МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

ЦЕНТР ЗАОЧНОЇ, ДИСТАНЦІЙНОЇ ТА ВЕЧІРНЬОЇ ФОРМ НАВЧАННЯ

КАФЕДРА КОМП'ЮТЕРНИХ НАУК

СЕКЦІЯ ІКТ

КВАЛІФІКАЦІЙНА МАГІСТЕРСЬКА РОБОТА

на тему:

«Інтелектуальна інформаційна система відновлення зображень за допомогою генеративної змагальної мережі»

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

Керівник роботи — **Шаповалов С.П.**

Студента групи ІНмз – 91с — **Городничого В.О.**

СУМИ 2020

Сумський державний університет

(назва вузу)

Факультет *ЦЗДВФН* Кафедра *Комп'ютерних наук.*

Спеціальність *«Інформатика»*

Затверджую:

зав.кафедрою _____

_____

"_____" _____ 20___р.

ЗАВДАННЯ

НА ДИПЛОМНИЙ ПРОЕКТ (РОБОТУ) СТУДЕНТОВІ

Городничому Владиславу Олексійовичу

(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) *«Інтелектуальна інформаційна система відновлення зображень за допомогою генеративної змагальної мережі»*

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

2. Термін здачі студентом закінченого проекту (роботи) _____

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

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) *1) Постановка задачі дослідження. Аналіз проблеми. 2) Інформаційний огляд. 3) Вибір методу рішення. 4) Розробка програмного забезпечення системи. 5) Оформлення пояснювальної записки до дипломної роботи.*

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

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

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів дипломного проекту (роботи)	Термін виконання проекту (роботи)	Примітка
1.	*Постановка задачі дослідження. Аналіз проблеми*		
2.	*Інформаційний огляд*		
3.	*Вибір методу рішення*		
4.	*Розробка програмного забезпечення системи*		
5.	*Оформлення пояснювальної записки до дипломної роботи*		

Студент – дипломник _____

(підпис)

Керівник проекту _____

(підпис)

РЕФЕРАТ

Записка: 42 стор., 18 рис., 1 додаток, 14 джерел інформації.

Об’єкт дослідження — зображення низької якості або які підлягли пошкодженню.

Мета роботи — дослідити модель нейронних мереж для покращення зображень та розробка інтелектуальної інформаційної системи відновлення якості зображень.

Методи дослідження — аналіз рішень на основі генеративної змагальної мережі та алгоритмів інтерполяції.

Результати — інтелектуальна система відновлення якості зображення близького до оригіналу.

ІНТЕРПОЛЯЦІЯ, НЕЙРОННІ МЕРЕЖІ, ГЕНЕРАТИВНА ЗМАГАЛЬНА МЕРЕЖА, ФУНКЦІЇ ЗБУДЖЕННЯ, ФУНКЦІЇ ВТРАТ, МЕТОД ЛАНЦОША, БІЛІНІЙНА ІНТЕРПОЛЯЦІЯ, БІКУБІЧНА ІНТЕРПОЛЯЦІЯ

ЗМІСТ

ВСТУП

Наше сучасне життя тісно пов’язане з цифровими технологіями. Існує багато різних сфер та видів технологій що якісно покращує людське сприйняття того чи іншого фізичного процесу. Широкого розповсюдження здобули цифрова фото та відеозйомка. Але також варто мати на увазі що далеко не кожен технічний пристрій здатен задовольнити потреби якості користувачів та сфер діяльності які потребують кращої деталізації та менше спотворень самого зображення [1-7].

Покращення зображення в загальному розумінні – це процес поліпшення якості картинки без втрати інформації до отримання бажаного візуального результату. Підвищення якості включає ряд перетворень: підвищення розширення, освітлення, усунення оптичних спотворень тощо [8-9].

Алгоритми покращення зображень застосовуються в різних галузях нашого життя. Це не тільки зображення для сімейних фотоальбомів, але і медичні зображення, зображення радіології, фото реєстраторів, супутникові знімки тощо [1-5].

Існує безліч інструментів для ретуші зображень, але варто мати на увазі що якість результату залежить від суб’єктивного сприйняття того хто цим процесом займається. До того ж редагування зображень вручну займає багато часу.

Глибоке навчання – це нова галузь машинного навчання, що можна ефективно використовувати для обробки зображень. Різні типи мереж можна використовувати для завдань щодо поліпшення якості зображень, таких як видалення шуму, відновлення зображень високої роздільної здатності з зображень із низької.

Метою дипломної роботи є створення програмної реалізації обробки зображень за допомогою штучної нейронної мережі.

1 ОГЛЯД БУДОВИ ШТУЧНИХ НЕЙРОННИХ МЕРЕЖ

1.1 Штучні нейронні мережі

Нейронна мережа(neural network, NN) – це послідовність нейронів, з'єднаних між собою синапсами. Структура нейронної мережі прийшла в світ програмування з біології. Такі системи навчаються задач, розглядаючи приклади, без спеціального програмування під задачу. Наприклад, у розпізнаванні зображень вони можуть навчатися ідентифікувати зображення, які містять певних тварин, аналізуючи приклади зображень помічених як «тварина», чи «не тварина». І використовуючи результати для ідентифікації тварин в інших зображеннях без жодного контекстного знання про тварин, розвиваючи власний набір характеристик з навчального матеріалу який вони оброблюють. Завдяки такій структурі, машина одержує можливість аналізувати і навіть запам'ятовувати різну інформацію. Нейронні мережі здатні не тільки аналізувати вхідну інформацію, а й відтворювати її зі своєї пам'яті. Іншими словами, нейромережа – це машинна інтерпретація мозку людини, в якому знаходяться мільйони нейронів які передають інформацію у вигляді електричних імпульсів[2].

Нейронні мережі використовуються для вирішення складних **завдань**, які вимагають аналітичних обчислень подібних тим, що робить людський мозок. Найпоширенішими застосуваннями нейронних мереж є:

- *Класифікація* - розподіл даних за параметрами. Наприклад, на вхід дається набір людей і потрібно вирішити, кому з них давати кредит, а кому ні. Цю роботу може виконати нейронна мережа, аналізуючи таку інформацію як: вік, платоспроможність, кредитна історія, тощо.
- *Передбачення* - можливість прогнозувати наступний крок. Наприклад, зростання або падіння акцій, опираючись на ситуацію фондового ринку.
- *Обробка даних та моделювання* – наприклад, моделювання геопроцесів в геонауках чи астрономії.

