \frac{|f'_{\hat{l}}|}{\|\boldsymbol{w}'_{\hat{l}}\|_2^2} \boldsymbol{w}'_{\hat{l}}$
  $\boldsymbol{x}_{i+1} \leftarrow \boldsymbol{x}_i + \boldsymbol{r}_i$
  $i \leftarrow i + 1$
end while
return $\hat{\boldsymbol{r}} = \sum_i \boldsymbol{r}_i$

Рисунок 2.10 - Псевдокод алгоритму DeepFool для багатокласового класифікатора[27]

Швидкий огляд кожного кроку алгоритму на рисунку 2.10:

- Вхідними даними є зображення x і класифікатор f, який є моделлю.
- Вихід, який є збуренням

- [Пустий]
- 4.Ініціалізація збуреного зображення оригінальним зображенням і змінною циклу.
- Початок ітерації яка продовжується до тих пір, поки вихідна мітка та збурена мітка не будуть рівними.
- 6–9. Беруться n класів, які мали найбільшу ймовірність після початкового класу, і зберігається мінімальна різниця між вихідними градієнтами та градієнтами кожного з цих класів (w_{k}) і різницею в мітках (f_{k}) .
- 10. Внутрішній цикл зберігає мінімальні w_{k} і f_{k}, і за допомогою цього обчислюється найближча гіперплощина для вхідних даних x
- 11. Виконується обчислення мінімального вектору, який проектує x на найближчу гіперплощину, яка була обчислена в кроці 10.
- 12. Додається мінімальне збурення до зображення та перевіряється його класифікація.
- 13–14. Змінна циклу збільшена; Кінцева петля
- 15. Повернення загального збурення, яке є сумою всіх обчислених збурень.[27]

2.2.5 Прогнозований градієнтний спуск(projected gradient descent)

Атака прогнозованого градієнтного спуску(PGD) - це атака білого ящика, що означає, що зловмисник має доступ до градієнтів моделі, тобто зловмисник має копію ваг вибраної моделі. Ця модель загроз дає зловмиснику набагато більше можливостей, ніж атаки «чорної скриньки», оскільки вони можуть спеціально створити свою атаку, щоб обдурити вашу модель, не покладаючись на атаки передачі, які часто призводять до видимих для людини збурень. PGD можна вважати найбільш «повним» супротивником білої

скриньки, оскільки він скасовує будь-які обмеження на кількість часу та зусиль, які зловмисник може витратити на пошук найкращої атаки.[28]

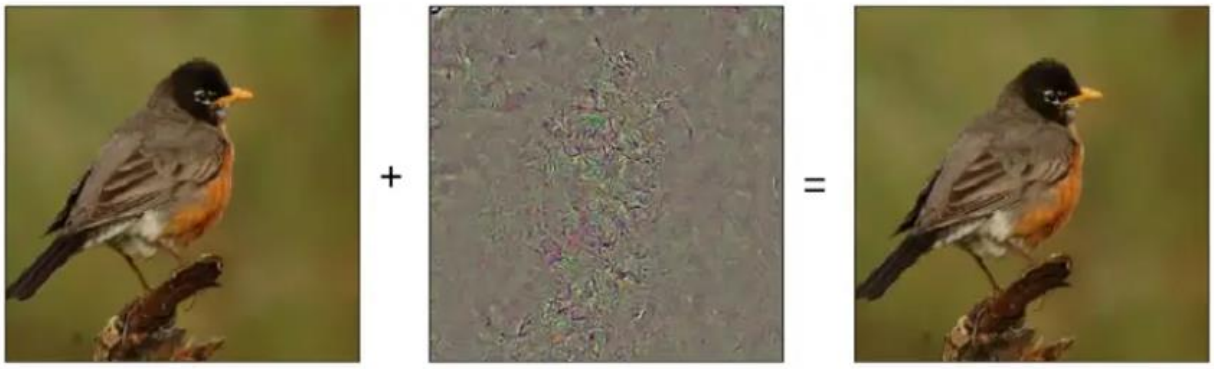

Рисунок 2.11 - Зліва: природне зображення. Посередині: змагальні збурення, виявлені під час атаки PGD на модель ResNet50, розмір збурення збільшено в 100 разів, щоб бути більш видимим. Праворуч: змагальний приклад.[28]

Ключ до розуміння PGD-атаки полягає в тому, щоб знайти приклад змагальності як проблему обмеженої оптимізації. PGD намагається знайти збурення, яке максимізує втрати моделі на певному вході, зберігаючи розмір збурення меншим за задану величину, яка називається епсилон. Це обмеження зазвичай виражається як L^2 або $L\infty$ норма збурення, і воно додається, щоб вміст змагального прикладу був таким же, як і незбуреного зразка - або навіть таким, щоб змагальний приклад непомітно відрізнявся для людей.

Реальні атаки, які можуть бути можливими за допомогою PGD:

- Зміна коду зловмисного програмного забезпечення, щоб обійти виявлення на основі машинного навчання
- Порушення зображення, щоб обійти заборону небажаної інформації на Tumblr
- Змушення стратегій ML трейдингу віддати всі свої гроші

Алгоритм PGD підсумовується за допомогою 5 кроків, наведених нижче, хоча зловмисник може вільно застосовувати будь-які вдосконалення оптимізації, такі як імпульс, Адам, багаторазові перезапуски тощо…

- Починається із випадкового збурення в кулі L^p навколо зразка
- Градієнтний крок у напрямку найбільших втрат
- Проектування збурення назад у кулю L^p, якщо необхідно
- Повтор кроків 2-3 до зближення

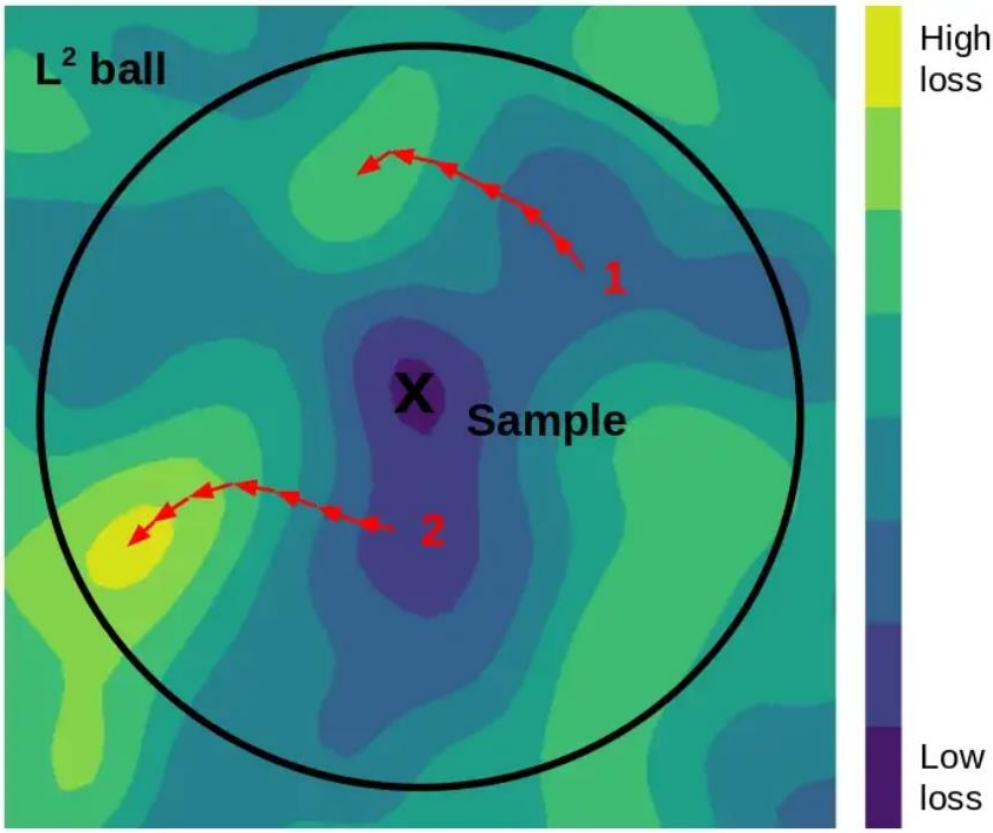

Рисунок 2.12 - Прогнозований градієнтний спуск із перезапуском. 2-й запуск знаходить приклад суперництва з великими втратами в межах м'яча L^2. Зразок знаходиться в області низьких втрат.[28]

«Проекція в кулю L^P» означає переміщення точки за межами деякого об'єму до найближчої точки всередині цього об'єму. У випадку норми L^2 у 2D це переміщення точки до відповідної найближчої точки на колі певного радіуса з центром у початку координат.[28]

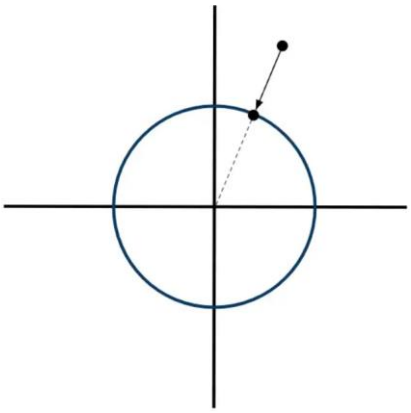

Рисунок 2.13 - Проектування точки назад на кулю L^2 у 2 вимірах[28]

2.2.6 Ітераційний метод імпульсу(momentum iterative fast gradient sign method(MIFGSM))

Метод імпульсу - це техніка для прискорення алгоритмів градієнтного спуску шляхом накопичення вектора швидкості в напрямку градієнта функції втрат через ітерації. Запам'ятовування попередніх градієнтів допомагає рухатися через вузькі долини, невеликі горби та погані локальні мінімуми чи максимуми. Метод імпульсу також показує свою ефективність у стохастичному градієнтному спуску для стабілізації оновлень. Застосовується ідею імпульсу, щоб генерувати суперечливі приклади та отримувати величезні переваги.[29] Для створення ненаціленого змагального прикладу x^* з реального прикладу x, який задовольняє обмеження норми L_∞, градієнтні підходи шукають змагальний приклад шляхом вирішення проблеми обмеженої оптимізації:

$$\underset{x^*}{\operatorname{argmax}} J(x^*, y), \qquad s.t. \|x^* - x\|_\infty \leq \epsilon \tag{2.23}$$

де ϵ - розмір змагального збурення. FGSM генерує змагальний приклад, застосовуючи знак градієнта до реального прикладу лише один раз за припущенням лінійності межі рішення навколо точки даних. Однак на практиці лінійне припущення може не виконуватися, коли викривлення велике, що робить змагальний приклад, створений FGSM, «недостатнім» для моделі, обмежуючи її здатність атакувати. Навпаки, ітераційний FGSM жадібно переміщує змагальний приклад у напрямку знака градієнта в кожній

ітерації. Таким чином, змагальний приклад може легко впасти в погані локальні максимуми та «переналаштувати» модель, що навряд чи перенесеться на інші моделі.[29]

Щоб подолати таку дилему, імпульс інтегрується в ітераційний FGSM з метою стабілізації напрямків оновлення та уникнення поганих локальних максимумів. Таким чином, метод, заснований на імпульсі, зберігає можливість передачі суперечливих прикладів при збільшенні ітерацій, і в той же час діє як сильний супротивник для моделей білого ящика, таких як ітераційний FGSM. Це полегшує компроміс між здатністю атаки та можливістю передачі, демонструючи сильні атаки чорної скриньки.

Метод ітераційного швидкого знакового градієнта імпульсу (MIFGSM) узагальнено в рисунку 2.14.[29]

Algorithm 1 MI-FGSM

Input: A classifier f with loss function J; a real example $\boldsymbol{x}$ and ground-truth label y;

Input: The size of perturbation ϵ; iterations T and decay factor μ.

Output: An adversarial example $\boldsymbol{x}^*$ with $\|\boldsymbol{x}^* - \boldsymbol{x}\|_\infty \leq \epsilon$.

