Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра математики факультету інформатики

Кваліфікаційна робота

за спеціальністю „Прикладна математика” 113

Використання змагальних атак у задачах збільшення роздільної здатності зображень

Керівник курсової роботи

Крюкова Галина Віталіївна

Доцент, кандидат фізико-математичних наук

_____

(підпис)

Виконав студент 4 курсу

спеціальності «Прикладна математика»

Новиков Дмитро Володимирович

«__» _____ 2023 р.

Київ – 2023

Міністерство освіти і науки України

Зміст

Вступ

У попередній роботі розглядалися змагальні атаки, їхні типи та небезпека, яку вони становлять. Також, оглядово розглядалися методи захисту від змагальних атак. Посеред інших, можна виділити метод з покращенням роздільної здатності зображень. Детальне дослідження щодо того яким саме чином змагальні атаки досягають своєї мети з використанням метрик дистанції, а також огляд сучасних методів покращення якості й роздільної здатності зображень будуть розглянуті в цій роботі.

Мета дослідження. Метою дослідження є аналіз існуючих рішень для покращення якості зображень, і використання нейронної мережі для послідовного покращення якості змагального прикладу, доки він не буде правильно класифікуватися мережею-класифікатором. Для виконання поставленої задачі використовуватиметься навчена нейронна мережа-класифікатор, а також буде реалізована та натренована модель, заснована на архітектурі *SRCNN*. За допомогою змагальної атаки *FGSM*, що була детально розглянута та досліджена в попередній роботі, будуть створені змагальні приклади, які будуть некоректно розпізнаватися класифікатором, і для яких буде застосована вищезазначена мережа *SRCNN* з метою збільшення стійкості (англ. robustness) моделі-класифікатора за допомогою поєднання двох зазначених мереж.

Актуальність дослідження. Як було зазначено раніше, змагальні атаки несуть небезпеку для використання нейронних мереж в області комп’ютерного зору, а значить і для їхнього розвитку. Створення нових або покращення існуючих методів захисту від змагальних атак є важливим внеском у науку про дані, адже надає можливість науковцям використовувати нововведення в своїй роботі та дослідженнях.

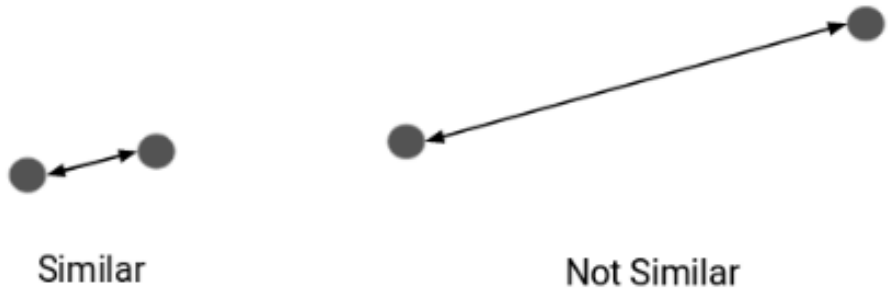
Список прийнятих скорочень і означень

1. **CNN** – Convolutional Neural Networks, згорткові нейронні мережі.
2. **SRCNN** – Super-Resolution Convolutional Neural Networks, згорткові нейронні мережі для створення зображень з високою роздільною здатністю.
3. **CW-L_p** – Carlini-Wagner Lp Distance Attack, атака Карліні-Вагнера з використанням L-норми.
4. **FGSM** – Fast Gradient Sign Method, один з видів змагальних атак, який використовує L_∞ та градієнт функції втрат моделі для її максимізації.
5. **ACE** – Adversarial Color Enhancement, один з видів змагальних атак, який створює реалістичні змагальні образи шляхом накладання ефекту фільтру на оригінальне зображення.
6. **GAN** – Generative Adversarial Network, алгоритм машинного навчання, який поєднує роботу двох нейронних мереж, одна з яких генерує (створює) нові образи, а інша намагається відрізнити правильні образи від неправильних.

РОЗДІЛ 1. Змагальні атаки та їхня реалізація, аналіз рішень для покращення якості зображень

1.1. Загальні дані про змагальні атаки

Як я раніше описував у своїй курсовій роботі, використання змагальних атак в цілому полягає у тому, що той, хто проводить атаку, генерує, так званий, змагальний приклад, який накладається на вхідне зображення для того, щоб нейронна мережа-класифікатор допустилася помилки, і згенерувала неправильний результат. Важливим є те, щоб після проведення змін зображення залишалося таким, що його може розпізнати людське око, як правильне. За рахунок вищевказаної бажаної поведінки, задача при проектуванні атаки зводиться до мінімізації модифікацій (для збереження валідності вхідних даних), які при цьому максимізують функцію втрат нейронної мережі. Функцією втрат у цьому контексті називається така функція, що обраховує різницю між очікуваним (правильним) і дійсним результатами, останній з яких є підсумком роботи нейронної мережі. Для мінімізації необхідних для внесення змін найчастіше використовують дистанційні метрики виду L_p, наприклад L_0, L_2, L_∞ тощо. Ці метрики слугують для визначення схожості між даними, коли розробляється класифікаційна модель. Якщо дві одиниці даних (точки) мають спільні риси, їх називають близькими.



Діаграма, яка ілюструє дистанцію між точками у залежності від наявності спільних характеристик

Зазвичай у галузі машинного використовуються наступні метрики дистанції:

1. Евклідова відстань
2. Манхеттенська відстань
3. Відстань Мінковського
4. Відстань Хемінга

1.2. Дослідження популярних підходів до знаходження відстані

Евклідова відстань

Евклідова відстань показує відстань між двома точками у просторі. Вона обраховується, як квадратний корінь квадратів різниці відповідних елементів. Вона відповідає метриці L_2, яка згадувалася вище і має широке використання в розробці нейронних мереж і алгоритмів (наприклад, алгоритмі K-середніх).

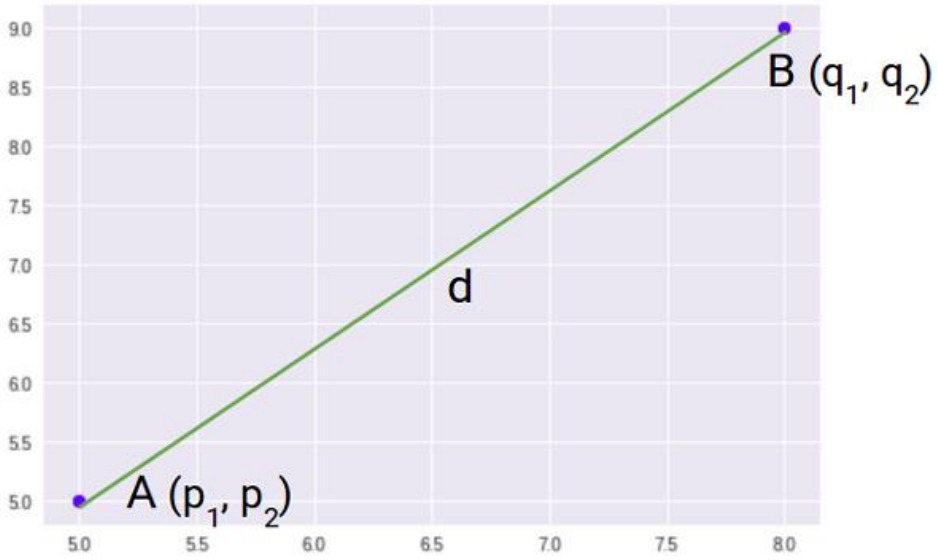
$$d = ((p_1 - q_1)^2 + (p_2 - q_2)^2)^{1/2}$$