- *Робототехніка* – керування автоматизованими маніпуляторами та протезами.
- *Розпізнавання* – має саме широке застосування нейронних мереж в наш час. Використовується в багатьох системах, коли ви, наприклад, шукаєте фото, або в камерах телефонів, коли вона визначає положення вашого обличчя і виділяє його і ще в багатьох інших.

1.2 Складові штучної нейронної мережі

1.2.1 Нейрони та синапси

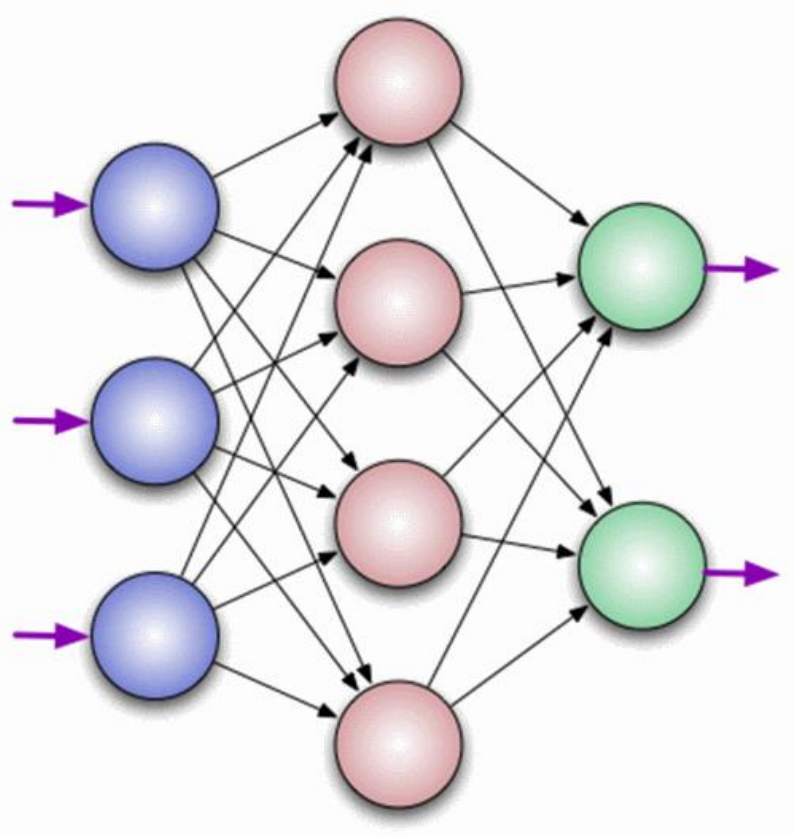

Рисунок 1.1 – Графічне зображення синапсів

Нейрон - це обчислювальна одиниця, яка отримує інформацію, виконує над нею прості обчислення і передає її далі. Вони діляться на три основні типи: вхідний (синій), прихований (червоний) і вихідний (зелений). Також існують нейрони зміщення і контекстний нейрон. У тому випадку, коли нейромережа складається з великої кількості нейронів, вводять термін шару. Відповідно, є

вхідний шар, який отримує інформацію, n прихованих шарів (зазвичай їх не більше 3), які її обробляють і вихідний шар, який виводить результат. У кожного з нейронів є 2 основні параметри: вхідні дані (input data) і вихідні дані (output data). У разі вхідного нейрона: input = output. В інших, в поле input потрапляє сумарна інформація всіх нейронів з попереднього шару, після чого, вона нормалізується за допомогою функції збудження (f (x)) і потрапляє в поле output.

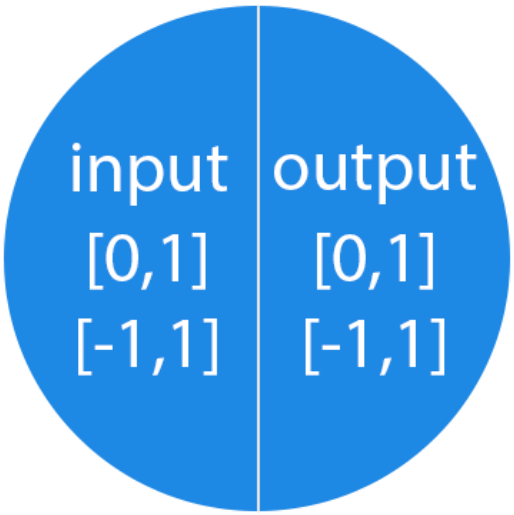

Рисунок 1.2 – Будова нейрону

Нейрони оперують числами в діапазоні [0,1] або [-1,1]. Якщо числа виходять с цього діапазону треба поділити 1 на це число. Цей процес називається нормалізацією, і він дуже часто використовується в нейронних мережах.

Синапс - це зв'язок між двома нейронами. У синапсів є 1 параметр - *вага*. Завдяки йому, вхідна інформація змінюється, коли передається від одного нейрона до іншого. Припустимо, є 3 нейрона, які передають інформацію наступному. Тоді у нас є 3 ваги, відповідні кожному з цих нейронів. У того нейрона, у якого вага буде більше, та інформація і буде домінуючою в наступному нейроні. Сукупність вагів нейронної мережі (або матриця вагів) - це своєрідний мозок всієї системи. Саме завдяки цим вагам, вхідна інформація

обробляється і перетворюється в результат. Під час ініціалізації нейронної мережі, ваги розставляються в випадковому порядку.

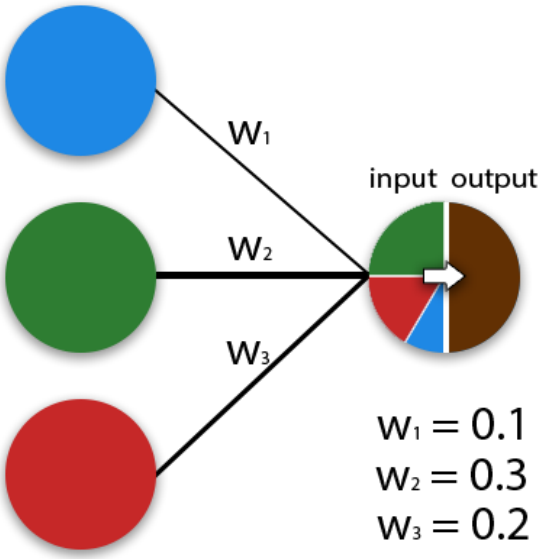

Рисунок 1.3 – Будова синапсу

1.2.2 Функції збудження

Функція збудження - це спосіб нормалізації вхідних даних. Тобто, якщо на вході є велике число, пропустивши його через функцію збудження, отримаємо вихід в потрібному нам діапазоні. Функцій збуджень досить багато тому ми розглянемо найосновніші: лінійну, сигмоїд (логістична) і гіперболічний тангенс. Головні їх відмінності - це діапазон значень.