$\alpha = \epsilon/T$;

$\boldsymbol{g}_0 = 0$; $\boldsymbol{x}_0^* = \boldsymbol{x}$;

for $t = 0$ to $T - 1$ **do**

  Input $\boldsymbol{x}_t^*$ to f and obtain the gradient $\nabla_{\boldsymbol{x}} J(\boldsymbol{x}_t^*, y)$;

  Update $\boldsymbol{g}_{t+1}$ by accumulating the velocity vector in the gradient direction as

$$\boldsymbol{g}_{t+1} = \mu \cdot \boldsymbol{g}_t + \frac{\nabla_{\boldsymbol{x}} J(\boldsymbol{x}_t^*, y)}{\|\nabla_{\boldsymbol{x}} J(\boldsymbol{x}_t^*, y)\|_1};$$

  Update $\boldsymbol{x}_{t+1}^*$ by applying the sign gradient as

$$\boldsymbol{x}_{t+1}^* = \boldsymbol{x}_t^* + \alpha \cdot \text{sign}(\boldsymbol{g}_{t+1});$$

end for

return $\boldsymbol{x}^* = \boldsymbol{x}_T^*$.

Рисунок 2.14 – Алгоритм MI-FGSM[29]

Висновки до розділу 2

Моделі для реалізації змагальних атак на системи розпізнавання образів

1. Багатошаровий персептрон - це модель нейронних мереж, яка працює як універсальний апроксиматор, тобто може апроксимувати будь-яку безперервну функцію.

2. Згорткова нейронна мережа - це алгоритм глибокого навчання, який може сприймати вхідне зображення, призначати важливість (вагові значення та зміщення) різним аспектам/об’єктам зображення та мати можливість відрізнити один від іншого. Попередня обробка, необхідна в ConvNet, набагато менша порівняно з іншими алгоритмами класифікації. У той час як у примітивних методах фільтри розробляються вручну, після достатнього навчання, ConvNets мають можливість вивчати ці фільтри/характеристики.

3. Мережа VGG19 – це згорткова нейронна марежа, яка має шість основних структур, кожна з яких в основному складається з кількох зв’язаних згорткових шарів і повнозв’язаних шарів. Розмір згорткового ядра становить 3*3, а вхідний розмір — 224*224*3. Кількість шарів, як правило, становить 16-19.

Змагальні атаки:

1. Метод швидкого градієнтного знака (FGSM) - фокусується на ефективному пошуку змагального збурення, обмеженого L_∞-нормою. Під час навчання моделі ML задана функція втрат мінімізується, щоб знайти оптимальний набір параметрів Θ для класифікатора C так, щоб C правильно класифікував більшість навчальних даних.

2. Атака Карліні та Вагнера (C&W) - спрямована на пошук мінімально збуреного збурення. Зокрема, вони перетворили задачу оптимізації з обмеженнями в коробку в задачу оптимізації без обмежень, яку потім можна вирішити за допомогою стандартних алгоритмів оптимізації.

3. Базовий ітераційний метод(BIM) - запропонований Куракіним та ін. є ітеративним уточненням FGSM. BIM використовує ітераційну лінеаризацію функції втрат, а не одноразову лінеаризацію в FGSM.

4. DeepFool - ефективний метод обчислення мінімальних збурень, необхідних для того, щоб змусити класифікатор неправильно класифікувати зображення.

5. Атака прогнозованого градієнтного спуску (PGD) - це атака білого ящика, що означає, що зловмисник має доступ до градієнтів моделі, тобто зловмисник має копію ваг вибраної моделі.

6. Метод імпульсу(MI-FGSM) - це техніка для прискорення алгоритмів градієнтного спуску шляхом накопичення вектора швидкості в напрямку градієнта функції втрат через ітерації.

РОЗДІЛ 3 РЕАЛІЗАЦІЯ АЛГОРИТМУ ТА ПОБУДОВА МОДЕЛЕЙ ЗМАГАЛЬНИХ АТАК НА СИСТЕМИ РОЗПІЗНАВАННЯ ОБРАЗІВ

3.1 Реалізація алгоритмів розпізнавання образів

Цей алгоритм був написаний під час російського вторгнення в Україну. Конфлікт висвітлив використання супутникових зображень журналістами, правозахисними організаціями та аналітиками розвідки з відкритих джерел. Зображення високої роздільної здатності створюються операторами приватних і державних супутників майже в режимі реального часу, і інформація використовується для відстеження переміщень військ, перевірки атак у недоступних районах, оцінки пошкоджень інфраструктури та документування можливих військових злочинів.

Оскільки супутникові зображення стали дуже важливими в сучасних конфліктах, була досліджена можливість використання моделі глибокого навчання для ідентифікації та точної класифікації різних типів військових транспортних засобів, які можуть бути знайдені на полі бою. Дедалі більше використання супутникових зображень для відстеження конфліктів призвело до відповідного збільшення величезного обсягу створюваних зображень, і завдяки цьому рішення, розроблене для автоматичної ідентифікації різних транспортних засобів, може допомогти спостерігачам переглядати тисячі зображень і швидше аналізувати ті, що містять предмети інтересу.

Для цього проекту використовувався набір даних для виявлення та розпізнавання рухомих і стаціонарних цілей (MSTAR), створений Агентством передових оборонних дослідницьких проектів США (DARPA) і Дослідницькою лабораторією ВПС США з 1995 по 1997 рік. Набір даних доступний для загального використання на веб-сайті науково-дослідної лабораторії ВПС. Дані містять такі 9 класів:

- 2C1 Гвоздика — Самохідна артилерійська установка
- ЗСУ-23–4 Шилка — зенітна самохідна установка

- БРДМ-2 — броньована машина-розвідник-амфібія
- БТР-60 — бронетранспортер
- D7 — Бульдозер Caterpillar
- ЗІЛ-131 — Військово-вантажний автомобіль
- Т-62 — Основний бойовий танк
- Т-72 — основний бойовий танк 2-го покоління
- SLICY — структура, що діє як «наземний елемент» (а не транспортний засіб)

Рисунок 3.1 - Класи транспортних засобів (ліворуч), супутникові знімки тих же класів (праворуч).

Зображення кожного типу техніки були зроблені за допомогою радара з синтетичною апертурою (SAR). Це форма супутникових зображень, яка має перевагу пронизливих атмосферних умов, таких як хмари. Як видно з наведених нижче прикладів, не зовсім зрозуміло, який технічний засіб містить зображення.

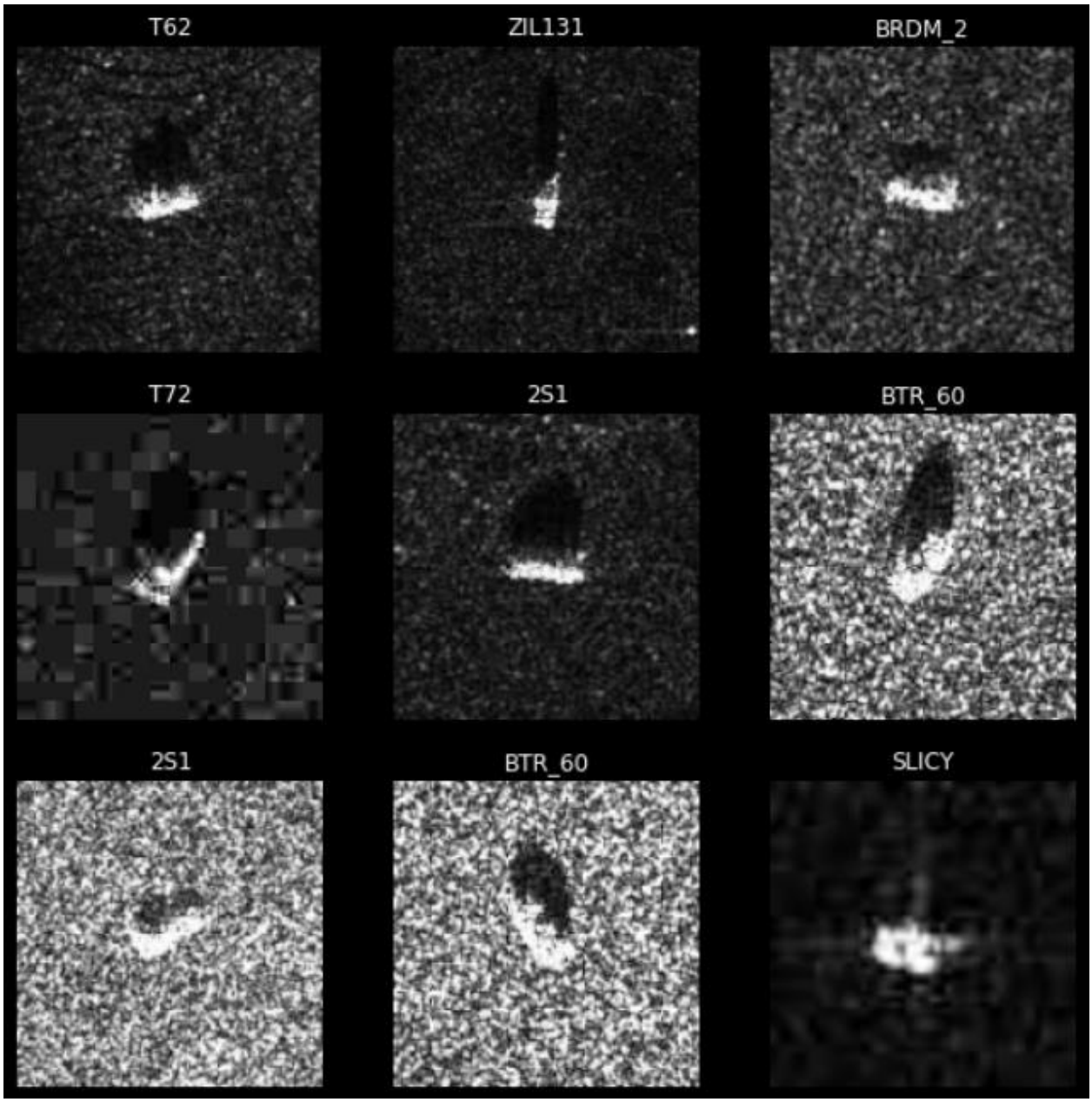

Рисунок 3.2 - Зображення SAR

Потрібно було побачити, наскільки точною може бути модель, тому були додані класи Т-62/Т-72. Було зроблено припущення, що вони виглядатимуть дуже схожими на супутникових зображеннях, і потрібно було оцінити, чи зможе модель виявити тонкі відмінності між ними.

Подібним чином потрібно було переконатися, що модель добре справляється з небойовими цілями, такими як бульдозер D7 і SLICY. Якби модель часто неправильно класифікувала ці класи, вона не мала б великої практичної користі та могла б викликати більше проблем, ніж вирішити.

Для того, щоб визначити точність класифікатора зображень, були використані такі метрики:

Категорична точність(Categorical Accuracy) - обчислює відсоток прогнозованих значень (yPred), які збігаються з фактичними значеннями (yTrue) для одноразових міток.

Визначається індекс, за яким виникає максимальне значення, використовуючи argmax(). Якщо воно однакове для yPred і yTrue, воно вважається точним. Потім обчислюється категорична точність, розділивши кількість точно передбачених записів на загальну кількість записів.

Оскільки категорійна точність шукає індекс максимального значення, yPred може бути логітом або ймовірністю прогнозів.[30]

Середньозважена оцінка F2 - є прикладом Fbeta-міри з бета-значенням 2,0. Це зменшує важливість точності(Precision) та підвищує важливість повноти(Recall). Якщо максимізація точності мінімізує хибно-позитивні результати, а максимізація повноти мінімізує хибно-негативні результати, тоді показник F2 приділяє більше уваги мінімізації хибно-негативних результатів, ніж мінімізації хибно-позитивних результатів.[31]

Показник F2 обчислюється наступним чином:

- F2 = ((1 + 2^2) * Precision * Recall) / (2^2 * Precision + Recall)
- F2 = (5 * Precision * Recall) / (4 * Precision + Recall)

Коефіцієнт кореляції Метью - також скорочений як MCC, був винайдений Браяном Метьюзом у 1975 році. MCC - це статистичний інструмент, який використовується для оцінки моделі. Його завдання полягає в оцінці або вимірюванні різниці між прогнозованими та фактичними значеннями та еквівалентно статистиці хі-квадрат для таблиці непередбачених обставин 2 x 2.