Формула Евклідової відстані для двовимірного простору

Узагальнимо наведену формулу для n-розмірного простору:

$$D_e = \left(\sum_{i=1}^{n} (p_i - q_i)^2 \right)^{1/2}$$

Формула Евклідової відстані для простору розмірності n



Геометрична інтерпретація Евклідової відстані

Манхеттенська відстань

Манхеттенська відстань показує суму абсолютних різниць (тобто, взятих за модулем) між точками вздовж кожного виміру. Загальну формулу цієї метрики можна записати наступним чином:

$$D_{\text{m}} = \sum_{i=1}^{n} |p_i - q_i|$$

Формула Манхеттенської відстані для n-вимірного простору

Манхеттенська відстань працює більш оптимально для багатовимірних даних, ніж Евклідова і створює більш реалістичні шляхи між даними, ніж прямий відрізок, як було у випадку Евклідової відстані.

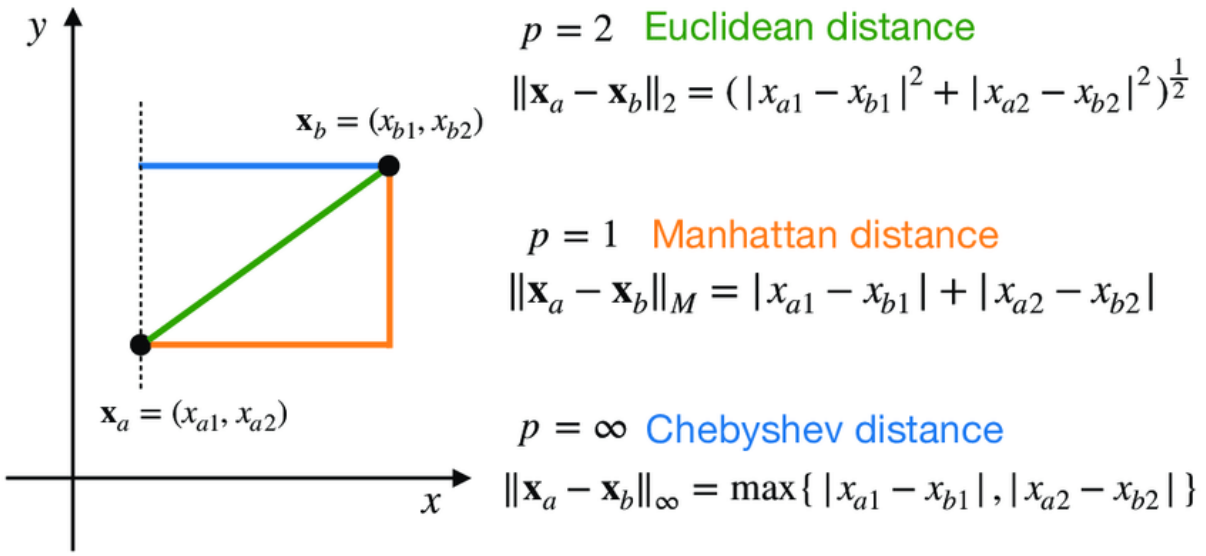
Відстань Мінковського

Відстань Мінковського це узагальнена форма Евклідової та Манхеттенської відстаней. Формула, очікувано, є комбінацією двох вищевказаних:

$$D = \left(\sum_{i=1}^{n} |p_i - q_i|^p \right)^{1/p}$$

Формула відстані Мінковського для n-вимірного простору

Параметр *p* відповідає за порядок норми. Для *p = 1*, відстань Мінковського буде рівна відповідній Манхеттенській, при *p = 2* – Евклідовій. Використання такої метрики дозволяє більш гнучко обирати між іншими за допомогою зміни порядку та тестування результатів для кожного з них.



Геометрична інтерпретація відстані Мінковського

Відстань Хеммінга

Відстань Хеммінга вимірює різницю між словами чи величинами однакової довжини. Вона дорівнює сумарній кількості елементів, які відрізняються у двох виразах на відповідних позиціях. Наприклад, нехай є слова *A = 00101101* та *B = 00101011*. Визначимо функцію *f: X => Y; x, y ∈{0; 1}* за умови *f(Ai ∧* Bi*) = 1* тоді і тільки тоді, коли *Ai ∧ Bi = 0*, де *i = [1, n]*, *n* – кількість символів у вхідних словах. Тоді шукану відстань *D* обраховується наступним чином:

$$D = \sum_{i=1}^{n} f(A_i \wedge B_i)$$

Формула відстані Хеммінга для вищезазначеної функції

Тоді для початкової умови $D = 2$.

1.3. Застосування метрик дистанцій у змагальних атаках

Як згадувалося раніше, основними метриками для дистанцій у змагальних атаках є векторні норми L_p. Вони слугують для підрахунку значень функції втрат, що має широке застосування в нейронних мережах. Загальна формула обрахунку норми для n-вимірного вектору:

$$\underbrace{\|x\|_p}_{p-Norm} = \left(\sum_{i}^{n} |x_i|^p\right)^{\frac{1}{p}} = \left(|x_1|^p + |x_2|^p + \cdots + |x_n|^p\right)^{\frac{1}{p}}$$

Формула обрахунку норми L_p у залежності від параметра p, $p \geq 1$

Тут x – вектор розмірності n. Розглянемо загальновживані випадки норм у залежності від параметра p:

1. $p = 0$, норма L_0

Незважаючи на умову-обмеження загальної формули ($p \geq 1$), можна порахувати норму для значення параметра $p = 0$. У такому разі справедлива наступна формула:

$$||v||_0 = \sum_{i=0}^{n} (v_i)^0, v \neq 0$$

Формула для обрахунку L_0 норми, $L_0(v)$ позначається як $||v||$

Тобто, за змістом L_0 – кількість ненульових елементів вектору v.

2. $p = 1$, норма L_1

$$\underbrace{\|x\|_1}_{L1\ Norm} = (\sum_{i}^{n} |x_i|) = (|x_1| + |x_2| + \cdots + |x_n|)$$

Формула обрахунку значення норми L_1

Тобто, фактично L_1 обраховує суму елементів вектору *v*. При застосуванні для функції втрат, за допомогою цієї норми рахується абсолютна похибка шляхом ділення отриманого результату на розмірність вектору.

3. $p = 2$, норма L_2

Серед усіх функцій виду L_p норма L_2 використовується найчастіше.

$$||v||_2 = (\sum_{i=0}^{n} |v_i|^2)^{\frac{1}{2}} = \sqrt{|v_1|^2 + |v_2|^2 + \ldots + |v_n|^2}$$

Формула норми L_2

Як слідує з формули вище, норма L_2 являє собою Евклідову відстань між точками вектору *v* та є коренем середньоквадратичної похибки при обрахунку значення функції втрат.