- *Лінійна функція*

$$f(x) = x$$

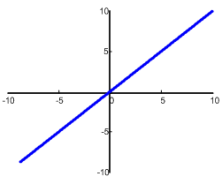

Рисунок 1.4 – Формула та графік лінійної функції

Ця функція майже ніколи не використовується, за винятком випадків, коли потрібно протестувати нейронну мережу або передати значення без перетворень[3].

- *Ступінчаста функція*

Одна з найпростіших видів. Вихідне значення нейрона – 0 або 1. Якщо сума більше значення параметру, то вихід – 1, якщо менше – 0.

Графік показано на малюнку.

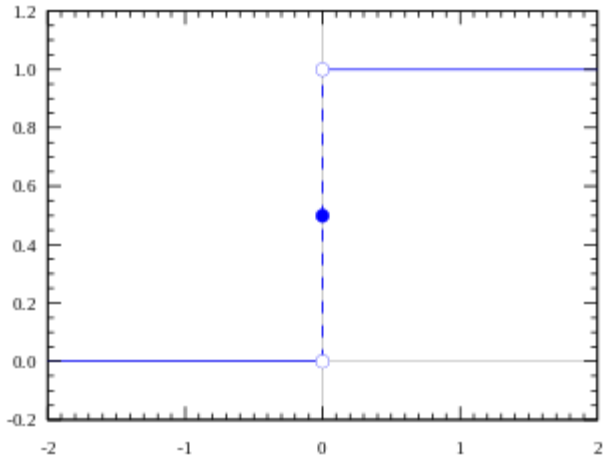

Рисунок 1.5 – Графік ступінчастої функції

Якщо у користувача буде один клас класифікації і два варіанта 0 або 1, то така функція буде добре працювати. В іншому випадку при активації більшого числа нейронів користувач не зрозуміє до якого класу підходить досліджуваний об’єкт[3].

- *Сигмоїд*

$$f(x) = \frac{1}{1+e^{-x}}$$

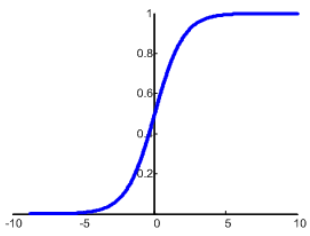

Рисунок 1.6 – Функція та графік сигмоїдної функції

Це найпоширеніша функція збудження, її діапазон значень [0,1]. Саме на ній показано більшість прикладів в мережі, також її іноді називають логістичною функцією. Відповідно, якщо в нашому випадку присутні негативні значення (наприклад, акції можуть йти не тільки вгору, але і вниз), то нам знадобитися функція яка оброблює і негативні значення[3].

- *Гіперболічний тангенс*

$$f(x) = \frac{e^{2x} - 1}{e^{2x} + 1}$$

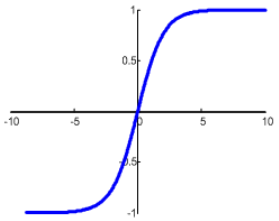

Рисунок 1.7 – Функція та графік функції гіперболічного тангенсу

Має сенс використовувати гіперболічний тангенс, тільки тоді, коли наші значення можуть бути і негативними, і позитивними, тобто як діапазон функції [-1,1]. Але використовувати цю функцію тільки для позитивних значень недоцільно тому що це значно погіршить результати нашої нейромережі.

1.2.3 З’єднання та ваги

Мережа (network) складається зі з'єднань (connection), кожне з яких передає вихід нейрону *i* до входу нейрону *j*. В цьому сенсі *i* є попередником (predecessor) *j*, а *j* є наступником (successor) *i* Кожному з'єднанню призначено вагу (weight) w_{ij}.

1.2.4 Функція поширення

Функція поширення (propagation function) обчислює вхід $p_j(t)$ до нейрону *j* з виходів $o_i(t)$ нейронів-попередників, і зазвичай має вигляд:

$$p_j(t) = \sum_i o_i(t) w_{ij} \quad (1.1)$$

1.2.5 Правило навчання

Правило навчання (learning rule) — це правило або алгоритм, який змінює параметри нейронної мережі, щоби заданий вхід до мережі видавав придатний вихід. Цей процес навчання зазвичай полягає в зміні ваг та порогів змінних мережі[6].

1.2.6 Нейронні мережі як функції

Нейромережеві моделі можна розглядати як прості математичні моделі, що визначають функцію $f : X \to Y$ (1.2) , або розподіл над X, або над X та Y. Іноді моделі тісно пов'язують з певним правилом навчання. Поширене використання фрази «модель штучної нейронної мережі (ШНМ)» насправді є визначенням класу таких функцій (де членів цього класу отримують варіюванням параметрів, ваг з'єднань, або особливостей архітектури, таких як число нейронів або їхня зв'язність)[6].

З математичної точки зору, нейромережеву функцію *f(x)* визначають як композицію інших функцій $g_i(x)$, які може бути розкладено далі на інші функції. Це може бути зручно представляти як мережеву структуру, де стрілки зображують залежність між функціями. Широко вживаним способом компонування є нелінійна зважена сума, де $f(x) = K\left(\sum_i w_i g_i(x)\right)$(1.3), де К (*функція збудження*) є визначеною наперед функцією, такою як гіперболічний тангенс, або сигмоїдна функція, або нормована експоненційна функція, або випрямляльна функція[10]. Важливою характеристикою функції збудження є те, що вона забезпечує плавний перехід при зміні значень входу, тобто, невелика зміна входу призводить до невеликої зміни виходу. Наведене нижче розглядає набір функцій g_i як вектор $g = (g_1, g_2, \ldots, g_n)$.

1.2.7 Навчання

Найбільше зацікавлення нейронними мережами викликала можливість навчання. Для заданої конкретної задачі для розв'язання та класу функцій F навчання означає використання набору спостережень для знаходження $f^* \in F$ яка розв'язує цю задачу в певному оптимальному сенсі.

Це тягне за собою визначення такої функції витрат (*cost function*) $C : F \to \mathbb{R}$(1.4), що, для оптимального розв'язку $f^*, C(f^*) \leq C(f) \forall f \in F$(1.5), тобто, жоден розв'язок не має витрат, менших за витрати оптимального розв'язку[10].