MCC - це найкращий показник класифікації з одним значенням, який допомагає узагальнити матрицю плутанини або матрицю помилок. Матриця плутанини має чотири елементи:[32]

- True positives (TP)

- True negatives (TN)
- False positives (FP)
- False negatives (FN)

І розраховується за формулою:

$$MCC = \frac{TN \times TP - FN \times FP}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

Рисунок 3.3 - Коефіцієнт кореляції Метью[32]

Для класифікації зображень було побудовано декілька моделей, перша з яких це проста багатошарова модель персептрона (MLP). Кожен із трьох шарів був налаштований на 100 вузлів і було використано функцію активації Rectified Linear Unit (ReLU).

Були отримані такі результати

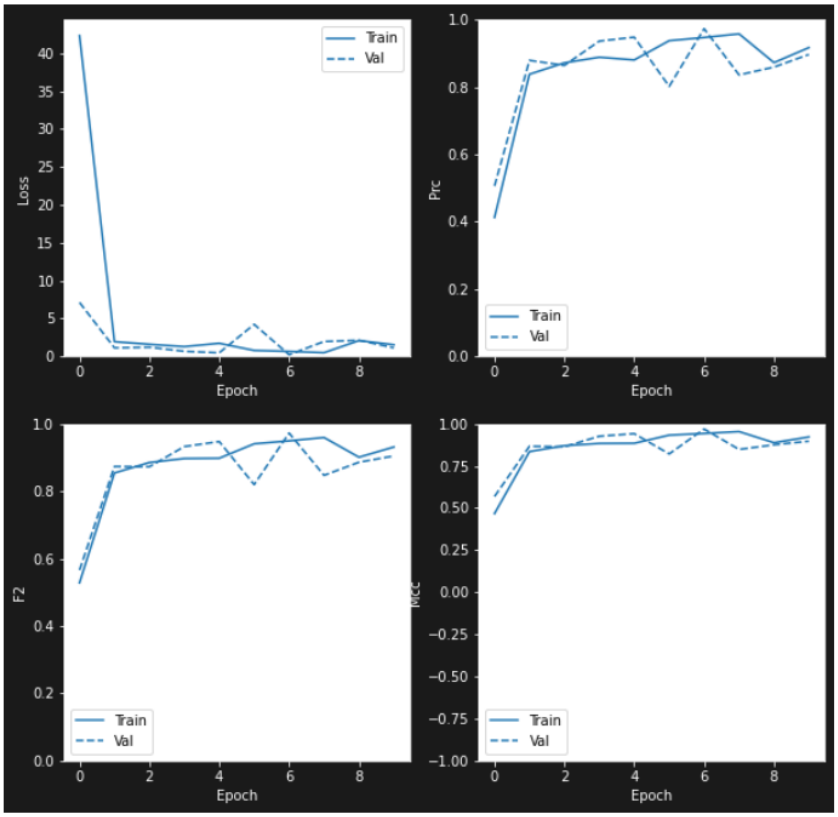

Рисунок 3.4 - Графік 1 моделі

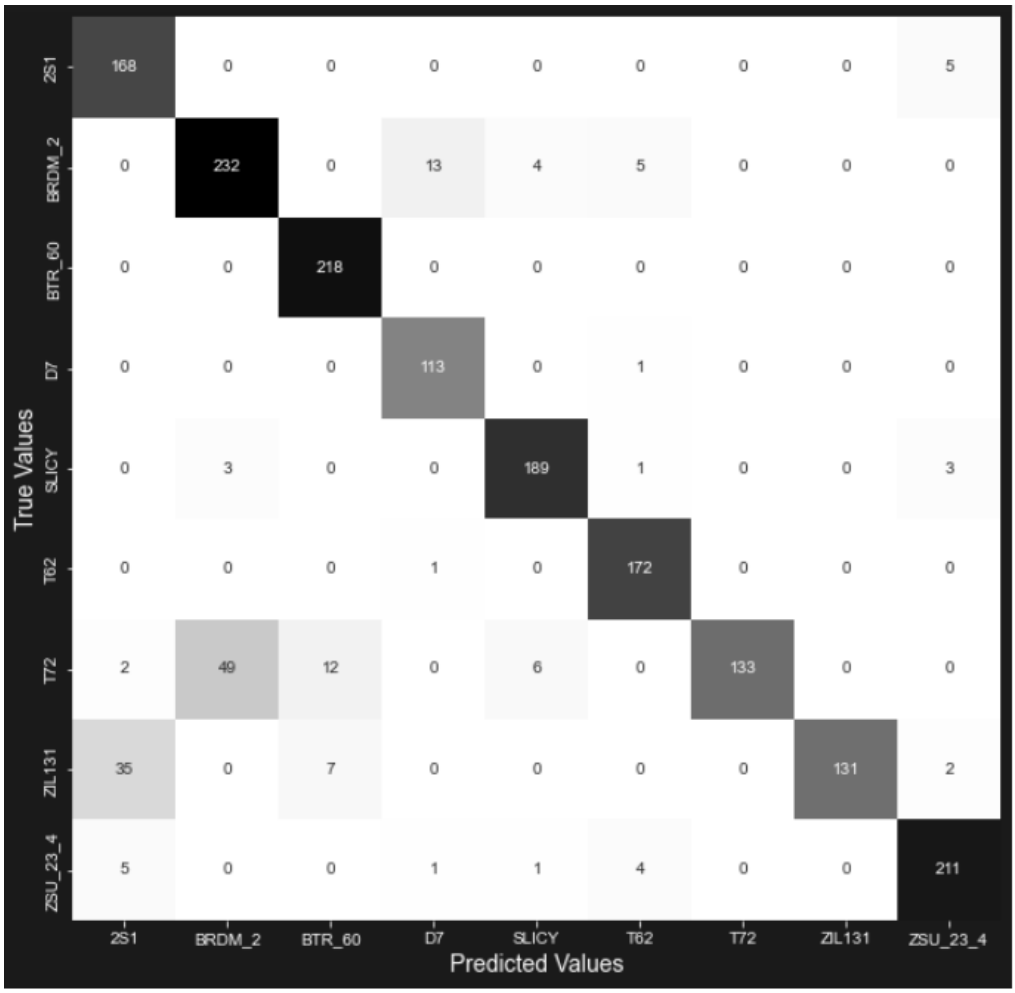

Рисунок 3.5 - Confusion matrix 1 моделі

```
{'loss': 1.4372074604034424,
 'categorical_accuracy': 0.8990384340286255,
 'MCC': 0.8889861106872559,
 'F2': 0.8948849439620972,
 'auc': 0.9638181328773499,
 'prc': 0.8897144794464111}
```

Рисунок 3.6 - Результати першої моделі в різних метриках

Точність моделі, з огляду на рисунок 3.6 виявилась 0.89, тому моделі для розпізнавання потрібно покращити.

Наступною моделлю була згорткова нейронна мережа (CNN). CNN особливо добре підходять для виявлення об’єктів на зображеннях, оскільки вони використовують ряд фільтрів (визначених у параметрах шару) для ітеративного сканування зображення та вилучення найбільш помітних

функцій. Протягом послідовних ітерацій CNN переходять від пошуку базових форм, таких як лінії та краї, до пошуку форм вищого порядку, таких як транспортні засоби чи тварини.

Отримані результати:

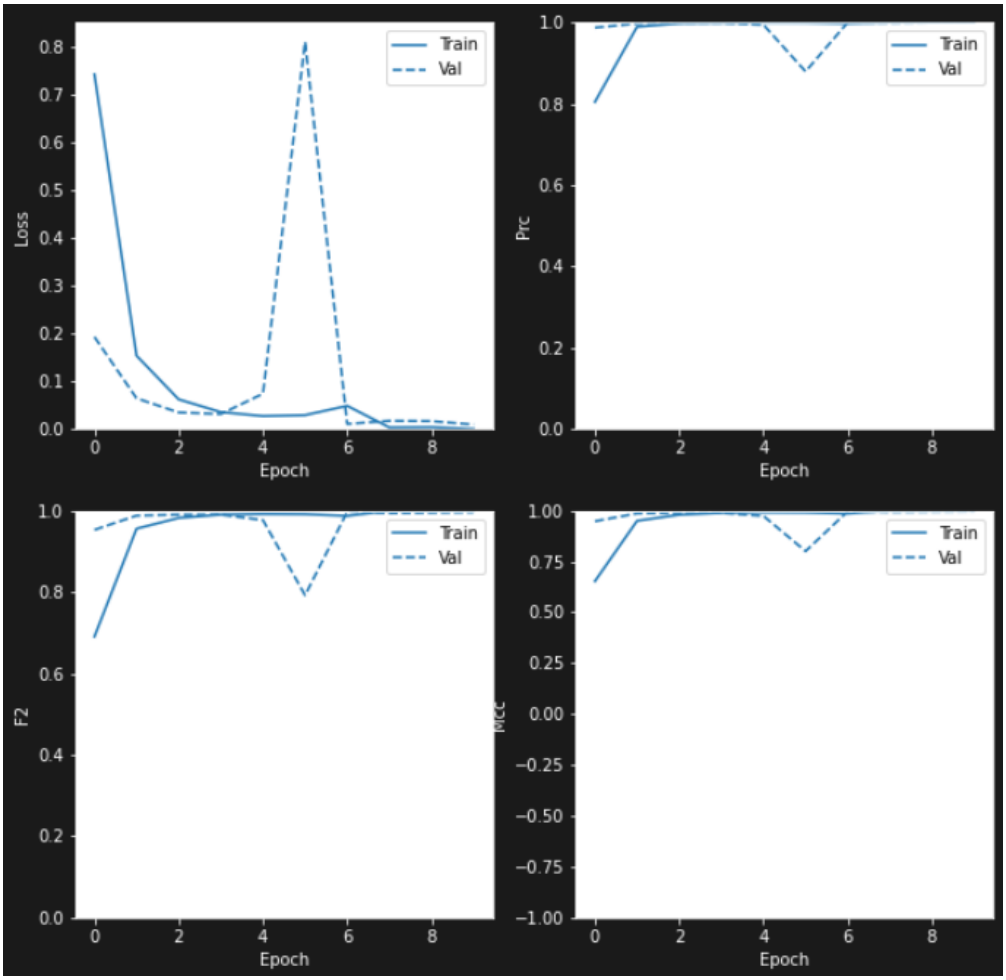

Рисунок 3.7 - Графік 2 моделі

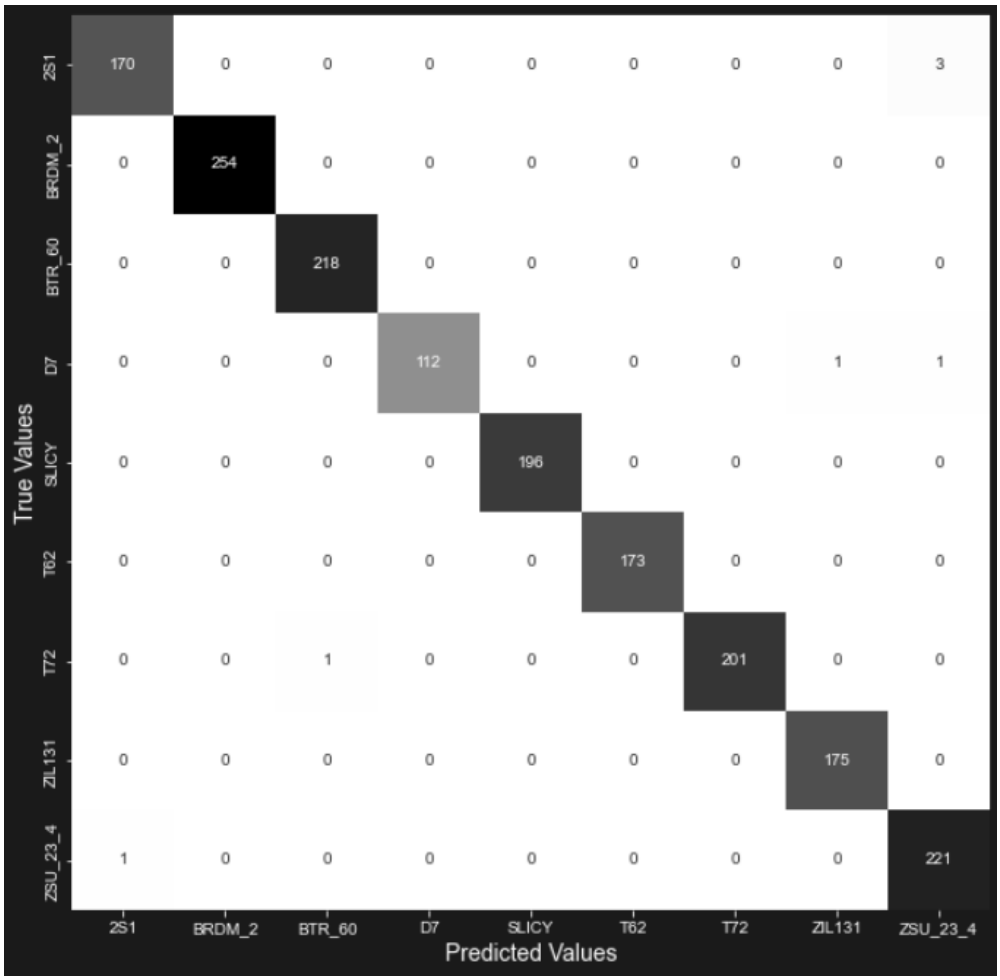

Рисунок 3.8 - Confusion matrix 2 моделі

```
{'loss': 0.0010492849396541715,
 'categorical_accuracy': 1.0,
 'MCC': 1.0,
 'F2': 1.0,
 'auc': 1.0,
 'prc': 1.0}
```

Рисунок 3.9 - Результати другої моделі в різних метриках

Отже, згорткові нейронні мережі працюють краще, ніж багатошаровий персептрон, і модель не була ще налаштована, щоб працювати якомога ефективніше. Розширимо її та побачимо, чи вийде отримати ще кращий результат.