4. $p => \infty$, норма L_∞

Хоча обрахувати значення функції для нескінченно великого аргументу неможливо, можна з’ясувати до якої величини вона прямує по мірі зростання значення аргументу, тобто визначити її границю. Справедливою буде наступна формула:

$$lim_{p \Rightarrow \infty} ||v||_p = max(|v_1|, |v_2|, \ldots, |v_n|)$$

Формула норми L_∞

Вищезазначені метрики дистанції використовуються в змагальних атаках. Наприклад, в FGSM атаці ціллю є максимізація значення функції втрат, що суттєво знижує ймовірність того, що нейронна мережа зможе правильно

класифікувати зображення. При цьому оптимізація відстані не є ключовою задачею, тому зазвичай використовується норма L_∞ для створення змагального прикладу з урахуванням градієнту функції втрат для кожного пікселю. У той же час, наприклад, атака Карліні-Вагнера (CW-L_2) використовує метрику L_2 , оскільки окрім створення змагального прикладу пріоритетною задачею для атаки також було максимальне збереження якості зображення після накладання шуму, а метрика L_2 (в цьому контексті – фактична різниця між вихідним та отриманим зображеннями) набуває досить малих значень, коли вноситься багато невеликих змін одразу до багатьох пікселів. За рахунок подібних модифікацій, атака залишається ефективною, при цьому зберігаючи достатню для людського сприйняття якість зображення, тобто залишає його валідним для розпізнавання.

РОЗДІЛ 2. Аналіз рішень для покращення якості зображень

2.1. Дослідження існуючих методів покращення якості зображень

На сьогоднішній день існує багато різних способів покращення якості зображень. Далі приводиться опис найбільш вживаних методів. Із подібною задачею добре справляються згорткові нейронні мережі, які з часом довели свою надійність у використанні для роботи з зображеннями. Для покращення якості зображень підходять і моделі з учителем, і моделі без учителя. Навчанням з учителем називають тренування моделі на датасеті, який окрім самих навчальних даних містить ще й класи або підписи до цих даних. Такий вид навчання добре підходить для класифікації зображень, адже під час тренування модель поєднує назву класу та певний набір характеристик, притаманних для об’єктів, що йому належать, який вона вивчила з датасету. Моделі навчені без учителя (тобто на датасеті без класів чи підписів до даних) підходять для сегментації зображень, задач кластеризації тощо. У контексті покращення якості зображень, навчання з учителем має за мету покращити якість зображення певних об’єктів, засновуючись на їхній класифікації. Прикладом моделей з учителем, що використовуються для реалізації Super-Resolution алгоритму є мережі сімейства *CNN*, наприклад *SRCNN* або *ESRCNN*, що будуть розглянуті

пізніше. Моделі сімейства *GAN*, такі як *SRGAN* або *ESRGAN* слугують прикладом використання моделей без учителя для збільшення роздільної здатності зображень.

2.2. Гіпотеза щодо покращення якості зображень з використанням змагальних атак

Як було досліджено, основна ціль змагальних атак – додавання певних образів до зображення з метою змінити його класифікацію нейронною мережею. Проблемою залишається захист від таких атак. Раніше детально розглядався механізм генерації змагального образу за допомогою метрик відстані, а також оглядово були розглянуті загальні підходи до підвищення якості зображень. Спираючись на ці дослідження, висунемо гіпотезу – нехай існує нейронна мережа-класифікатор, що вміє розпізнавати зображення та відносити його до певного класу. Нехай, є певна бібліотека зображень і змагальних зображень, що були згенеровані після застосування атаки на початкову бібліотеку. Мета – розробка алгоритму Super-Resolution, задачею якого буде мінімізація впливу змагального образу на мережу-класифікатор, а також генерація валідних, з точки зору класифікатора, зображень. Розглянемо варіанти використання мереж сімейства *GAN* і *CNN*, які можна використати для покращення якості зображень: *SRGAN*, *ESRGAN, RCAN, SRCNN*. Розглянемо зазначені види моделей:

SRGAN – вид генеративної змагальної мережі, що використовується для підвищення роздільної здатності зображень. Ця нейронна мережа базується на використанні GAN, що дозволяє їй генерувати фотореалістичні зображення високої якості. SRGAN складається з двох компонентів – генератора та дискримінатора. Генератору передається зображення з низькою якістю, після чого він намагається його відтворити, збільшивши роздільну здатність. Дискримінатор намагається відрізнити згенеровані на попередньому кроці зображення від оригінальних високороздільних зображень з датасету. У результаті з’являється змагання між генератором і дискримінатором, що призводить до покращення якості роботи моделі, і, як результат, більш якісних

зображень, що отримуються на виході. Головною особливістю мереж SRGAN є використання perceptual loss і content loss, що сприяє збереженню деталей та характеристик зображення під час зміни його роздільної здатності, що впливає на генерацію більш природніх та реалістичних зображень. Розглянемо формулу для content loss, який є важливою складовою алгоритму:

$$l_{MSE}^{SR} = \frac{1}{r^2WH}\sum_{x=1}^{rW}\sum_{y=1}^{rH}(I_{x,y}^{HR} - G_{\theta_G}(I^{LR})_{x,y})^2$$

Середньоквадратична похибка, або content loss

Тут обраховується попіксельна середньоквадратична похибка для згенерованого зображення та оригінального з високою роздільною здатністю. W, H – відповідні розміри згенерованого зображення, rW, rH – розміри оригінального зображення, G – функція-генератор. Розглянемо змагальну втрату, що обраховується через передбачення дискримінатора для кожного з зображень у тренувальному датасеті:

$$l_{Gen}^{SR} = \sum_{n=1}^{N} -\log D_{\theta_D}(G_{\theta_G}(I^{LR}))$$

Змагальна втрата

Тут D – ймовірність, що G – оригінальне зображення з високою роздільною здатністю. За допомогою цих втрат можна обрахувати загальний perceptual loss:

$$l^{SR} = \underbrace{\underbrace{l_{\mathrm{X}}^{SR}}_{\text{content loss}} + \underbrace{10^{-3}l_{Gen}^{SR}}_{\text{adversarial loss}}}_{\text{perceptual loss (for VGG based content losses)}}$$

Формула для perceptual loss

Таким чином відповідно маємо загальну формулу функції втрат для мережі SRGAN, мінімізація якої дозволяє дозволяє нейронній мережі бути настільки ефективною.

ESRGAN – це такой вид генеративних змагальних мереж, який працює за схожим на *SRGAN* алгоритмом, тобто використовує концепції генератора, дискримінатора та perceptual loss. Відмінність полягає в архітектурі генератора, який у випадку *ESRGAN* складається не тільки зі звичайних згорткових і розгорткових шарів, а ще й з додаткових згорткових шарів зі збільшенням розміру, які слугують для підвищення роздільної здатності зображень. Також *ESRGAN* використовує модуль уваги до каналів, який має за мету зберегти важливі канали зображення, за рахунок чого покращуються його текстурні характеристики. За рахунок використання цих додаткових модулів, використання *ESRGAN* є більш ефективним методом для покращення роздільної здатності зображень, ніж звичайний *SRGAN*.