Функція витрат C є важливим поняттям у навчанні, оскільки вона є мірою того, наскільки далеким є певний розв'язок від оптимального розв'язку задачі, яку потрібно розв'язати. Алгоритми навчання здійснюють пошук простором розв'язків, щоби знайти функцію, яка має найменші можливі витрати.

Для тих застосувань, де розв'язок залежить від даних, витрати обов'язково мусять бути функцією від спостережень, бо інакше модель не матиме зв'язку з даними. Їх часто визначають як статистику, для якої може бути зроблено лише наближення. Як простий приклад, розгляньмо задачу знаходження моделі *f*, яка зводить до мінімуму $C = E\left[(f(x) - y)^2\right]$(1.6)для пар даних *(x, y)*, що витягають з певного розподілу $\mathcal{D}$. В практичних ситуаціях ми матимемо лише *N* зразків з $\mathcal{D}$, і, відтак, для наведеного вище прикладу ми будемо зводити до мінімуму лише $\hat{C} = \frac{1}{N} \sum_{i=1}^{N} (f(x_i) - y_i)^2$(1.7). Таким чином, витрати зводяться до мінімуму над вибіркою з даних, а не над усім розподілом[10].

Коли $N \to \infty$, мусить застосовуватися якийсь різновид інтерактивного машинного навчання, в якому витрати знижуються з кожним побаченим зразком. І хоча інтерактивне машинне навчання часто застосовують за

незмінного $\mathcal{D}$, найкориснішим воно є у випадку, коли цей розподіл повільно змінюється з часом. В нейромережевих методах якісь різновиди інтерактивного машинного навчання часто застосовують для скінченних наборів даних.

Навіть коли можливо визначити функцію витрат , часто використовують конкретні витрати (функцію витрат), або через те, що вони мають бажані властивості (такі як опуклість), або через те, що вони природно виникають з певного формулювання задачі (наприклад, у ймовірнісному формулюванні як обернені витрати можна використовувати апостеріорну ймовірність моделі). Кінець кінцем, функція витрат залежить від задачі.

В загальному розумінні навчання відбувається в 2 етапи – пряме та зворотне розповсюдження помилки. Відповідно при прямому розповсюдженню помилки робиться передбачення результату, а при зворотному мінімізується похибка між відповіддю та передбаченим результатом. Часткову похідну помилки можна обчислити по вагам, а диференціали вказують на вклад ваги на загальну помилку. Загальну помилку можна обчислити в вигляді різниці фактичного значення з набору даних та отриманим результатом.

1.3 Постановка задачі

У роботі ставляться такі задачі:

- провести аналіз алгоритмів інтерполяції зображень;
- вивчити модель алгоритму штучного інтелекту – генеративної змагальної мережі;
- створити інтелектуальну систему покращення якості масштабованих зображень для відновлення до оригінального розміру;
- проаналізувати різницю між різними алгоритмами інтерполяції та роботою генеративної змагальної мережі.

2 ГЕНЕРАТИВНІ ЗМАГАЛЬНІ МЕРЕЖІ ТА МЕТОДИ ПОКРАЩЕННЯ ЗОБРАЖЕНЬ

2.1 Генеративні змагальні мережі

2.1.1 Опис та складові

Одним з варіантів штучної нейронної мережі є генеративна змагальна мережа.

Генеративна змагальна мережа(ГЗМ) в загальному розумінні – це клас алгоритмів штучного інтелекту, що використовуються в машинному навчанні без учителя, які реалізовані системою двох штучних нейронних мереж, що змагаються одна з одною в рамках гри з нульовою сумою. 2014-го року їх запровадив Ян Гудфелоу. Методика дозволяє створювати фотографії, що мають вигляд реальних та мають багато реалістичних елементів. Ступінь реалістичності залежить від натренованості мережі в цілому[1].

Генеративні змагальні мережі використовують у цифровій фотографії для творення фотореалістичних зображень з метою візуалізації дизайнів інтер’єру, промислового дизайну, предметів одягу, 3-д моделей в комп’ютерній графіці та багато чого іншого. Також використовуються для покращення якості нечітких зображень, таких як астрономічні зображення зірок, віддалених галактик, фотографій зйомки систем стеження з поганою якістю. Від деяких існуючих алгоритмів та методів обробки зображень за допомогою фільтрів вони відрізняються в більшості випадків тим що вони використовують допоміжні текстури та генеровані зображення для заповнення чи заміщення частин зображень[1]. В першу чергу від таких систем вимагалося створення реалістичних текстур, а не покращення деталізації. Чим вища роздільна здатність використовуваних зображень тим краще результуюча якість. Також варто зазначити що такі мережі використовують такі ІТ-гіганти, як Google та Facebook.

Загальний метод роботи ГЗМ наступний – одна мережа(генератор) генерує кандидатів, а друга(дискримінатор) їх оцінює. Мережа навчається будувати відповідності з латентного простору до певного розподілу даних, а дискримінаційна мережа розрізняє представників справжнього розподілу даних та кандидатів, вироблених генератором. Мета тренувальної мережі – збільшення частоти помилок дискримінаційної мережі. Генератор є деконволюційною нейронною мережею, а дискримінатор – згортковою нейронною мережею. Заздалегідь відомий набір даних використовують як початкові навчальні данні для дискримінатора. Навчання дискримінатора передбачає забезпечення його зразками з набору даних, доки він не досягне певного рівня точності. Генератор на початку отримує випадково відібрані дані із заздалегідь визначеного латентного простору за допомогою багатовимірного нормального розподілу. Після цього зразки, синтезовані генератором, оцінюються дискримінатором. Метод зворотного поширення помилки застосовується в обох мережах так що генератор створює кращі зображення, тоді як дискримінатор стає більш кваліфікованим при визначенні синтезованих зображень.