Під час останньої ітерації було додано деяке розширення даних і шари відсіву, щоб уникнути перенавчання моделі. Також було додано два параметри

зворотного виклику: ранню зупинку та зниження швидкості навчання(працює якщо якість начання перестає рости, або росте на число, менше ніж 2^-10).

Як випливає з назви, рання зупинка призведе до раннього припинення навчання моделі, якщо відстежувані показники не покращилися протягом певної кількості епох. Тим часом зниження швидкості навчання(ReduceLROnPlateau) призведе до зниження швидкості навчання моделі, якщо не буде покращення протягом певної кількості епох. Важливо встановити, щоб швидкість навчання була нижчою за швидкість ранньої зупинки, щоб модель мала можливість спробувати нову швидкість перед зупинкою. Було використано 10 на ранню зупинку(.EarlyStopping) та 3 на зниження швидкості навчання(ReduceLROnPlateau).

Було зроблено 40 ітерацій, і отриманий результат виглядає так:

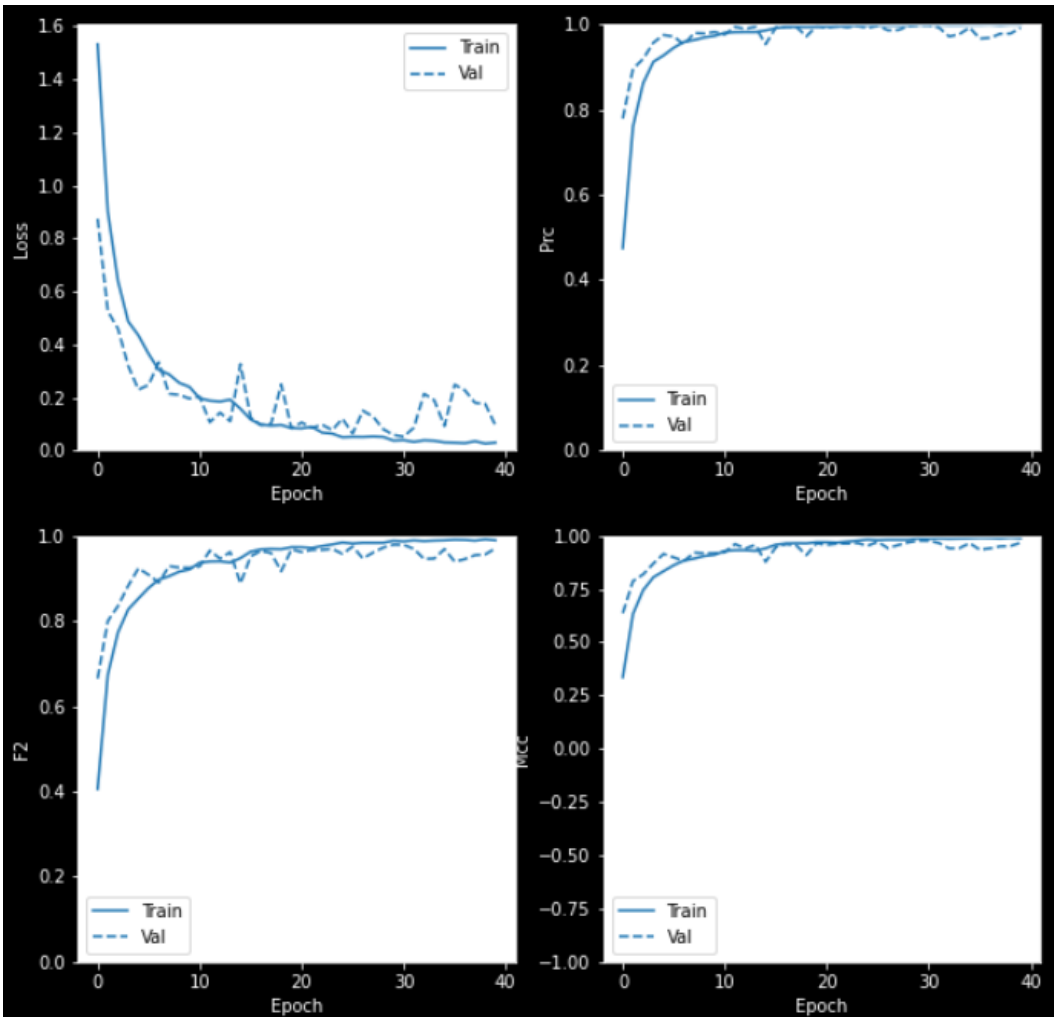

Рисунок 3.10 - Графік 3 моделі

Дуже близько за продуктивністю до першої моделі CNN і, ймовірно, трохи менш точні через випадання та доповнення даних, які були включені. Матриця помилок для даних перевірки:

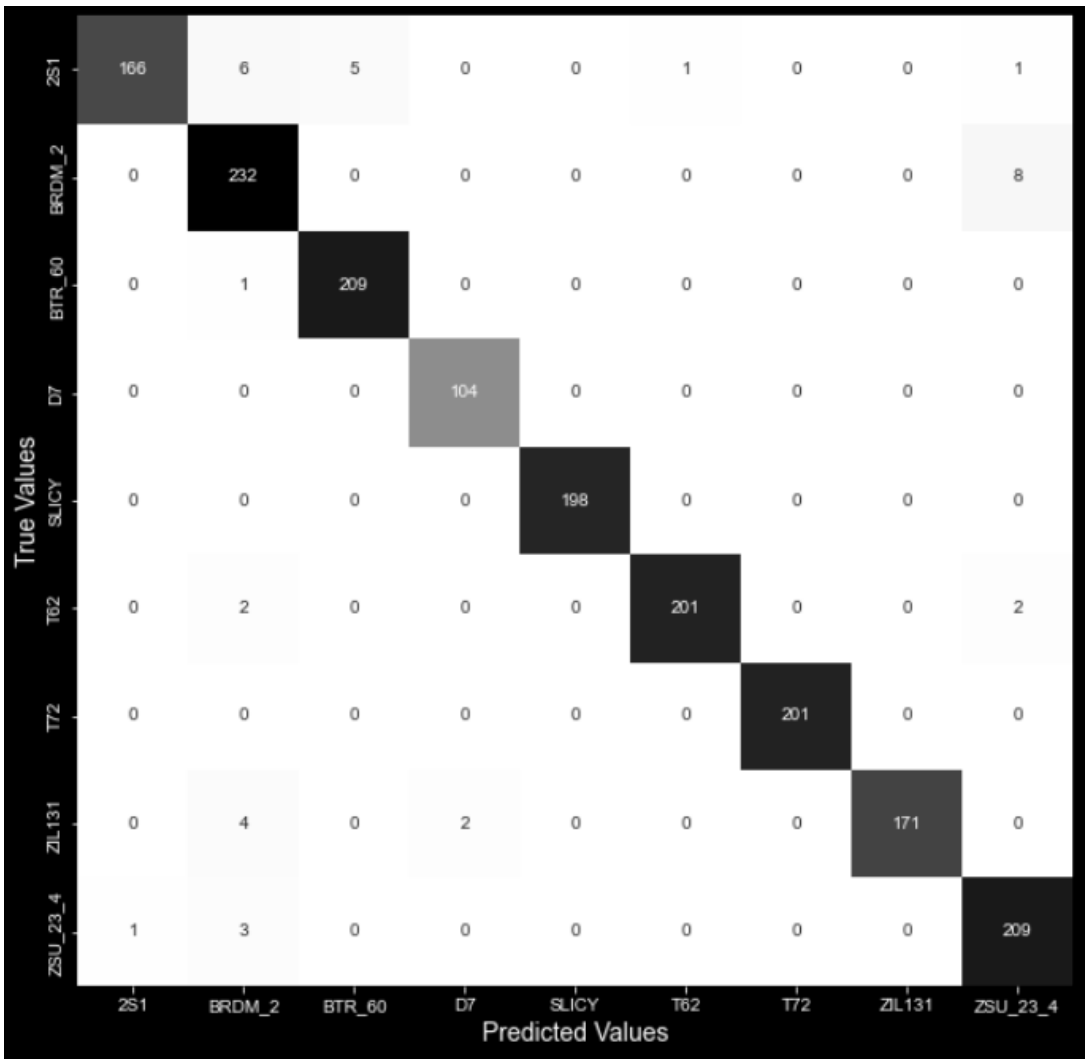

Рисунок 3.11 - Confusion matrix 3 моделі

Остаточні показники наших даних:

```
{'loss': 0.06638450920581818,
 'categorical_accuracy': 0.9783653616905212,
 'MCC': 0.9755039215087891,
 'F2': 0.978276789188385,
 'auc': 0.9985445737838745,
 'prc': 0.9964696168899536}
```

Рисунок 3.12 - Результати третої моделі в різних метриках

Отже третя модель показала результат в 0.98 точності, що є задовільним результатом для подальшого використання. Далі для останньої моделі були реалізовані змагальні атаки, мета яких була схибити модель, щоб вона невірно розпізнавала данні.

3.2 Реалізація змагальних атак на отримані моделі

Для реалізації змагальних атак на отримані моделі було взято зображення з датасету, який використовувався для навчання моделі, а саме датасету MSTAR з супутниковими зображеннями військової техніки. Це зображення танку Т-62, яке виглядає так:

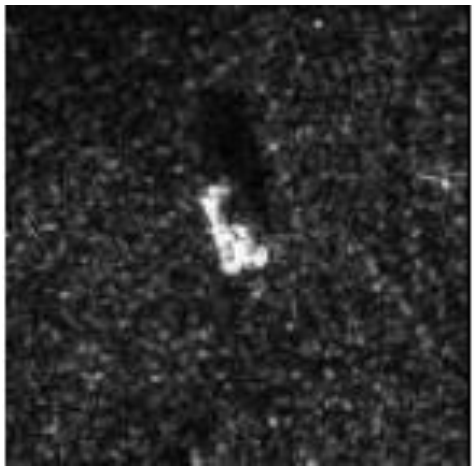

Рисунок 3.13 - Використане зображення танку Т-62

Для початку це зображення запустили в розпізнавальній моделі, і був отриманий такий результат:

```
'This image most likely belongs to T62 with a 99.99 percent confidence.'
```

Рисунок 3.14 - Результат розпізнавання

Першою атакою був реалізований Fast Gradient Sigh method(FGSM), формула якого виглядає так:

$$adv\_x = x + \epsilon * \text{sign}(\nabla_x J(\theta, x, y))$$

Рисунок 3.15 - Формула FGSM

Де

- Adv_x - змагальне зображення
- X - вхідне зображення
- ϵ - невелике значення, на яке множаться градієнти зі знаком, щоб збурення були достатньо малими для людського зору, але достатньо сильними для комп'ютера
- θ - модель нейронної мережі
- J - функція втрат

Для цієї атаки був вибраний епсілон = 0.1, порядок норми був заданий як представлення нескінченності з плаваючою комою IEEE 754, максимальне і мінімальне плаваюче значення для змагальних прикладних компонентів було установлене як -1 і 100.