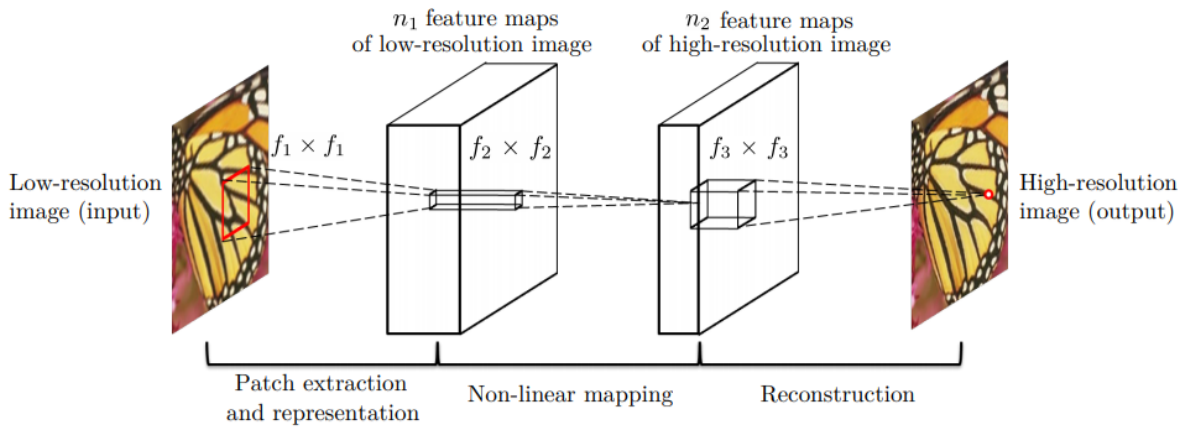
RCAN (Residual Channel Attention Network) – алгоритм, що заснований на ідеї моделей глибокого навчання з залишковим з’єднанням і увагою до каналів. Мета *RCAN* – покращення роздільної здатності зображення за допомогою використання корисної інформації з різних його каналів. Мережі *RCAN* складаються з шарів *RCAB* (англ. *Residual Channel Attention Block*), які, в свою чергу, складаються з двох основних компонентів – залишкової згортки та уваги до каналів. Перший з них дозволяє мережі зберігати та переносити корисну інформацію між шарами, уникаючи втрати важливих деталей. Залишкова згортка працює для створення «резидуальних» мап, що містять у собі корисну інформацію, яку необхідно додати до оригінального зображення. Увага до каналів використовується для зосередження на найбільш суттєвих каналах зображення. За рахунок використання цих двох компонентів, а також модулей групової згортки, який відповідає за зменшення втрат при обчисленнях і

покращення ефективності моделі, *RCAN* є дуже потужною та ефективною мережею для збільшення роздільної здатності зображень.

SRCNN – алгоритм для збільшення роздільної здатності зображень, що складається з трьох основних компонентів – вхідного шару, прихованих шарів і вихідного шару. Перший з них відповідає за отримання низькороздільного зображення та передачу його до прихованих шарів, які використовують модулі згортки та активації для отримання важливих ознак (шаблонів) зображення. Вихідний шар проводить згортку, засновуючись на даних, отриманих з попередніх шарів, з метою отримати нове високороздільне зображення. За рахунок використання згорткових шарів, нейронна мережа здатна навчитися виявляти складні залежності між пікселями зображення, за рахунок чого вона може відтворювати втрачені деталі та видавати більш природні та реалістичні суперрозширені зображення. SRCNN використовує метод навчання з учителем, де нейромережа навчається на парах низькороздільних та високороздільних зображень. У процесі навчання модель оптимізує свої ваги для мінімізації втрати між суперрозширеним зображенням, отриманим моделлю, і справжнім високороздільним зображенням.

$$L(\Theta) = \frac{1}{n} \sum_{i=1}^{n} ||F(\mathbf{Y}_i; \Theta) - \mathbf{X}_i||^2,$$

Функція втрат SRCNN, що визначає попіксельну різницю між оригінальним зображенням і створеним за допомогою мережі



Демонстрація принципу роботи SRCNN

РОЗДІЛ 3. Застосування змагальної атаки для покращення якості зображення

3.1. Опис ідеї, що лежить за створенням атаки

У результаті проведених досліджень, було встановлено, що використання нейронних мереж, а саме Super-Resolution моделей для підвищення стійкості мереж-класифікаторів, є актуальною темою робіт науковців у області великих даних. У такій парадигмі замість того, щоб робити передбачення за допомогою мережі-класифікатора, його спочатку обробляють з використанням Super-Resolution мережі, а вже результат її роботи передається далі. Ідея нового проекту полягає в тому, щоб підвищити стійкість нейронної мережі до змагальних атак з використанням алгоритму Super-Resolution. Оскільки створення власного зображення вимагає опанування певних інструментів, як, наприклад, Adobe Photoshop, для спрощення задачі з точки зору користувача, після проведеного дослідження стосовно вибору оптимального датасету, для нейронної мережі, що буде класифікувати зображення, був обраний датасет MNIST. Цей датасет складається з цифр, що написані від руки, і нейронна мережа, навчена на ньому, здатна розпізнавати цифру на зображенні. Для покращення якості зображення після накладання ефекту розмиття буде реалізована та використана нейронна мережа *SRCNN*, яка може з деякою ефективністю збільшувати роздільну здатність зображення й таким чином прибирати ефект розмиття та інші небажані та нереалістичні фрагменти з зображення. Важливими критеріями під час реалізації цих двох мереж є здатність мережі-класифікатора (*CNN*) правильно визначати яку цифру передав користувач, і спроможність мережі-покращувача (*SRCNN*) до підвищення якості отриманого після накладення змагального образу на зображення. Після того, як будуть досягнені вищезазначені цілі, необхідно ввести механізм змагальної атаки на мережу-класифікатор і переконатися, що тепер генеруються помилкові підписи. Будуть проведені тестування щодо того чи не змушує мережа-генератор

мережу-класифікатора робити некоректні передбачення, а також буде представлене покращення SRCNN мережі для підвищення стійкості CNN.