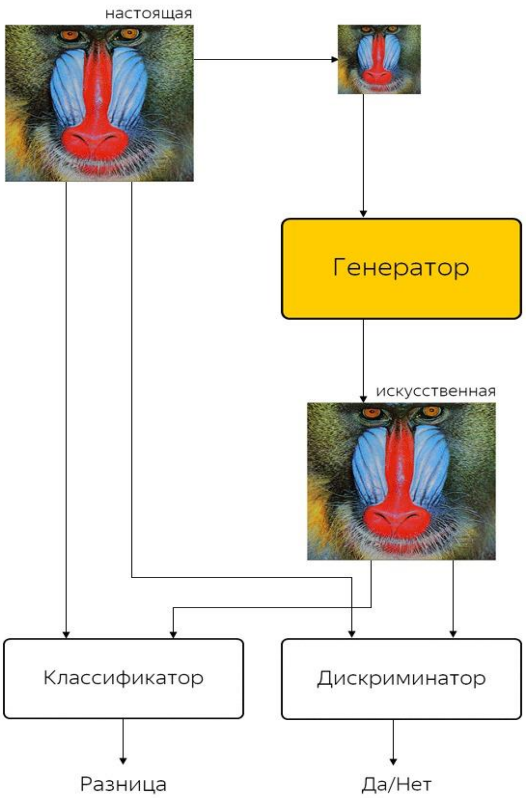

Рисунок 2.1 – Загальна будова генеративної змагальної мережі

Ідея вивести моделі в конкурентному середовищі була запропонована Лі, Гаучі та Гросом 2013 року. Метод використовується для висновків поведінки, що називається навчанням по Тюрінгу (Turing Learning), оскільки цей параметр схожий на тест Тюрінга. Навчання по Тюрінгу є узагальненням генеративної змагальної мережі. У них можуть розглядатись і моделі відмінні від нейронних мереж. Крім того, дискримінаторам дозволяється впливати на процеси, з яких отримані набори даних, що робить їх активними учасниками, як у тесті Тюрінга. Ідею змагального навчання можна знайти й у більш ранніх роботах, таких як стаття Шмідхубера 1992 року.

Навчання мережі складається з декількох етапів:

- По-перше оброблюється зображення високої роздільної здатності. Результатом є зображення низької роздільної здатності з пониженою частотою дискретизації. Тобто набір тренувальних даних має образи низького та високого розширення.

- Далі використовується генератор, що збільшить роздільну здатність зображень.
- І в кінці процесу використовується дискримінатор для того щоб розрізнити зображення високої роздільної здатності та покращені зображення за допомогою генератору.

2.1.2 Функції втрат

Функції втрат важливі для створення та ефективної роботи генеративної змагальної мережі. Існують наступні два типи: функція сприйняття та функція втрати змагальності[5].

Функція сприйняття обчислюється як зважена сума функції втрат змісту і функції втрат змагальності. Вона потрібна для того щоб мережа віддала перевагу справжньому зображенню, а не згенерованому[5].

Формула функції сприйняття:

$$l^{SR} = l_X^{SR} + 10^{-3} l_{Gen}^{SR} \quad (2.1)$$

Функція втрати змагальності обчислюється за ймовірністю того чи є згенероване зображення високої роздільної здатності. Обчислюється за наступною формулою:

$$l_{Gen}^{SR} = \sum_{n=1}^{N} -\log D_{\theta_D}(G_{\theta_G}(I^{LR})) \quad (2.2)$$

Щоб зберегти схожість сприйняття замість подібності по пікселям використовують втрату контенту. Вона обчислюється як евклідова відстань між ознаками початкового зображення та згенерованого на основі шарів попередньо натренованої VGG мережі за допомогою наступної формули[5]:

$$l_{VGG}^{SR} = \frac{1}{W_{i,j} H_{i,j}} \sum_{x=1}^{W_{i,j}} \sum_{y=1}^{H_{i,j}} (\phi_{i,j}(I^{HR})_{x,y} - \phi_{i,j}(I^{SR})_{x,y})^2 \quad (2.3)$$

2.1.3 Пікове співвідношення сигналу до шуму

Пікове співвідношення сигналу до шуму(англ. PSNR) є інженерним терміном, що означає співвідношення між максимумом можливого значення сигналу та потужністю шуму, що спотворює значення сигналу.

Найчастіше використовується для вимірювання рівня спотворень при стисненні зображень. Визначити його найпростіше через середньоквадратичне відхилення(MSE(англ. Mean square error)), яке для двох монохромних зображень *I* та *K* розміру *mxn*, одне з яких вважається зашумленими наближенням іншого обчислюється за формулою[4]:

$$MSE = \frac{1}{mn} \sum_{i=0}^{m} \sum_{j=0}^{n} |I(i,j) \quad K(i,j)|^2 \tag{2.4}$$

PSNR визначається за формулою:

$$PSNR = 10 \log_{10} \left(\frac{MAX_I^2}{MSE} \right) = 20 \log_{10} \left(\frac{MAX_I}{\sqrt{MSE}} \right) \tag{2.5}$$

2.2 Методи покращення зображень за допомогою інтерполяції

Інтерполяція зображень необхідна при проектуванні інформаційних систем що використовують функції покращення зображень чи зміни їх розмірів для задоволення потреб вихідного дисплею або випадків коли швидкість передачі сигналу недостатня. Передача версії зображення низької якості є більш ефективним рішенням, але і процедура апроксимації до якості оригіналу в більшості випадків необхідна, оскільки користувач може захотіти переглянути і повноцінну версію того ж самого зображення.

В цьому підрозділі будуть розглянуті деякі з алгоритмів інтерполяції зображень. Деякі з них є більш швидкими та вимагають низьких витрат зі сторони технічних приладів, інші навпаки мають високу якість, але і низьку швидкодію. Всі вони широко розповсюджені, і мають свою область дії. Так, наприклад методі, що орієнтовані на мобільні додатки та працюють в реальному часі повинні задовольняти невеликі витрати і використовувати невеликий об’єм пам’яті.

2.2.1 Метод найближчого сусіда

Алгоритм найближчого сусіда один із найпростіших методів інтерполяції, швидко виконується, але може видавати неоптимальні рішення. Обчислювальна складність алгоритму – O(n2). Алгоритм має низьку складність, що дає високу швидкодію, але якість вихідного зображення низька, спостерігається ефект згладжування.

Результатом виконання алгоритму є маршрут, приблизно на 25% довший від оптимального. Одним з евристичних критеріїв оцінки рішення є правило: якщо шлях, пройдений на останніх кроках алгоритму, зіставний зі шляхом, пройденим на початкових кроках, то можна умовно вважати знайдений маршрут прийнятим, інакше, імовірно, існують кращі рішення. Інший варіант оцінки рішення полягає у використанні алгоритму нижньої граничної оцінки[13].

Алгоритм найближчого сусіда починається в довільній точці та поступово відвідує кожну найближчу точку, яка ще не була відвідана. Пункти обходу плану послідовно включаються до маршруту, а кожен черговий пункт, що включається до маршруту, повинен бути найближчим до останнього вибраного пункту серед усіх інших, ще не включених до складу маршруту. Завершується алгоритм коли відвідані всі точки, а остання точка з’єднується з першою. Вхідні дані: множина точок V розмірністю N. Вихідні дані: маршрут T, що складається з послідовності відвідування точок множини V[13].