Отриманий результат:

```
This image belongs to 2S1 with a 1.40 percent confidence.
This image belongs to BRDM_2 with a 77.24 percent confidence.
This image belongs to BTR_60 with a 0.49 percent confidence.
This image belongs to D7 with a 0.00 percent confidence.
This image belongs to SLICY with a 0.09 percent confidence.
This image belongs to T62 with a 19.95 percent confidence.
This image belongs to T72 with a 0.01 percent confidence.
This image belongs to ZIL131 with a 0.00 percent confidence.
This image belongs to ZSU_23_4 with a 0.81 percent confidence.
```

Рисунок 3.16 - Результат FGSM

Як можна побачити на рисунку 3.16, зображення було класифіковано як брдм-2 з ймовірністю 77.24%, а як Т62 з ймовірністю 19.95 відсотків. Також з незначними значеннями ймовірностей малюнок був позначений як 2С1, ЗСУ-23-4 та БТР-60.

Зображення Т62, на яке був накладений шум, і яке потім було розпізнано з використанням побудованої моделі:

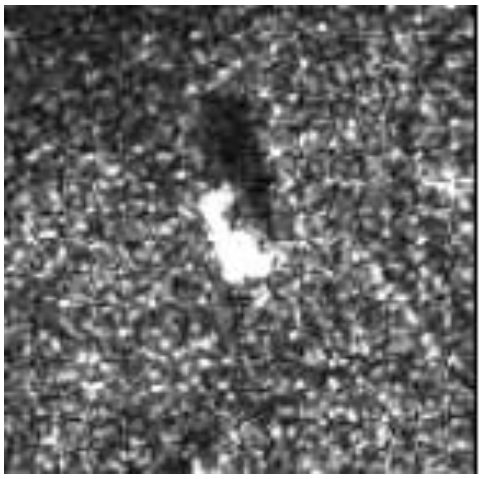

Рисунок 3.17 - Зображення T62 з шумом при використанні FGSM

Вигляд шуму, який накладається на зображення:

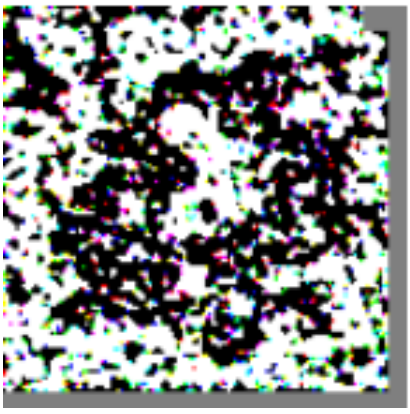

Рисунок 3.18 - Шум, який накладається на зображення

З рисунка 3.17 і результатів розпізнавання можна побачити, що хоча зображення і було неправильно класифіковане як брдм-2, шум, який був накладений на картинку доволі помітний, тобто прийшлось доволі сильно зіпсувати початкове фото, щоб змагальна атака вдалась. Тому потрібно використати інші варіанти алгоритмів атаки для вибору найкращого.

Другою атакою був реалізований прогнозований градієнтний спуск(projected gradient descent або PGD) – алгоритм, який намагається знайти збурення, яке максимізує втрати моделі на певному вході, зберігаючи розмір збурення меншим за епсилон. Це обмеження зазвичай виражається як L^2 або $L\infty$ норма збурення, і воно додається, щоб вміст змагального прикладу був

таким же, як і незбуреного зразка - або навіть таким, щоб змагальний приклад не відрізнявся для людського ока.

Для цієї атаки був вибраний епсілон = 0.1 з кроком 0.05 для кожної ітерації, кількість ітерацій =50, порядок норми був заданий як представлення нескінченності з плаваючою комою IEEE 754, максимальне і мінімальне плаваюче значення для змагальних прикладних компонентів було установлене як -1 і 100.

Отриманий результат:

```
This image belongs to 2S1 with a 0.59 percent confidence.
This image belongs to BRDM_2 with a 90.60 percent confidence.
This image belongs to BTR_60 with a 0.32 percent confidence.
This image belongs to D7 with a 0.00 percent confidence.
This image belongs to SLICY with a 0.06 percent confidence.
This image belongs to T62 with a 8.07 percent confidence.
This image belongs to T72 with a 0.01 percent confidence.
This image belongs to ZIL131 with a 0.00 percent confidence.
This image belongs to ZSU_23_4 with a 0.36 percent confidence.
```

Рисунок 3.19 - Результат PGD

Як можна побачити на рисунку 3.19, зображення було класифіковано як брдм-2 з ймовірністю 90.6%, а як Т62 з ймовірністю 8.07%. Також з незначними значеннями ймовірностей малюнок був позначений як 2С1, ЗСУ-23-4 та БТР-60.

Зображення Т62, на яке був накладений шум, і яке потім було розпізнано з використанням побудованої моделі:

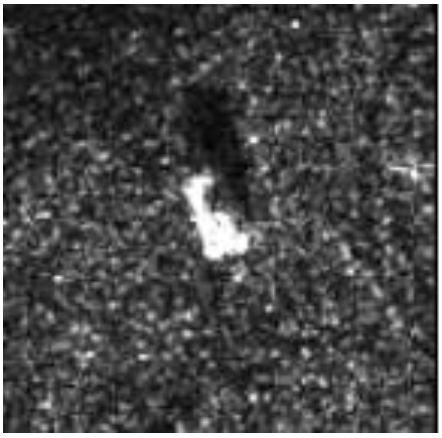

Рисунок 3.20 - Зображення Т62 з шумом при використанні PGD

Як можна побачити з рисунка 3.19, зображення було класифіковане як T62 з ймовірністю лише 8.07%, при чому шум на зображені став меншим. Також на відміну від використання FGSM при збільшені епсілон, і зменшені кількості накладеного шуму на зображення можна отримати такий результат:

```
This image belongs to 2S1 with a 88.80 percent confidence.
This image belongs to BRDM_2 with a 0.03 percent confidence.
This image belongs to BTR_60 with a 0.02 percent confidence.
This image belongs to D7 with a 0.00 percent confidence.
This image belongs to SLICY with a 0.00 percent confidence.
This image belongs to T62 with a 10.22 percent confidence.
This image belongs to T72 with a 0.02 percent confidence.
This image belongs to ZIL131 with a 0.80 percent confidence.
This image belongs to ZSU_23_4 with a 0.10 percent confidence.
```

Рисунок 3.21 - Результат PGD при збільшені епсілон

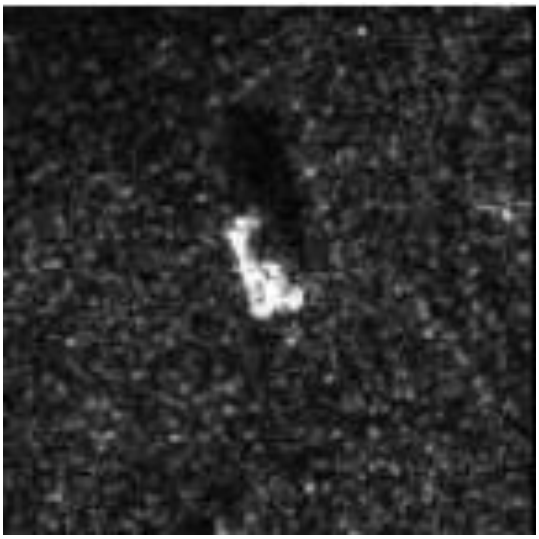

Рисунок 3.22 - Зображення T62 при використанні PGD з збільшеним епсілон

З рисунків 3.21 і 3.22 можна побачити що тепер зображення розпізнається як 2S1 з високою ймовірністю, при чому саме початкове зображення слабо змінилось порівнюючи з початковим. Отже, цей алгоритм є ефективнішим ніж FGSM.

Третьою атакою був реалізований алгоритм ітераційного методу імпульсу(momentum iterative method(mim)) - техніка для прискорення

алгоритмів градієнтного спуску шляхом накопичення вектора швидкості в напрямку градієнта функції втрат через ітерації.

Отриманий результат:

```
This image belongs to 2S1 with a 0.62 percent confidence.
This image belongs to BRDM_2 with a 16.71 percent confidence.
This image belongs to BTR_60 with a 0.06 percent confidence.
This image belongs to D7 with a 0.00 percent confidence.
This image belongs to SLICY with a 0.02 percent confidence.
This image belongs to T62 with a 81.80 percent confidence.
This image belongs to T72 with a 0.01 percent confidence.
This image belongs to ZIL131 with a 0.00 percent confidence.
This image belongs to ZSU_23_4 with a 0.78 percent confidence.
```

Рисунок 3.23 - Результат MIM

Як можна побачити на рисунку 3.23, зображення було класифіковано як брдм-2 з ймовірністю 16.71%, а як Т62 з ймовірністю 81.8%. Також з незначними значеннями ймовірностей малюнок був позначений як 2С1, ЗСУ-23-4 та БТР-60.

Зображення Т62, на яке був накладений шум, і яке потім було розпізнано з використанням побудованої моделі:

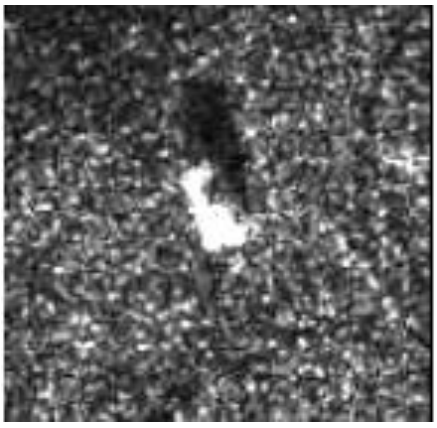

Рисунок 3.24 - Зображення Т62 з шумом при використанні MIM

З отриманих результатів на зображеннях 3.23 та 3.24 можна побачити, що атака виявилась неефективною, навіть при високому значені шуму, покладеного на зображення.

Четвертою атакою був реалізований алгоритм Карліні та Вагнера(C&W) - Carlini & Wagner запропонували атаку, яка була побудована на ідеї вирішення проблеми оптимізації неправильної цілі прогнозу та мінімального терміну відстані. Після опублікованої роботи, де стверджувалося, що захисна дистиляція підвищила надійність моделей глибокого навчання, що призвело до зниження рівня успішності змагальних прикладів з 95% до 0,5%, Carlini & Wagner оскаржили це твердження, представивши нову методологію атаки, яка була успішний як проти дистильованих, так і дистильованих моделей. У їхньому підході використовувалася функція втрат на основі логітів замість перехресних втрат ентропії softmax, а цільова змінна була перенесена в простір argtanh, що допомогло вирішити проблему за допомогою сучасних розв'язувачів, таких як Adam. Цей підхід також використовував методи бінарного пошуку, щоб знайти оптимальний коефіцієнт, який врівноважував компроміс між прогнозом і мірою відстані. Використовуючи атаку CWL2, вони змогли побудувати більш успішні змагальні приклади з меншими збуреннями.

Для цієї атаки були вибрані такі кроки бінарного пошуку - кількість разів, яку ми виконуємо двійковий пошук, щоб знайти оптимальний компроміс - константа між нормою збурення та достовірністю класифікації, кількість ітерацій - 1000, кількість атак, які потрібно виконати одночасно - 128.