3.2. Реалізація згорткової мережі для класифікації зображень

Реалізація нейронної мережі *CNN*, навченої на датасеті *MNIST* буде представлена з використанням мови програмування *Python*, а також бібліотек (пакетів) *tensorflow* і *keras*.

```
from keras.datasets import mnist
from keras.models import Sequential
import keras
from keras.utils import np_utils
```

Імпорти необхідних для навчання мережі пакетів

Після цього необхідно імпортувати датасет за допомогою команди *(training_data, training_labels), (test_data, test_labels) = mnist.load_data()* та нормалізувати його, тобто привести зображення до роздільної здатності 28x28 пікселів, а також для зручності запевнитися, що замість значень 0 і 255 пікселі міститимуть відповідні їм 0 та 1:

```
training_data = training_data.reshape(training_data.shape[0], 28, 28, 1)
test_data = test_data.reshape(test_data.shape[0], 28, 28, 1)
training_data = training_data.astype('float32')
test_data = test_data.astype('float32')
training_data = training_data / 255
test_data = test_data / 255
```

Нормалізація даних і приведення до зручних значень

Проведемо розбиття підписів для тренування та валідації (тестування) по класам:

classes_number = 10
training_labels = np_utils.to_categorical(training_labels, classes_number)
test_labels = np_utils.to_categorical(test_labels, classes_number)

Оскільки кількість наявних класів є відомою, і завжди дорівнює 10 для кожної з цифр від 0 до 9, визначаємо константу *classes_number* з відповідним значенням. Після цього розбиваємо підписи (англ. labels) на відповідну кількість

категорій або класів. Можна переходити до створення моделі. Визначаємо модель, до якої будуть пізніше додаватися нові шари: *model = Sequential()*

Після цього додаємо перший шар - згортковий: *model.add(keras.layers.Conv2D(25, kernel_size=(3, 3), strides=(1, 1), padding='valid', activation='relu', input_shape=(28, 28, 1)))*

Тут перший параметр (25) означає кількість фільтрів, які будуть застосовані для обробки зображення, *kernel_size* - розмір “вікна”, тобто тієї частини матриці зображення, яку в певний момент часу “бачить” фільтр, *strides* - розмір зсуву фільтру від час руху по матриці для висоти та ширини відповідно, *activation* - активуюча функція для шару, *input_shape* був пояснений на попередньому кроці. Значення 1 - ширина каналу, яке показує, що в цьому випадку ми працюємо з чорно-білим зображенням (для кольорового ширина рівна 3 для кожного кольору з палітри RGB). Активуюча функція *ReLU* має зміст *ReLU(x) = max(0, x)*, тобто фактично заміняє всі негативні значення, отримані в результаті роботи шару, на 0. Застосуємо згладжування: *model.add(keras.layers.Flatten()).* Це проста дія, яка прибирає зайві виміри, тобто, наприклад, перетворює матрицю у вектор. Наступні *Dense* шари є повнозв’язними та слугують для формування передбачення мережі. Вони використовують активаційні функції, що передаються у якості аргумента, наприклад, функцію *softmax*.

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

Формула функції softmax

Вищезазначена функція конвертує вектор логітів у розподіл ймовірностей щодо належності зображення до певних класів, де логіти - значення, які обрахувалися в попередніх шарах нейронної мережі. На цьому побудова моделі

закінчена. Наступні етапи - компіляція, навчання та збереження моделі для подальшого її використання без необхідності її тренувати знову.

```
model.compile(loss='categorical_crossentropy', metrics=['accuracy'], optimizer='adam')
model.fit(training_data, training_labels, batch_size=128, epochs=20, validation_data=(test_data, test_labels))
model.save('./model.h5')
```

Компіляція, тренування та збереження моделі

Розглянемо результати тренування моделі:

```
Epoch 16/20
469/469 [==============================] - 9s 19ms/step - loss: 0.0038 - accuracy: 0.9986 - val_loss: 0.0806 - val_accuracy: 0.9838
Epoch 17/20
469/469 [==============================] - 9s 19ms/step - loss: 0.0043 - accuracy: 0.9985 - val_loss: 0.0883 - val_accuracy: 0.9830
Epoch 18/20
469/469 [==============================] - 9s 19ms/step - loss: 0.0038 - accuracy: 0.9988 - val_loss: 0.0705 - val_accuracy: 0.9852
Epoch 19/20
469/469 [==============================] - 9s 19ms/step - loss: 8.9706e-04 - accuracy: 0.9998 - val_loss: 0.0813 - val_accuracy: 0.9852
Epoch 20/20
469/469 [==============================] - 9s 19ms/step - loss: 2.6305e-04 - accuracy: 1.0000 - val_loss: 0.0758 - val_accuracy: 0.9865
```

Результати тренування моделі, 16-20 епоха

Як видно з рисунку вище, на двадцятій епосі була досгнена максимальна точність на тренувальних даних - 100%. Проведено тестування нейронної мережі на декількох зображеннях, створених самостійно:

```
cnn_model = tf.keras.models.load_model('./model.h5')

cnn_model.summary()


1 usage
def get_prediction_by_filename(model, filename):
    if os.path.isfile(f"images/{filename}"):
        img = cv2.imread(f"images/{filename}")[:, :, 0]
        img = cv2.resize(img, (28, 28))
        img = np.invert(np.array([img]))
        prediction = model.predict(img)
        print(f"Зображена цифра - {np.argmax(prediction)}")
        plt.imshow(img[0], cmap=plt.cm.binary)
        plt.show()


for filename in os.listdir(os.path.join(os.path.curdir, 'images')):
    get_prediction_by_filename(cnn_model, filename=filename)
```

Завантаження навченої моделі та прогнозування результатів для всіх файлів з директорії images

Використані зображення з цифрами 2, 4 та 8

Розглянемо результати роботи створеної мережі для цих зображень:

```
1/1 [==============================] - 0s 55ms/step
Зображена цифра - 2
1/1 [==============================] - 0s 9ms/step
Зображена цифра - 4
1/1 [==============================] - 0s 9ms/step
Зображена цифра - 8
```

Передбачення нейронної мережі для класифікації зазначених зображень

Як видно з результатів, наданих вище, створена нейронна мережа справляється з задачею коректної класифікації самостійно згенерованих зображень.

3.3. Реалізація згорткової мережі для покращення якості зображень

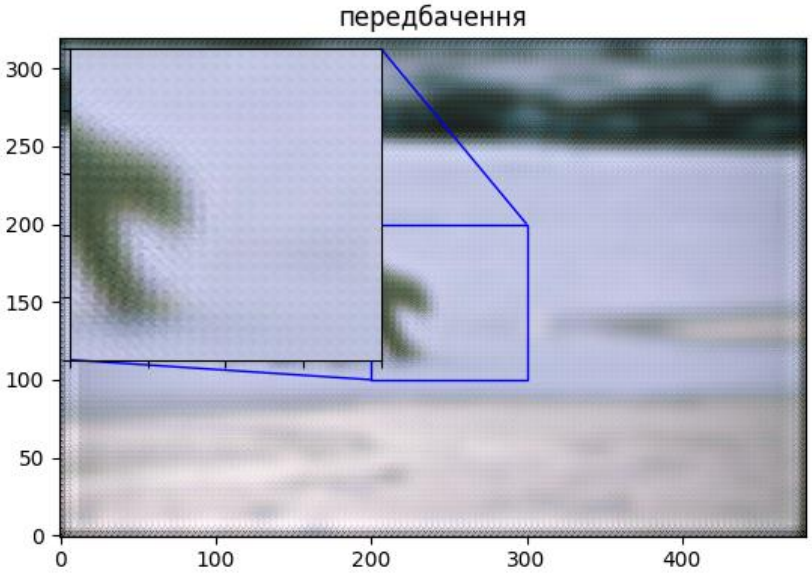
Оскільки перший з заданих критеріїв виконаний, перейдемо до створення *SRCNN* мережі для покращення якості зображень. Датасет для навчання мережі візьмемо з популярного сайту для інженерів машинного навчання Kaggle: https://www.kaggle.com/datasets/adityachandrasekhar/image-super-resolution. Цей датасет був обраний за рахунок того, що він спеціально розроблявся для мереж типу SRCNN, тож він найбільше підходить для вирішення задачі. За виключенням декількох несуттєвих деталей щодо підготовки датасету, початок процесу розробки такий самий, як і у випадку *CNN* моделі. Задача створення відповідної мережі зводиться до максимізації значення *PSNR* - пікового відношення сигналу до шуму, який його спотворює.

```
input_image = Input(shape=(256, 256, 3))
l1 = tf.keras.layers.Conv2D(64, 9, padding='same', activation='relu')(input_image)
l2 = tf.keras.layers.Conv2D(32, 1, padding='same', activation='relu')(l1)
l3 = tf.keras.layers.Conv2D(3, 5, padding='same', activation='relu')(l2)

SRCNN = Model(input_image, l3)
```