Кроки алгоритму:

- довільно обрати поточну точку;
- знайти найкоротше ребро, що сполучає поточну точку з досі ще не відвіданою точкою V;
- зробити точку V поточною;
- позначити точку V, як відвідану;
- коли всі точки розмірності N відвідані, припинити пошук маршруту;

- перейти до другого кроку.

2.2.2 Білінійна інтерполяція

Білінійна інтерполяція – узагальнення лінійної інтерполяції для функції двох змінних. Ідея в тому, що проводиться лінійна інтерполяція по одній осі, а потім по іншій осі. Інтерполюючі значення обчислюються на основі середньозваженого двох точок, а ваги обернено пропорційні до відстані від вхідних точок до точки, яку інтерполюють. Алгоритм аналізує квадрат 2x2 відомих пікселів, які розташовані навколо невідомого[14].

Цей алгоритм інтерполяції використовує більше інформації ніж метод найближчого сусіда, і звичайно вихідне зображення виходить більш високої якості, Збільшується складність, час виконання, та використання ресурсів пристрою що його виконує. Ефект згладжування зникає, але зображення стає більше розмитим.

Функція має вигляд:

$$F(x,y) = b_1 + b_2 \cdot x + b_3 \cdot y + b_4 \cdot x \cdot y \quad (2.6)$$

2.2.3 Бікубічна інтерполяція

Бікубічна інтерполяція схожа на білінійну, але на відміну від лінійної метод інтерполює масив 4x4 відомих пікселів, які знаходяться навколо невідомого. Але так як ці пікселі знаходяться на різній відстані від невідомого то вони мають різну вагу: ті що знаходяться близько – більшу, а ті що далі – менше. Так як масив даних більший, то і швидкодія зменшується, але і результат здебільшого кращий. Зображення мають більш різкий вигляд на відміну від білінійного. Вважається одним із компромісних алгоритмів інтерполяції в комп’ютерній графіці і став доволі популярним[14].

2.2.4 Метод Ланцоша

Метод Ланцоша(або фільтр Ланцоша) – спосіб математичної обробки рядів даних. Використовується для інтерполяції функції між заданими

точками, або в якості фільтра нижніх частот. Найбільш поширено використовується в обробці зображень та для зміни розміру їх роздільної здатності – передискретизації. В процесі роботи алгоритму використовується 36 пікселів для інтерполяції. Вихідне зображення позбавлене ефекту розмиття, але можуть з’явитись деякі артефакти, такі як «звін». Тобто це спотворення полягає в появі навколо контрастних переходів яскравості вузьких контрастних ореолів[13].

Фільтр Ланцоша отримується помножуючи вікно Ланцоша на sinc(x)-функцію.

Формула вікна Ланцоша має вигляд:

$$L(x) = \begin{cases} \operatorname{sinc}(x)\operatorname{sinc}\left(\frac{x}{a}\right), & \text{при } -a < x < a, \\ 0, & \text{иначе} \end{cases} \quad (2.7)$$

Функція sinc(x) має вигляд:

$$sinc(x) = \begin{cases} \frac{\sin(\pi x)}{\pi x}, x \neq 0 \\ 1; x = 0 \end{cases} \quad (2.8)$$

Використання цього методу дозволяє отримати зображення високої якості, але його використання є досить ресурсоїмким.

3 РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ СИСТЕМИ

3.1 Вимоги до середовища розробки ПЗ.

Для розроблення ПЗ було використано мову програмування Python, ОС – Windows 10, також використовувались допоміжні файли .ipunb системи JupyterNotebook.

Пояснити вибір мови програмування доволі просто, тому як вона є найпоширенішою для задач машинного навчання, обробки та класифікації даних. Також вона не дуже складна для вивчення початківцям. Програми створені мовою Python можна використовувати незалежно від платформи, в таких системах як: windows, linux, android тощо. Написаний код також не дуже заплутаний і легко читається на відміну від деяких подібних мов програмування. Але це не всі причини, до безумовних плюсів можна віднести також велику кількість допоміжних бібліотек, які створені безліччю людей по всьому світі. Створені бібліотеки легко підключити до своєї програми командами імпортування, і мати можливість використовувати вже розроблені функції. Так, наприклад, було використано бібліотеку для глибокого навчання *Keras*. За її допомогою можна швидко створювати і обчислювати прототипи, а також має прискорення графічного процесора. А також *Scikit-image* для обробки зображень.

3.2 Реалізація програмної частини.

По-перше генеративну змагальну мережу потрібно натренувати. Запускати генератор та дискримінатор навчатися одночасно без тренування генератора не можна, тому що в такому випадку систему навчити буде майже не можливо. В роботі був використаний метод оптимізації – Adam з параметром навчання 0.0001. Розмір пакету даних – 64, а кількість ітерацій – 1000.

Архітектура генеративної змагальної мережі зображена на рисунку 3.1.