Отриманий результат:

```
This image belongs to 2S1 with a 0.34 percent confidence.
This image belongs to BRDM_2 with a 0.30 percent confidence.
This image belongs to BTR_60 with a 0.00 percent confidence.
This image belongs to D7 with a 0.01 percent confidence.
This image belongs to SLICY with a 0.00 percent confidence.
This image belongs to T62 with a 48.78 percent confidence.
This image belongs to T72 with a 0.00 percent confidence.
This image belongs to ZIL131 with a 0.00 percent confidence.
This image belongs to ZSU_23_4 with a 50.57 percent confidence.
```

Рисунок 3.25 - Результат CW

Як можна побачити на рисунку 3.25, зображення було класифіковано як зсу-23-4 з ймовірністю 50.57%, а як Т62 з ймовірністю 48.78%. Також з незначними значеннями ймовірностей малюнок був позначений як 2С1, та БРДМ-2.

Зображення Т62, на яке був накладений шум, і яке потім було розпізнано з використанням побудованої моделі:

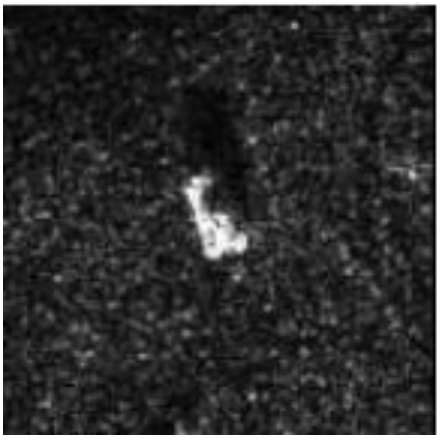

Рисунок 3.26 - Зображення Т62 з шумом при використанні CW

З отриманих результатів розпізнавання отримали, що зображення при вирахуванні максимального співпадіння буде позначене як зсу-23-4, при чому при накладанні шуму отримана картинка не відрізняється від початкової. Цей алгоритм є самим ефективним з перелічених, бо не змінює початкове зображення накладанням великої кількості шумів.

Висновки до розділу 3

Для побудови моделі змагальних атак на системи розпізнавання образів було використано набір даних для виявлення та розпізнавання рухомих і стаціонарних цілей (MSTAR), створений Агентством передових оборонних дослідницьких проектів США (DARPA) і Дослідницькою лабораторією ВПС США з 1995 по 1997 рік.

Перша модель розпізнавання - багатошарова модель персептрона (MLP), з трьома шарами налаштованих на 100 вузлів і з функцією активації Rectified Linear Unit (ReLU).

Друга модель розпізнавання – згорткова нейронна мережа з розширенням даних і доданими шарами відсіву, щоб уникнути перенавчання моделі. Також було додано два параметри зворотного виклику.

Були реалізовані такі змагальні атаки:

1. Метод швидкого градієнтного знака (FGSM) - якиф фокусується на ефективному пошуку змагального збурення, обмеженого L_∞-нормою.

2. Атака Карліні та Вагнера - атака, яка була побудована на ідеї вирішення проблеми оптимізації неправильної цілі прогнозу та мінімального терміну відстані.

3. Прогнозований градієнтний спуск - алгоритм, який намагається знайти збурення, яке максимізує втрати моделі на певному вході, зберігаючи розмір збурення меншим за епсилон.

4. Ітераційний метод імпульсу - техніка для прискорення алгоритмів градієнтного спуску шляхом накопичення вектора швидкості в напрямку градієнта функції втрат через ітерації.

Отримані такі результати:

Перший метод - зображення було класифіковано як брдм-2 з ймовірністю 77.24%, а як Т62 з ймовірністю 19.95 відсотків. Також з незначними значеннями ймовірностей малюнок був позначений як 2С1, ЗСУ-23-4 та БТР-60.

Другий метод - зображення було класифіковано як зсу-23-4 з ймовірністю 50.57%, а як Т62 з ймовірністю 48.78%. Також з незначними значеннями ймовірностей малюнок був позначений як 2С1, та БРДМ-2.

Третій метод - зображення було класифіковано як брдм-2 з ймовірністю 90.6%, а як Т62 з ймовірністю 8.07%. Також з незначними значеннями ймовірностей малюнок був позначений як 2С1, ЗСУ-23-4 та БТР-60.

Четвертий метод - зображення було класифіковано як брдм-2 з ймовірністю 16.71%, а як Т62 з ймовірністю 81.8%. Також з незначними значеннями ймовірностей малюнок був позначений як 2С1, ЗСУ-23-4 та БТР-60.

Порівняльний аналіз засвідчив переваги та недоліки розглянутих методів, і результати якого представлено в таблиці

	FGSM	CW	PGD	MIM
2S1	1.4	0.34	0.59	0.62
BRDM-2	77.24	0.3	90.6	16.71
BTR-60	0.49	0	0.32	0.06
D7	0	0.01	0	0
SLICY	0.09	0	0.06	0.02
T62	19.95	48.78	8.07	81.8
T72	0.01	0	0.01	0.01
ZILI131	0	0	0	0
ZSU-23-4	0.81	50.57	0.36	0.78

ВИСНОВКИ

Є такі три основні типи змагальних атак на системи розпізнавання образів:

Атаки ухилення - є найпоширенішими та найбільш дослідженими видами атак. Зловмисник маніпулює даними під час розгортання, щоб ввести в оману попередньо навчених класифікаторів. Оскільки вони виконуються на етапі розгортання, це найбільш практичні типи атак і найчастіше використовувані атаки на сценарії вторгнення та зловмисного програмного забезпечення.

Атаки отруєння - зловмисник впливає на навчальні дані або їх мітки, щоб призвести до зниження продуктивності моделі під час розгортання. Отже, отруєння - це по суті змагальне зараження навчальних даних. Оскільки системи ML можна перенавчати за допомогою даних, зібраних під час роботи, зловмисник може отруїти дані, впроваджуючи шкідливі зразки під час роботи, які згодом порушують або впливають на повторне навчання.

Дослідницькі атаки - не змінюють навчальний набір, а натомість намагаються отримати інформацію про стан, досліджуючи учня. Змагальні приклади створені таким чином, що учень передає їх як законні приклади на етапі тестування.

Моделі для реалізації змагальних атак на системи розпізнавання образів

1. Багатошаровий персептрон - це модель нейронних мереж, яка працює як універсальний апроксиматор, тобто може апроксимувати будь-яку безперервну функцію.

2. Згорткова нейронна мережа - це алгоритм глибокого навчання, який може сприймати вхідне зображення, призначати важливість (вагові значення та зміщення) різним аспектам/об’єктам зображення та мати можливість відрізнити один від іншого. Попередня обробка, необхідна в ConvNet, набагато менша порівняно з іншими алгоритмами класифікації. У той час як у

примітивних методах фільтри розробляються вручну, після достатнього навчання, ConvNets мають можливість вивчати ці фільтри/характеристики.

3. Мережа VGG19 – це згорткова нейронна марежа, яка має шість основних структур, кожна з яких в основному складається з кількох зв'язаних згорткових шарів і повнозв'язаних шарів. Розмір згорткового ядра становить 3*3, а вхідний розмір — 224*224*3. Кількість шарів, як правило, становить 16-19.

Змагальні атаки:

1. Метод швидкого градієнтного знака (FGSM) - фокусується на ефективному пошуку змагального збурення, обмеженого L_∞-нормою. Під час навчання моделі ML задана функція втрат мінімізується, щоб знайти оптимальний набір параметрів Θ для класифікатора C так, щоб C правильно класифікував більшість навчальних даних.

2. Атака Карліні та Вагнера (C&W) - спрямована на пошук мінімально збуреного збурення. Зокрема, вони перетворили задачу оптимізації з обмеженнями в коробку в задачу оптимізації без обмежень, яку потім можна вирішити за допомогою стандартних алгоритмів оптимізації.

3. Базовий ітераційний метод(BIM) - запропонований Куракіним та ін. є ітеративним уточненням FGSM. BIM використовує ітераційну лінеаризацію функції втрат, а не одноразову лінеаризацію в FGSM.

4. DeepFool - ефективний метод обчислення мінімальних збурень, необхідних для того, щоб змусити класифікатор неправильно класифікувати зображення.

5. Атака прогнозованого градієнтного спуску (PGD) - це атака білого ящика, що означає, що зловмисник має доступ до градієнтів моделі, тобто зловмисник має копію ваг вибраної моделі.

6. Метод імпульсу(MI-FGSM) - це техніка для прискорення алгоритмів градієнтного спуску шляхом накопичення вектора швидкості в напрямку градієнта функції втрат через ітерації.

Для побудови моделі змагальних атак на системи розпізнавання образів було використано набір даних для виявлення та розпізнавання рухомих і стаціонарних цілей (MSTAR), створений Агентством передових оборонних дослідницьких проектів США (DARPA) і Дослідницькою лабораторією ВПС США з 1995 по 1997 рік.

Перша модель розпізнавання - багатошарова модель персептрона (MLP), з трьома шарами налаштованих на 100 вузлів і з функцією активації Rectified Linear Unit (ReLU).

Друга модель розпізнавання – згорткова нейронна мережа з розширенням даних і доданими шарами відсіву, щоб уникнути перенавчання моделі. Також було додано два параметри зворотного виклику.

Були реалізовані такі змагальні атаки:

1. Метод швидкого градієнтного знака (FGSM) - якиф фокусується на ефективному пошуку змагального збурення, обмеженого L_∞-нормою.

2. Атака Карліні та Вагнера - атака, яка була побудована на ідеї вирішення проблеми оптимізації неправильної цілі прогнозу та мінімального терміну відстані.

3. Прогнозований градієнтний спуск - алгоритм, який намагається знайти збурення, яке максимізує втрати моделі на певному вході, зберігаючи розмір збурення меншим за епсилон.

4. Ітераційний метод імпульсу - техніка для прискорення алгоритмів градієнтного спуску шляхом накопичення вектора швидкості в напрямку градієнта функції втрат через ітерації.

Отримані такі результати:

Перший метод - зображення було класифіковано як брдм-2 з ймовірністю 77.24%, а як Т62 з ймовірністю 19.95 відсотків. Також з незначними значеннями ймовірностей малюнок був позначений як 2С1, ЗСУ-23-4 та БТР-60.

Другий метод - зображення було класифіковано як зсу-23-4 з ймовірністю 50.57%, а як Т62 з ймовірністю 48.78%. Також з незначними значеннями ймовірностей малюнок був позначений як 2С1, та БРДМ-2.

Третій метод - зображення було класифіковано як брдм-2 з ймовірністю 90.6%, а як Т62 з ймовірністю 8.07%. Також з незначними значеннями ймовірностей малюнок був позначений як 2С1, ЗСУ-23-4 та БТР-60.

Четвертий метод - зображення було класифіковано як брдм-2 з ймовірністю 16.71%, а як Т62 з ймовірністю 81.8%. Також з незначними значеннями ймовірностей малюнок був позначений як 2С1, ЗСУ-23-4 та БТР-60.