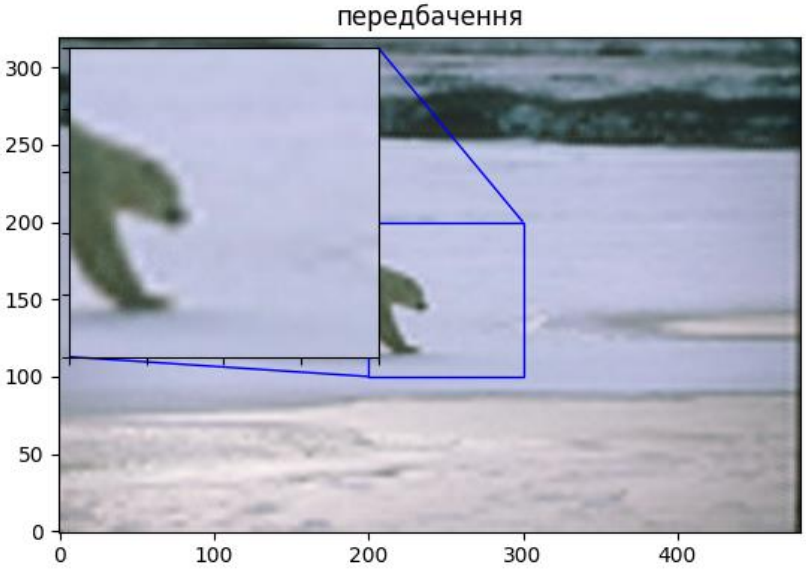
Створення SRCNN моделі для покращення якості зображень

Отже, мережа досить проста в своїй реалізації, та включає в себе три згорткових шари. Інші види шарів (наприклад, повнозв’язні) не потрібні для роботи мережі, бо вона фактично не робить передбачень з людської точки зору, тож інтерполяцію логітів у розподіл вводити сенсу немає. Для оцінки роботи нейронної мережі проаналізуємо її передбачення:



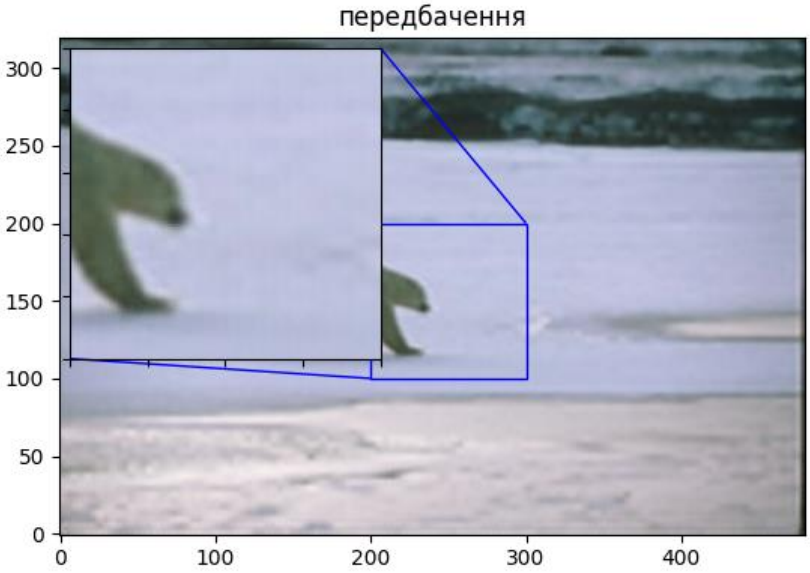
Початкове зображення

Як видно з першого рисунку, початкова якість картинки дуже низька навіть для людини, точно тож розпізнати образ, що на ній зображений, та бути впевненому в своєму результаті на 100 відсотків, не можна.



Результат після 20 епохи

Після 20 епохи навчання результати суттєво покращилися. Тепер можна сказати, що на рисунку зображений ведмідь. Проте, кількість шумів на зображенні дуже велика, а краї об’єктів дуже нерівні, за рахунок чого ще залишаються можливості його покращувати.



Результат після 80 епохи

Після 80 епохи навчання кількість шумів на зображенні зменшилася, контури об’єктів стали чіткішими. Враховуючи початкове зображення, є підстави вважати нейронну мережу такою, що справляється з поставленою задачею. Розглянемо результати, згенеровані мережею для валідаційного датасету:

Початкове зображення та покращене за допомогою створеної моделі

Як видно з результатів тестування, нейронна мережа дійсно підвищує роздільну здатність зображення. Перевіримо критерій два, а саме покращення якості зображення, та результат роботи попередньої створеної *CNN* мережі для

класифікації згенерованого зображення. Для накладання ефектів було обране зображення з цифрою 4:

Оригінальне зображення

Покращимо якість модифікованого зображення за допомогою мережі *SRCNN*:

Покращене зображення

Зі згенерованого варіанту можна зробити наступні висновки: по-перше, були прибрані шуми навколо контурів цифри 4, значно зменшене розмиття, збільшений розмір і роздільна здатність, що свідчить про покращення зображення. По-друге, отримана картинка схожа на зразок з тренувального датасету *MNIST,* який використовувався для навчання мережі-класифікатора, що трохи полегшує задачу створеній мережі. Перевіримо валідність результату:

```
1/1 [==============================] - 0s 49ms/step
Зображена цифра - 2
1/1 [==============================] - 0s 9ms/step
Зображена цифра - 4
1/1 [==============================] - 0s 9ms/step
Зображена цифра - 4
1/1 [==============================] - 0s 10ms/step
Зображена цифра - 8
```

Результати роботи мережі-класифікатора

Оскільки зображення 2, 4 та 8 уже були розташовані в директорії з попереднього експерименту, маємо коректне передбачення нейронної мережі. Отже, реалізація нейронної мережі для покращення якості зображень завершена.