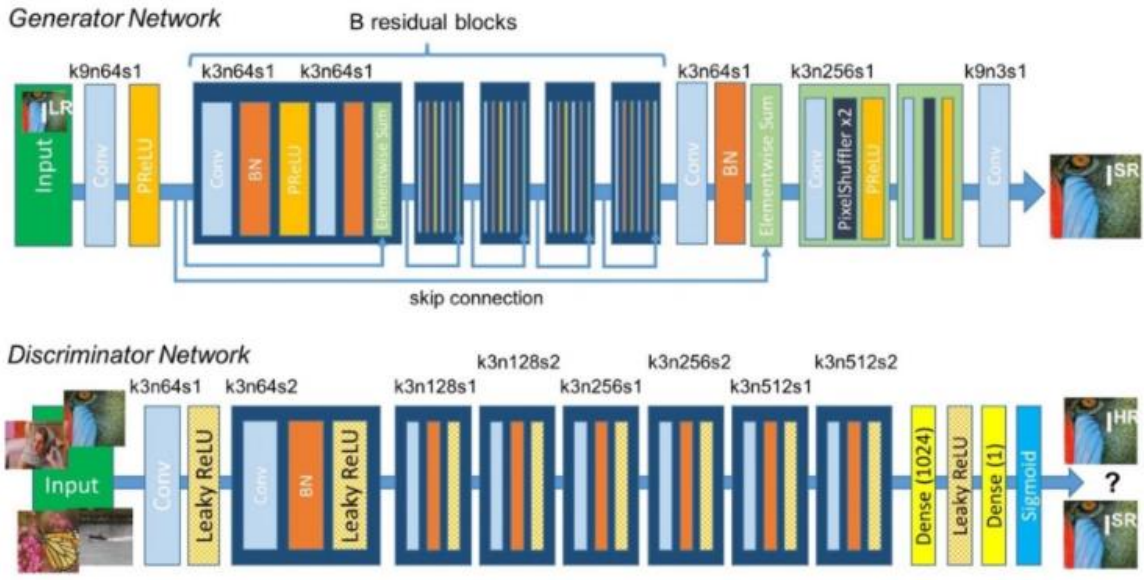

Рисунок 3.1 – Архітектура генеративної змагальної мережі

Її складові наступні – 16 залишкових блоків, k3n64s1(тобто 3 ядра, 64 канали з кроком 1), функція PReLU(функція активації, допомагає адаптуватися у випадки негативних коефіцієнтів), перцептивна функція втрат, що складається зі змагальної функції втрат і функції втрат змісту[11].

Обраний датасет зображень COCO data set 2017, кількість зображень 800. Всі зображення однакової роздільної здатності – 384x384. Це так звані дані для порівняння, вважаємо їх за ідеальні для інтерполяції. Для тренування мережі також потрібно ці зображення зменшити, для більш ефективної роботи можна їх зменшити по висоті та ширині в 4(або в 4-крат) рази відповідно. Під час навчання вони порівнюватимуться з оригіналами[9].

Принцип мережі наступний:

- обробка оригіналу для отримання зменшених зображень;
- зображення 96x96 подаються до генератора;
- масштабування генератором зменшених зображень та видання на вихід збільшених варіантів;
- використовується дискримінатор, щоб порівняти ці два зображення;
- дискримінатор та генератор навчаються методом оберненого поширення помилки функції втрат.

3.3 Результати роботи ПЗ та тестування.

Було побудовано і навчено нейронну мережу за допомогою мови програмування Python. Вона генерувала збільшене в 4 рази зображення, яке до цього спеціальним способом за допомогою алгоритму бікубічної інтерполяції зменшувалось для процедури навчання мережі.

Для тестування роботи мережі можна порівняти зображення оброблені за допомогою наступних методів: інтерполяція за допомогою нейронної мережі, методом бікубічної, та лінійної інтерполяції, а також методу Ланцоша і методу найближчого сусіда.

Спочатку обирається зона для масштабування, все інше обрізається, створюється копія цієї частини, копія зменшується, а далі зображення за допомогою обраного методу інтерполюється до розміру оригіналу. Тож далі слід відобразити деякі з прикладів зображень які були використані, оброблені, та інтерпольовані за допомогою генеративної змагальної системи.

Спочатку використаємо чорно-білий малюнок для тестування системи та порівняння алгоритмів на наявні артефакти. На рисунку 3.2 зображено оригінал зображення, а на рисунку 3.3 його обрізані інтерпольовані варіанти.

Рисунок 3.2 – Чорно-білий оригінал зображення №1

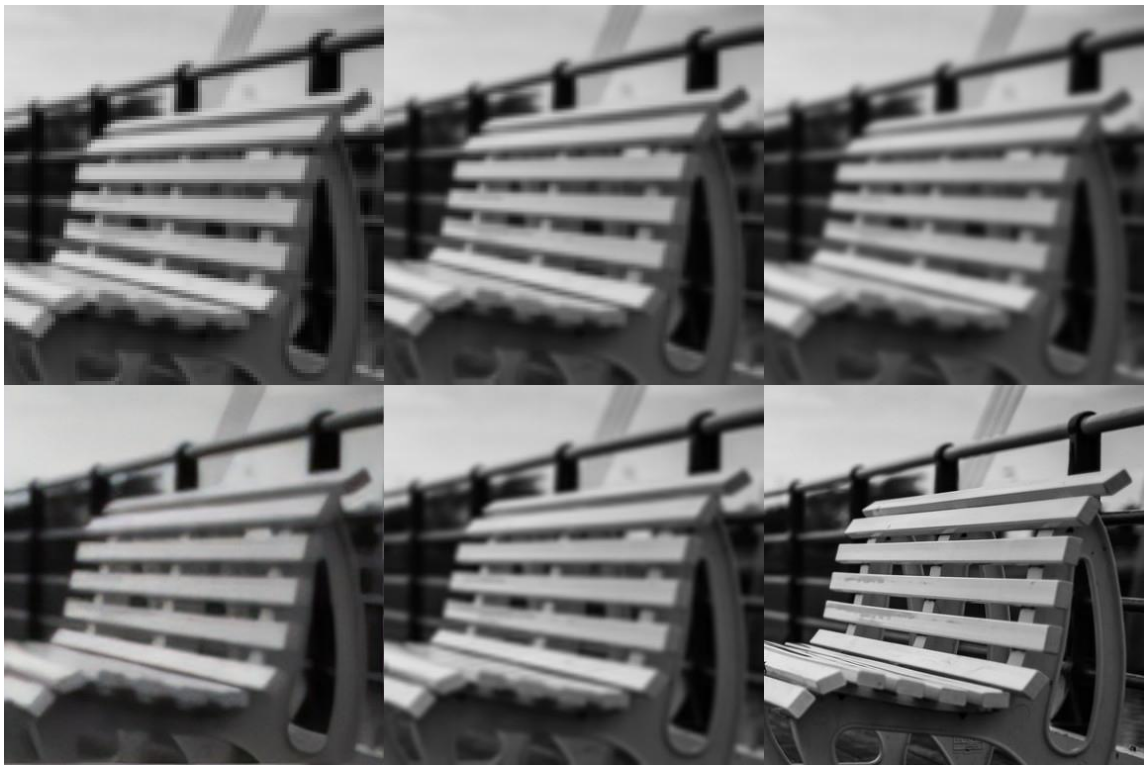

Рисунок 3.3 – Оброблене чорно-біле зображення №1

Тут і далі оброблені зображення представлення по такому порядку зліва направо та зверху вниз відповідно: метод найближчого сусідів, білінійна інтерполяція, бікубічна інтерполяція, фільтр Ланцоша, генеративна змагальна мережа та оригінал.

Як бачимо на прикладі першого зображення при використанні методу найближчого сусіда чіткість зображення найнижча, спостерігаються невеликі кубики, а в випадку білінійної та бікубічної інтерполяції спостерігається ефект розмиття. Фільтр Ланцоша робить зображення більш чітким, але спостерігається спотворення рівня кольору. Генеративна змагальна мережа показала себе найкраще, завдяки використаній перцептивній функції оптимізації зображення найближче відповідає оригіналу.