Порівняльний аналіз засвідчив переваги та недоліки розглянутих методів, і результати якого представлено в таблиці

	FGSM	CW	PGD	MIM
2S1	1.4	0.34	0.59	0.62
BRDM-2	77.24	0.3	90.6	16.71
BTR-60	0.49	0	0.32	0.06
D7	0	0.01	0	0
SLICY	0.09	0	0.06	0.02
T62	19.95	48.78	8.07	81.8
T72	0.01	0	0.01	0.01
ZILI131	0	0	0	0
ZSU-23-4	0.81	50.57	0.36	0.78

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Adversarial Attacks and Defences: A Survey [Електронний ресурс]. - Режим доступу: https://arxiv.org/pdf/1810.00069.pdf

2. Multi-Targeted Adversarial Example in Evasion Attack on Deep Neural Network [Електронний ресурс]. - Режим доступу: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8439941

3. Secure Kernel Machines against Evasion Attacks [Електронний ресурс]. - Режим доступу: https://pralab.diee.unica.it/sites/default/files/russu16-aisec.pdf

4. Mitigating Evasion A□acks to Deep Neural Networks via Region-based Classification [Електронний ресурс]. - Режим доступу: https://arxiv.org/pdf/1709.05583.pdf

5. A survey on adversarial attacks and defences [Електронний ресурс]. - Режим доступу: https://ietresearch.onlinelibrary.wiley.com/doi/10.1049/cit2.12028

6. Poisoning attacks and countermeasures in intelligent networks: Status quo and prospects [Електронний ресурс]. - Режим доступу: https://www.sciencedirect.com/science/article/pii/S235286482100050X#sec3

7. Privacy in Pharmacogenetics: An End-to-End Case
Study of Personalized Warfarin Dosing [Електронний ресурс]. - Режим доступу: https://www.usenix.org/system/files/conference/usenixsecurity14/sec14-paper-fredrikson-privacy.pdf

8. How to attack Machine Learning (Evasion, Poisoning, Inference, Trojans, Backdoors) [Електронний ресурс]. - Режим доступу: https://towardsdatascience.com/how-to-attack-machine-learning-evasion-poisoning-inference-trojans-backdoors-a7cb5832595c

9. What Is Adversarial Machine Learning? Attack Methods in 2022 [Електронний ресурс]. - Режим доступу: https://viso.ai/deep-learning/adversarial-machine-learning/

10. Deep Learning Classification of Land Cover and Crop Types Using Remote Sensing Data [Електронний ресурс]. - Режим доступу: https://www.researchgate.net/publication/315939269_Deep_Learning_Classification_of_Land_Cover_and_Crop_Types_Using_Remote_Sensing_Data

11. TensorFlow [Електронний ресурс]. - Режим доступу: https://ru.wikipedia.org/wiki/TensorFlow

12. Computational Analysis and Understanding of Natural Languages: Principles, Methods and Applications [Електронний ресурс]. - Режим доступу: https://www.sciencedirect.com/topics/computer-science/multilayer-perceptron

13 Using Goals in Model-Based Reasoning [Електронний ресурс]. - Режим доступу: https://www.sciencedirect.com/topics/computer-science/multilayer-perceptron

14 The Digital Twin Paradigm for Smarter Systems and Environments: The Industry Use Cases [Електронний ресурс]. - Режим доступу: https://www.sciencedirect.com/topics/computer-science/multilayer-perceptron

15 21st European Symposium on Computer Aided Process Engineering [Електронний ресурс]. - Режим доступу: https://www.sciencedirect.com/topics/computer-science/multilayer-perceptron

16. Convolutional Neural Networks [Електронний ресурс]. - Режим доступу: https://www.ibm.com/cloud/learn/convolutional-neural-networks

17. Application of a Novel and Improved VGG-19 Network in the Detection of Workers Wearing Masks [Електронний ресурс]. - Режим доступу: https://iopscience.iop.org/article/10.1088/1742-6596/1518/1/012041/pdf

18. What is the VGG-19 neural network? [Електронний ресурс]. - Режим доступу: https://www.quora.com/What-is-the-VGG-19-neural-network

19. Adversarial attacks in machine learning: What they are and how to stop them [Електронний ресурс]. - Режим доступу: https://venturebeat.com/security/adversarial-attacks-in-machine-learning-what-they-are-and-how-to-stop-them/

20. ML Attack Models: Adversarial Attacks and Data Poisoning Attacks [Електронний ресурс]. - Режим доступу: https://arxiv.org/ftp/arxiv/papers/2112/2112.02797.pdf

21. 4.3.2 Fast Gradient Sign Method, p. 8 [Електронний ресурс]. - Режим доступу: https://arxiv.org/ftp/arxiv/papers/2112/2112.02797.pdf

22. Adversarial attacks with FGSM (Fast Gradient Sign Method) [Електронний ресурс]. - Режим доступу: https://pyimagesearch.com/2021/03/01/adversarial-attacks-with-fgsm-fast-gradient-sign-method/

23. 4.3.6 Carlini and Wagner's Attack , p. 8 [Електронний ресурс]. - Режим доступу: https://arxiv.org/ftp/arxiv/papers/2112/2112.02797.pdf

24. 4.3.3 Basic Iterative Method, p. 9 [Електронний ресурс]. - Режим доступу: https://arxiv.org/ftp/arxiv/papers/2112/2112.02797.pdf

25. 4.3.4 DeepFool, p. 10 [Електронний ресурс]. - Режим доступу: https://arxiv.org/ftp/arxiv/papers/2112/2112.02797.pdf

26. A Review of DeepFool: a simple and accurate method to fool deep neural networks [Електронний ресурс]. - Режим доступу: https://medium.com/machine-intelligence-and-deep-learning-lab/a-review-of-deepfool-a-simple-and-accurate-method-to-fool-deep-neural-networks-b016fba9e48e

27. DeepFool - A simple and accurate method to fool Deep Neural Networks. [Електронний ресурс]. - Режим доступу: https://towardsdatascience.com/deepfool-a-simple-and-accurate-method-to-fool-deep-neural-networks-17e0d0910ac0

28. Know your enemy How you can create and defend against adversarial attacks [Електронний ресурс]. - Режим доступу: https://towardsdatascience.com/know-your-enemy-7f7c5038bdf3#:~:text=adversarially%20robust%20model.-,Projected%20Gradient%20Descent%20(PGD),copy%20of%20your%20model's%20weights.

29. Boosting Adversarial Attacks with Momentum. [Електронний ресурс]. - Режим доступу: https://arxiv.org/pdf/1710.06081.pdf

30. Keras’ Accuracy Metrics. [Електронний ресурс]. - Режим доступу: https://towardsdatascience.com/keras-accuracy-metrics-8572eb479ec7

31. A Gentle Introduction to the Fbeta-Measure for Machine Learning [Електронний ресурс]. - Режим доступу: https://machinelearningmastery.com/fbeta-measure-for-machine-learning/#:~:text=The%20F2%2Dmeasure%20is%20calculated,%5E2%20*%20Precision%20%2B%20Recall)

32. Matthews’s correlation coefficient: Definition, Formula and advantages [Електронний ресурс]. - Режим доступу: https://www.voxco.com/blog/matthewss-correlation-coefficient-definition-formula-and-advantages/#:~:text=Matthew's%20correlation%20coefficient%2C%20also%20abbreviated,2%20x%202%20contingency%20table.