3.4. Реалізація змагальної атаки та використання SRCNN мережі

Для виконання останнього критерію та завершення практичного тестування реалізації нейронної мережі для покращення якості зображення залишилося створити імплементацію змагальної атаки та провести декілька атак на нейронну мережу-класифікатор, після чого передати результати (отримані зображення) до генератора й подивитися результати. У якості методу використаємо добре досліджену та достатньо потужну атаку FGSM. Коли мережа-генератор готова повернути зображення, воно буде конвертуватися в необхідний формат, розмір тензора зменшуватиметься з 256 на 256 чисел до 28 на 28, як того вимагає CNN мережа, а також 3 канали (які з’являються за рахунок підтримки кольору в SRCNN) перетворяться в 1.

```
img = cv2.imread(f"images/{filename}")
img = cv2.resize(img, (256, 256))
img = np.invert(np.array([img]))
prediction = model.predict(img)
```

Отримання згенерованого зображення за допомогою SRCNN

Після отримання результату, перейдемо до необхідних трансформацій:

```
prediction_image = tf.convert_to_tensor(prediction[0])
gray_scaled = tf.image.rgb_to_grayscale(prediction_image)
resized_image = tf.image.resize(gray_scaled, [28, 28])
resized_image = tf.expand_dims(resized_image, axis=-1)
resized_image = tf.expand_dims(resized_image, axis=0)
```

Перетворення над зображенням для поєднання роботи двох мереж

У першому рядку зображення з передбачення мережі-генератора конвертується в тензор, оскільки саме такий формат очікується у CNN-класифікаторі. Далі кількість каналів зменшується до одного (відбувається grayscaling). Після цього проводиться зміна розміру зображення до 28 на 28 пікселів. Після видалення зайвих просторів, отримуємо форму зображення (28, 28, 1). Перейдемо до реалізації атаки FGSM:

```
def create_adversarial_pattern(input_image, input_label):
    with tf.GradientTape() as tape:
        tape.watch(input_image)
        prediction = cnn_model(input_image)
        loss = loss_object(input_label, prediction)

    grad = tape.gradient(loss, input_image)
    signed_grad = tf.sign(grad)
    return signed_grad
```

Реалізація атаки FGSM

Як видно з коду, для реалізації потрібен доступ до оригінальної моделі. Тобто, подібна атака можлива у випадку white-box атаки або при застосуванні методу вилучення моделі, який описувався в минулій роботі. У цілому, атака зводиться до обрахунку градієнту функції втрат і ставить собі за ціль її максимізацію. Застосуємо атаку для отримання бібліотеки змагальних зображень:

```python
def fill_adversarial_directory(model):
    epsilon = 0.01
    index = 0
    original_predictions, _ = make_cnn_predictions(model)
    for filename in os.listdir('./images'):
        image = transform_image(f"./images/{filename}", 28)
        transformed_prediction = tensorflow.reshape(original_predictions[index],
                                                    (1, original_predictions[index].shape[-1]))
        pattern = create_adversarial_images(model, image, transformed_prediction)[0]
        adv_x = image + epsilon * pattern
        adv_x = tensorflow.clip_by_value(adv_x, -1, 1)
        adv_x = tensorflow.squeeze(adv_x, axis=0)
        save_img(f'./adversarial/{filename}', adv_x)
        index += 1
```

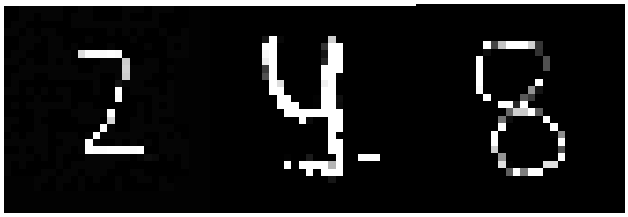
Застосування атаки FGSM

Спочатку виявляємо справжнє передбачення атакованої моделі. Це необхідно зробити, адже саме шанс випадіння цього значення передбачення потрібно мінімізувати, схиливши нейронну мережу до іншого вибору. Після цього застосовуємо атаку на модифікованому раніше зображенні та отриманому результату передбачення. Застосовуємо математичну формулу FGSM атаки, що має наступний вигляд:

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

Формула FGSM атаки

Параметром ε у кодовій реалізації є число 0.25, а усі інші компоненти є очевидними після співставлення формули та реалізації. Розглянемо результати роботи змагальної атаки:



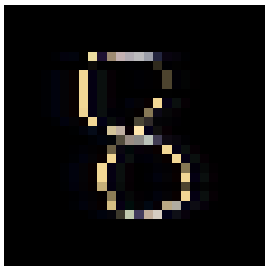
Зліва-направо: раніше представлені зображення 2, 4, 8

Маємо бібліотеку зображень, які можна в подальшому використати для покращення за допомогою SRCNN. Для виконання останнього з перелічених на початку критеріїв, необхідно переконатися в тому, що нейронна мережа-

класифікатор допускається помилки при розпізнаванні цих зображень, що доведе ефективність атаки.

```
1/1 [==============================] - 0s 48ms/step
Зображена цифра - 8
1/1 [==============================] - 0s 9ms/step
Зображена цифра - 8
1/1 [==============================] - 0s 9ms/step
Зображена цифра - 8
1/1 [==============================] - 0s 9ms/step
Зображена цифра - 8
```

Результати роботи створеної CNN мережі після перевірки на змагальних прикладах



Зображення, згенероване за допомогою SRCNN на прикладі цифри 8

Як видно з рисунку зверху, після застосування змагальної атаки, нейронна мережа-класифікатор почала допускатися помилок, і видає неправильні результати. Застосуємо SRCNN мережу для деяких з наведених оригінальних зображень та перевіримо результат:

```
Перевіримо чи правильно розпізнаються зображення після застосування Super-Resolution
------------------------------------------------
Зображене число - 2
Зображене число - 2
Зображене число - 3
Зображене число - 5
Зображене число - 8
------------------------------------------------
```

Як видно з рисунку, були додані деякі додаткові зображення, й вони всі класифікуються коректно. Для покращення алгоритму Super-Resolution, проведемо змагальне тренування (англ. Adversarial training) за допомогою вилучення шаблонів з CNN моделі та накладання змагальних прикладів:

```python
def extract_features(images):
    feature_images = []
    for batch in images:
        feature_batch = []
        for image in batch:
            if len(image.shape) < 3:
                image = tf.expand_dims(image, axis=-1)
            resized_image = tf.image.resize(image, (28, 28))
            resized_image = tf.squeeze(resized_image, axis=-1)
            resized_image = tf.convert_to_tensor(resized_image, dtype=tf.float32)
            resized_image /= 255.0
            feature_batch.append(resized_image)
        feature_images.append(feature_batch)

    feature_images = tf.concat(feature_images, axis=0)
    feature_images = tf.reshape(feature_images, (-1, 28, 28, 1))
    features = cnn.predict(feature_images)
    features = tf.convert_to_tensor(features)
    return features
```

Вилучення шаблонів (англ. features)

Тут визначаються шаблони, на які звертає увагу мережа-класифікатор. Ця інформація є важливою в задачі підвищення стійкості моделі, бо для коректної роботи генератора, йому необхідно знати на що саме звертати увагу. Після цього необхідно реалізувати вищевказану змагальну атаку за тим самим алгоритмом. Разом з вилученими шаблонами та змагальною атакою, що проводиться на зображення в датасеті, необхідно додати нові картинки до тренувального сету моделі:

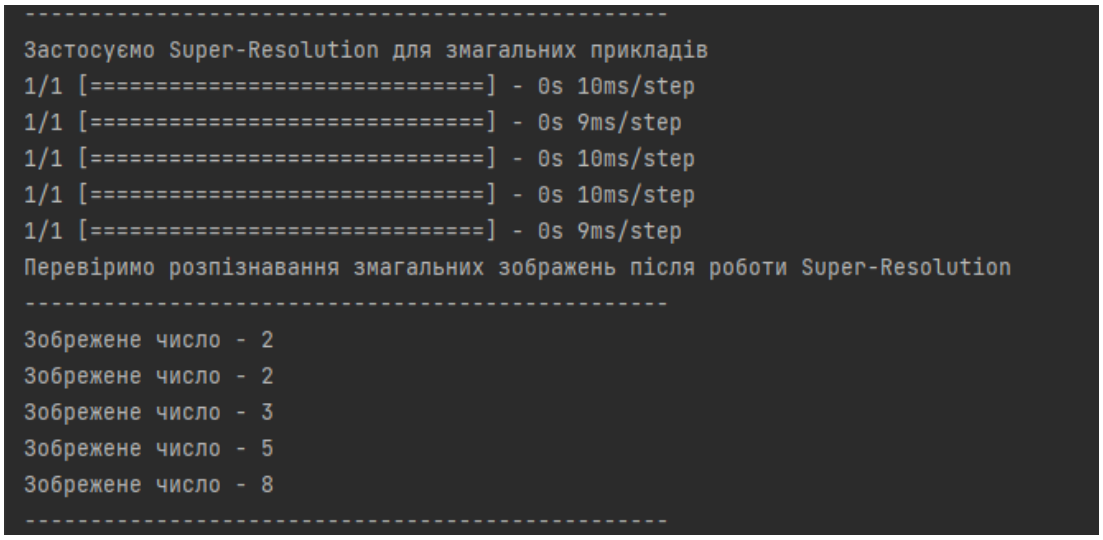
```python
def feature_extractor_adversarial_training(dataset, epochs=10, batch_size=32):
    for epoch in range(epochs):
        print(f"Epoch {epoch + 1}/{epochs}")
        for batch_images in dataset.batch(batch_size):
            adversarial_images = generate_adversarial_perturbations(batch_images)
            model.train_on_batch(adversarial_images, batch_images)

feature_extractor_adversarial_training(train_ds, epochs=epochs, batch_size=32)
```

Додавання змагальних зображень до готового датасету

Такий вид тренування значно підвищить обізнаність SRCNN мережі про модель, яку необхідно захищати та її алгоритми класифікації, а також навчить

мережу розпізнавати змагальні приклади та покращувати їх, щоб збільшити коректність передбачень моделі-цілі. Розглянемо результати роботи мережі:



Результати передбачень після застосування Super-Resolution

Як видно з рисунку, правильними виявилися всі п’ять з п’яти передбачень. Отже, мета досягнена - була підвищена стійкість моделі до змагальних атак з використанням алгоритму Super-Resolution та його покращення шляхом введення змагального навчання.

3.5. Висновок з практичної задачі

Під час виконання практичної задачі була висунута гіпотеза стосовно використання змагальних атак в задачах збільшення роздільної здатності зображень. Були реалізовані дві нейронні мережі: CNN для класифікації зображень, навчена на датасеті MNIST, та SRCNN для підвищення якості зображень, зокрема збільшення їхньої роздільної здатності. Обидві мережі були ретельно протестовані та довели свою дієздатність. Також була досліджена та використана атака FGSM, яка використовувалася для генерації змагальних образів, що потім покращувалися за допомогою вищезгаданої SRCNN мережі. Для поліпшення отриманого результату були проведені відповідні дослідження, які показали, що кращих результатів можна досягти за допомогою збільшення кількості змагальних прикладів у тренувальному датасеті мережі-генератора, а отже й збільшення чи заміна самого датасету, покращення архітектури обох

моделей, проведення додаткових експериментів з вагами та параметрами моделей тощо. Розроблена та використана SRCNN модель була досить простою та примітивною і скоріше слугувала для цілей демонстрації можливості підвищення стійкості моделей для змагальних атак з використанням алгоритму Super-Resolution. При заміні цієї моделі на більш досконалу SRGAN, приклад реалізації якої можна знайти на репозиторії https://github.com/sgrvinod/a-PyTorch-Tutorial-to-Super-Resolution, або ж спеціальної моделі, що була розроблена для захисту від змагальних атак (https://github.com/yuejiutao/Robust-Real-World-Image-Super-Resolution-against-Adversarial-Attacks) та запропонована в роботі “Robust Real-World Image Super-Resolution against Adversarial Attacks” за авторством Jiutao Yue, Haofeng Li, Pengxu Wei, Guanbin Li, Liang Lin.

Висновок

У цій роботі був розглянутий метод збільшення роздільної здатності зображень з використанням змагальних атак. Була реалізована мережа типу CNN для класифікації зображень і змагальна атака FGSM, що використовувалася для генерації бібліотеки змагальних зображень. Також був реалізований і покращений алгоритм Super-Resolution на прикладі мережі SRCNN, що в подальшому використовувався для підвищення стійкості мережі-класифікатора до змагальних атак. Експеримент виявився вдалим, оскільки після застосування нейронної мережі для підвищення роздільної здатності зображень, що була натренована на датасеті BSDS500, а також власних змагальних прикладах, CNN мережа почала правильно розпізнавати покращені атаковані зображення. У подальшому ці напрацювання можна використовувати для обробки вхідних даних мереж-класифікаторів, щоб мінімізувати вплив змагального образу на класифікацію, і таким чином зменшувати шкоду, що потенціально могла бути нанесена за допомогою використання змагальної атаки. Окрім цього був розглянутий сам механізм роботи змагальної атаки з метою дослідження методу їх роботи, яке необхідне для висунення правильної гіпотези стосовно методів захисту від атак.

Джерела

1. Zhengyu Zhao, Zhuoran Liu, Martha Larson. Adversarial Color Enhancement: Generating Unrestricted Adversarial Images by Optimizing a Color Filter, 2020
2. http://www.eecs.berkeley.edu/Research/Projects/CS/vision/grouping/BSR/BSR_bsds500.tgz
3. https://www.kaggle.com/datasets/adityachandrasekhar/image-super-resolution
4. https://github.com/sgrvinod/a-PyTorch-Tutorial-to-Super-Resolution
5. https://arxiv.org/pdf/1609.04802v5.pdf
6. Jiutao Yue, Haofeng Li, Pengxu Wei, Guanbin Li, Liang Li. Robust Real-World Image Super-Resolution against Adversarial Attacks
7. https://github.com/yuejiutao/Robust-Real-World-Image-Super-Resolution-against-Adversarial-Attacks
8. https://keras.io/
9. Christian Ledig, Lucas Theis, Ferenc Huszar, Jose Caballero. Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network
10. Aamir Mustafa, Salman H. Khan, Munawar Hayat, Jianbing Shen and Ling Shao. Image Super-Resolution as a Defense Against Adversarial Attacks
11. https://github.com/xinntao/ESRGAN
12. Andreas Lugmayr, Martin Danelljan, Radu Timofte. Unsupervised Learning for Real-World Super-Resolution