ДОДАТОК 1 ТЕКСТ ПРОГРАМИ

```
#!/usr/bin/env python
# coding: utf-8
# In[15]:
get_ipython().run_line_magic('reset', '-fs')
# In[1]:
import tensorflow as tf
tf.compat.v1.enable_eager_execution()
from tensorflow import keras
from tensorflow.keras.preprocessing import image_dataset_from_directory
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import *
from tensorflow.keras.applications.vgg19 import VGG19
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix
import tensorflow_addons as tfa
#from tensorflow.python.framework.ops import enable_eager_execution

#enable_eager_execution()
params = {"ytick.color" : "w",
          "xtick.color" : "w",
          "axes.labelcolor" : "w",
          "axes.edgecolor" : "w",
          "figure.figsize" : (10,10),
          "axes.titlecolor" : 'w',
          "axes.facecolor" : 'w',
          "figure.facecolor" : 'k'}
```

```
colors = plt.rcParams['axes.prop_cycle'].by_key()['color']

get_ipython().run_line_magic('matplotlib', 'inline')
image_size = (128,128)
batch_size = 32
tf.executing_eagerly()
# In[2]:
train_ds = image_dataset_from_directory('D:/diplom_dataset/mstar_imgs',
                                        subset='training',
                                        image_size=image_size,
                                        labels='inferred',
                                        validation_split=.2,
                                        seed=123,
                                        label_mode='categorical',
                                        color_mode='grayscale',
                                        batch_size=batch_size)

val_ds = image_dataset_from_directory('D:/diplom_dataset/mstar_imgs',
                                        subset='validation',
                                        image_size=image_size,
                                        labels='inferred',
                                        validation_split=.2,
                                        seed=123,
                                        label_mode='categorical',
                                        color_mode='grayscale',
                                        batch_size=batch_size)

class_names = train_ds.class_names
with plt.rc_context(params):
```

```
  plt.figure(figsize=(10, 10))
  for images, labels in train_ds.take(1):
    for i in range(9):
      ax = plt.subplot(3, 3, i + 1)
      plt.imshow(images[i].numpy().astype("uint8"), cmap='gray')
      plt.title(class_names[np.argmax(labels[i])], )
      plt.axis("off")

val_batches = tf.data.experimental.cardinality(val_ds)
test_ds = val_ds.take(val_batches // 5)
val_ds = val_ds.skip(val_batches // 5)

print('Number of validation batches: %d' %
tf.data.experimental.cardinality(val_ds))
print('Number of test batches: %d' % tf.data.experimental.cardinality(test_ds))

AUTOTUNE = tf.data.AUTOTUNE

train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size=AUTOTUNE)
val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)
test_ds = test_ds.cache().prefetch(buffer_size=AUTOTUNE)

num_classes=9

metrics = [
      keras.metrics.CategoricalAccuracy(name='categorical_accuracy'),
      tfa.metrics.MatthewsCorrelationCoefficient(num_classes=9, name='MCC'),
      tfa.metrics.FBetaScore(num_classes=9, average='weighted', beta=2.0,
name='F2'),
      keras.metrics.AUC(name='auc'),
```

```
    keras.metrics.AUC(name='prc', curve='PR'),
]
print("Num GPUs Available: ", len(tf.config.list_physical_devices('GPU')))
# In[21]:
class_names
# In[22]:
model1 = keras.Sequential()

model1.add(InputLayer(input_shape=(image_size + (1,))))

model1.add(Flatten())

model1.add(Dense(100, activation='relu'))
model1.add(Dense(100, activation='relu'))
model1.add(Dense(100, activation='relu'))

model1.add(Dense(num_classes, activation='softmax'))

model1.summary()
# In[23]:
model1.compile(
  optimizer='adam',
  loss='categorical_crossentropy',
  metrics=metrics)
# In[24]:
history1 = model1.fit(train_ds, epochs=10, validation_data=val_ds)
# In[3]:
def holdout_results(model):
    result = model.evaluate(test_ds)
    return dict(zip(model.metrics_names, result))
```

```
# In[4]:
def plot_metrics(history):
  with plt.rc_context(params):
    metrics = ['loss', 'prc', 'F2', 'MCC']
    plt.figure(figsize=(10,10))
    for n, metric in enumerate(metrics):
      name = metric.replace("_"," ").capitalize()
      plt.subplot(2,2,n+1)
      plt.plot(history.epoch, history.history[metric], color=colors[0], label='Train')
      plt.plot(history.epoch, history.history['val_'+metric],
               color=colors[0], linestyle="--", label='Val')
      plt.xlabel('Epoch')
      plt.ylabel(name)
      if metric == 'loss':
        plt.ylim([0, plt.ylim()[1]])
      elif metric == 'auc':
        plt.ylim([0.8,1])
      elif metric == 'MCC':
        plt.ylim([-1,1])
      else:
        plt.ylim([0,1])


      plt.legend();
# In[5]:
def plot_cm(model, data):
    with plt.rc_context(params):
        y_true = []
        y_pred = []
        for x,y in data:
            y= tf.argmax(y,axis=1)
```

```
            y_true.append(y)
            y_pred.append(tf.argmax(model.predict(x),axis = 1))

        y_pred = tf.concat(y_pred, axis=0)
        y_true = tf.concat(y_true, axis=0)

        cm = confusion_matrix(y_true, y_pred)
        fig = plt.figure(figsize = (10,10))
        ax1 = fig.add_subplot(1,1,1)
        sns.set(font_scale=1.4) #for label size
        sns.heatmap(cm,cmap='binary', annot=True, fmt='d',
xticklabels=class_names, yticklabels=class_names, annot_kws={"size": 10},
            cbar = False);
        ax1.set_ylabel('True Values',fontsize=14)
        ax1.set_xlabel('Predicted Values',fontsize=14)
        plt.show()
# In[28]:
plot_metrics(history1)
# In[29]:
plot_cm(model1, val_ds)
# In[30]:
holdout_results(model1)
# In[34]:
model2 = tf.keras.Sequential([
  tf.keras.layers.InputLayer(input_shape=(image_size + (1,))),
  tf.keras.layers.Rescaling(1./255),
  tf.keras.layers.Conv2D(32, 3, activation='relu'),
  tf.keras.layers.MaxPooling2D(),
  tf.keras.layers.Conv2D(32, 3, activation='relu'),
  tf.keras.layers.MaxPooling2D(),
```

```
  tf.keras.layers.Conv2D(32, 3, activation='relu'),
  tf.keras.layers.MaxPooling2D(),
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dense(num_classes, activation='softmax')
])
model2.summary()
# In[35]:
model2.compile(
  optimizer='adam',
  loss='categorical_crossentropy',
  metrics=metrics)
# In[36]:
history2 = model2.fit(train_ds, epochs=10, validation_data=val_ds)
# In[37]:
plot_metrics(history2)
# In[38]:
plot_cm(model2, val_ds)


# In[39]:
holdout_results(model2)
# In[22]:
CNN = Sequential()
CNN.add(InputLayer(input_shape=(image_size + (1,))))
CNN.add(Conv2D(filters=10, kernel_size=3, activation='relu', padding='same'))
CNN.add(MaxPooling2D())
CNN.add(Conv2D(filters=20, kernel_size=3, activation='relu', padding='same'))
CNN.add(MaxPooling2D())
CNN.add(Conv2D(filters=30, kernel_size=3, activation='relu', padding='same'))
```

```
CNN.add(GlobalAveragePooling2D())
CNN.add(Dense(20, activation='relu'))
CNN.add(Dense(num_classes, activation='softmax'))
CNN.summary()
# In[23]:
CNN.compile(optimizer='adam',
            loss='categorical_crossentropy',
            metrics=metrics)
cnn_hist = CNN.fit(train_ds, epochs=10, validation_data=val_ds)
# In[24]:
plot_metrics(cnn_hist)
plot_cm(CNN, val_ds)
holdout_results(CNN)
# In[25]:
resize_and_rescale = tf.keras.Sequential([
  keras.layers.Resizing(128,128),
  keras.layers.Rescaling(1./255)])
data_augmentation = keras.Sequential(
    [
        keras.layers.RandomFlip("horizontal_and_vertical"),
        keras.layers.RandomRotation(0.2),
        keras.layers.RandomZoom(0.1),
    ]
)

plt.figure(figsize=(10, 10))
for images, _ in train_ds.take(1):
    for i in range(9):
        augmented_images = data_augmentation(images)
        ax = plt.subplot(3, 3, i + 1)
```

```
        plt.imshow(augmented_images[0].numpy().astype("uint8"), cmap='gray')
        plt.axis("off")
# In[26]:
def make_model(input_shape, num_classes):
    inputs = keras.Input(shape=input_shape)
    # Image augmentation block
    x = resize_and_rescale(inputs)
    x = data_augmentation(x)

    # Entry block
    x = keras.layers.Conv2D(32, 3, strides=2, padding="same")(x)
    x = keras.layers.BatchNormalization()(x)
    x = keras.layers.Activation("relu")(x)

    x = keras.layers.Conv2D(64, 3, padding="same")(x)
    x = keras.layers.BatchNormalization()(x)
    x = keras.layers.Activation("relu")(x)

    previous_block_activation = x  # Set aside residual

    for size in [128, 256, 512, 728]:
        x = keras.layers.Activation("relu")(x)
        x = keras.layers.SeparableConv2D(size, 3, padding="same")(x)
        x = keras.layers.BatchNormalization()(x)

        x = keras.layers.Activation("relu")(x)
        x = keras.layers.SeparableConv2D(size, 3, padding="same")(x)
        x = keras.layers.BatchNormalization()(x)

        x = keras.layers.MaxPooling2D(3, strides=2, padding="same")(x)
```

```
        # Project residual
        residual = keras.layers.Conv2D(size, 1, strides=2, padding="same")(
            previous_block_activation
        )
        x = keras.layers.add([x, residual])  # Add back residual
        previous_block_activation = x  # Set aside next residual

    x = keras.layers.SeparableConv2D(1024, 3, padding="same")(x)
    x = keras.layers.BatchNormalization()(x)
    x = keras.layers.Activation("relu")(x)

    x = keras.layers.GlobalAveragePooling2D()(x)
    if num_classes == 2:
        activation = "sigmoid"
        units = 1
    else:
        activation = "softmax"
        units = num_classes

    x = keras.layers.Dropout(0.5)(x)
    outputs = keras.layers.Dense(units, activation=activation)(x)
    return keras.Model(inputs, outputs)
xception = make_model(input_shape=image_size + (1,), num_classes=9)
xception.summary()
# In[27]:
callbacks=[
        keras.callbacks.EarlyStopping(patience=10, verbose=1, monitor='val_F2',
mode='max', restore_best_weights=True),
        keras.callbacks.ReduceLROnPlateau(factor=.5, patience=3, verbose=1),
```

```
    ]
xception.compile(
    optimizer=keras.optimizers.Adam(1e-3),
    loss="categorical_crossentropy",
    metrics=metrics,
)
xception_hist = xception.fit(
    train_ds, epochs=100, callbacks=callbacks, validation_data=val_ds,
)
# In[28]:
plot_metrics(xception_hist)
plot_cm(xception, val_ds)
holdout_results(xception)
# In[6]:
train_ds = image_dataset_from_directory('D:/diplom_dataset/mstar_imgs',
                                        subset='training',
                                        image_size=image_size,
                                        labels='inferred',
                                        validation_split=.2,
                                        seed=10,
                                        label_mode='categorical',
                                        color_mode='rgb',
                                        batch_size=batch_size)

val_ds = image_dataset_from_directory('D:/diplom_dataset/mstar_imgs',
                                        subset='validation',
                                        image_size=image_size,
                                        labels='inferred',
                                        validation_split=.2,
                                        seed=10,
```

```
                                label_mode='categorical',
                                color_mode='rgb',
                                batch_size=batch_size)
#class_names = train_ds.class_names
val_batches = tf.data.experimental.cardinality(val_ds)
test_ds = val_ds.take(val_batches // 5)
val_ds = val_ds.skip(val_batches // 5)
print('Number of validation batches: %d' %
tf.data.experimental.cardinality(val_ds))
print('Number of test batches: %d' % tf.data.experimental.cardinality(test_ds))

AUTOTUNE = tf.data.AUTOTUNE
train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size=AUTOTUNE)
val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)
test_ds = test_ds.cache().prefetch(buffer_size=AUTOTUNE)
#num_classes=9
#metrics = [
#      keras.metrics.CategoricalAccuracy(name='categorical_accuracy'),
#      tfa.metrics.MatthewsCorrelationCoefficient(num_classes=9, name='MCC'),
#      tfa.metrics.FBetaScore(num_classes=9, average='weighted', beta=2.0,
name='F2'),
#      keras.metrics.AUC(name='auc'),
#      keras.metrics.AUC(name='prc', curve='PR'),
#]

callbacks=[
      keras.callbacks.EarlyStopping(patience=10, verbose=1, monitor='val_F2',
mode='max', restore_best_weights=True),
      keras.callbacks.ReduceLROnPlateau(factor=.5, patience=3, verbose=1),
   ]
```

```
# In[7]:
last_model = tf.keras.Sequential([
  tf.keras.layers.InputLayer(input_shape=(image_size + (3,))),
  tf.keras.layers.Rescaling(1./255),
  tf.keras.layers.RandomFlip('horizontal_and_vertical'),
  tf.keras.layers.RandomRotation(0.2),
  tf.keras.layers.RandomZoom(0.1),
  tf.keras.layers.Conv2D(128, 5, activation='relu'),
  tf.keras.layers.MaxPooling2D(),
  tf.keras.layers.Conv2D(64, 4, activation='relu'),
  tf.keras.layers.MaxPooling2D(),
  tf.keras.layers.Conv2D(32, 3, activation='relu'),
  tf.keras.layers.MaxPooling2D(),
  tf.keras.layers.Conv2D(16, 2, activation='relu'),
  tf.keras.layers.MaxPooling2D(),
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(1024, activation='relu'),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dropout(0.5),
  tf.keras.layers.Dense(num_classes, activation='softmax')
])
last_model.summary()
# In[8]:
last_model.compile(
    optimizer=keras.optimizers.Adam(1e-3),
    loss="categorical_crossentropy",
    metrics=metrics,)

last_model_hist = last_model.fit(
    train_ds, epochs=100, callbacks=callbacks, validation_data=val_ds,
```

```
)
# In[9]:
plot_metrics(last_model_hist)
# In[10]:
plot_cm(last_model, val_ds)
# In[11]:
holdout_results(last_model)
# In[49]:
image = keras.utils.load_img(
    path="D:/diplom_dataset/t72_for_attacks/HB14931(t62).jpg",
    color_mode='rgb',
    target_size=(128,128)
)
image
image_array = keras.utils.img_to_array(image)
image_array = tf.expand_dims(image_array, 0)
image
# In[53]:
def predict_mod(model):
    predictions = model.predict(image_array)
    score = predictions[0]
    return(
    "This image most likely belongs to {} with a {:.2f} percent confidence."
    .format(class_names[np.argmax(score)], 100 * np.max(score)))
# In[144]:
def predict_mod_v2(imagee):
    predictions = last_model(imagee)
    score = predictions[0]
    return ("This image most likely belongs to {} with a {:.2f} percent confidence."
    .format(class_names[np.argmax(score)], 100 * np.max(score)))
```

```
predict_mod_v2(image_array)



# In[23]:
k = predict_mod(last_model)
k
# In[55]:
def predict_mod_v3(imagee):
    predictions = last_model(imagee)
    score = predictions[0]
    return predictions
def predict_mod_v4(imagee):
    predictions = last_model(imagee)
    score = predictions[0]
    for i in range(9):
        print ("This image belongs to {} with a {:.2f} percent
confidence.".format(class_names[i], 100 * score[i]))
    return "end"
# In[56]:
import cleverhans
from cleverhans.tf2.utils import compute_gradient
from cleverhans.tf2.attacks.fast_gradient_method import fast_gradient_method as
FGSM
# In[171]:
eps = 2.0 * 16.0 / 255.0
eps1 = 0.1
fgsm = FGSM(predict_mod_v3, image_array, eps=eps1, norm = np.inf,
targeted=False,clip_min=-1, clip_max=100.)
predict_mod_v4(fgsm)
# In[74]:
```

```
image_from_array = keras.utils.array_to_img(fgsm[0])
image_from_array
# In[42]:
predict_mod(last_model)
# In[76]:
image
# In[77]:
from cleverhans.tf2.attacks.projected_gradient_descent import
projected_gradient_descent as PGD
# In[204]:
pgd = PGD(predict_mod_v3, image_array, eps=1.2,eps_iter = 0.05,nb_iter = 50,
norm = np.inf, targeted=False,clip_min=-1, clip_max=220.)
# In[205]:
predict_mod_v4(pgd)
# In[199]:
image_from_array_2 = keras.utils.array_to_img(pgd[0])
image_from_array_2
# In[117]:
from cleverhans.tf2.attacks.momentum_iterative_method import
momentum_iterative_method as MIM
# In[212]:
mim = MIM(predict_mod_v3, image_array, eps=0.1,eps_iter = 0.05,nb_iter = 50,
norm = np.inf, targeted=False,  sanity_checks = False,clip_min=-1,
clip_max=110.)
predict_mod_v4(mim)
# In[213]:
image_from_array_3 = keras.utils.array_to_img(mim[0])
image_from_array_3
# In[137]:
from cleverhans.tf2.attacks.carlini_wagner_l2 import carlini_wagner_l2 as CW
```

```
# In[138]:



cw = CW(predict_mod_v3, image_array, clip_max=256.0)
# In[139]:
predict_mod_v4(cw)
# In[141]:
image_from_array_4 = keras.utils.array_to_img(cw[0])
image_from_array_4
# In[145]:
from cleverhans.tf2.attacks.spsa import spsa as SPSA
# In[226]:
sps = SPSA(predict_mod_v3, image_array,
tf.argmax(predict_mod_v3(image_array), 1),  eps=1.9,nb_iter = 5, clip_min = 0,
clip_max=255.)
# In[227]:
predict_mod_v4(sps)
# In[217]:
grad = compute_gradient(predict_mod_v3,
tf.nn.sparse_softmax_cross_entropy_with_logits, image_array,
tf.argmax(predict_mod_v3(image_array), 1), targeted = False)
optimal_perturbation = optimize_linear(grad, 0.1, np.inf)
# In[223]:
image_from_array_5 = keras.utils.array_to_img(optimal_perturbation[0])
image_from_array_5
#optimal_perturbation
```