Розглянемо інший випадок чорно-білого зображення з більшою кількістю деталей. На рисунке 3.4 бачимо оригінал, а на рисунке 3.5 відповідно його оброблені версії.

Рисунок 3.5 – Чорно-білий оригінал зображення №2

Рисунок 3.6 – Оброблене чорно-біле зображення №2

При розгляді другого чорно-білого зображення з більшою кількістю деталей спостерігаються такі ж артефакти як і в першому випадку.

Розглянемо також на прикладі декілька кольорових зображень. Оригінали відповідно на рисунках 3.7 та 3.9, а їх інтерпольовані та оброблені версії на рисунках 3.8 та 3.10.

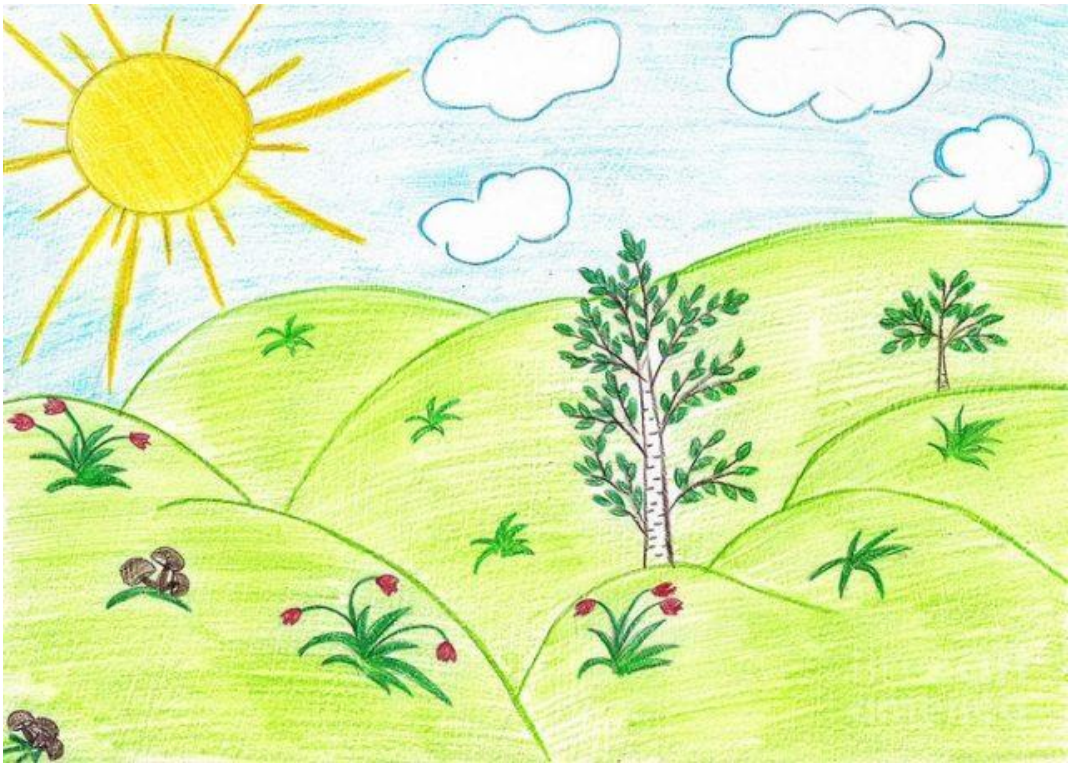

Рисунок 3.7 – Кольоровий оригінал зображення №3

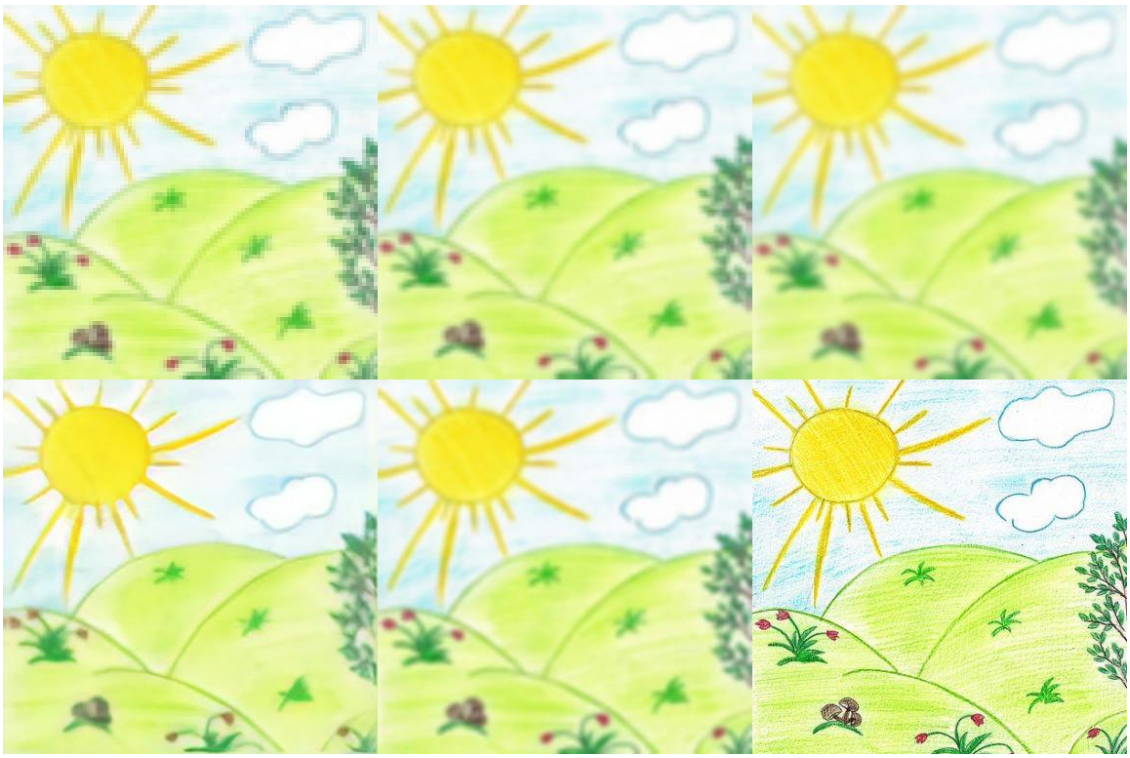

Рисунок 3.8 – Оброблене кольорове зображення №3

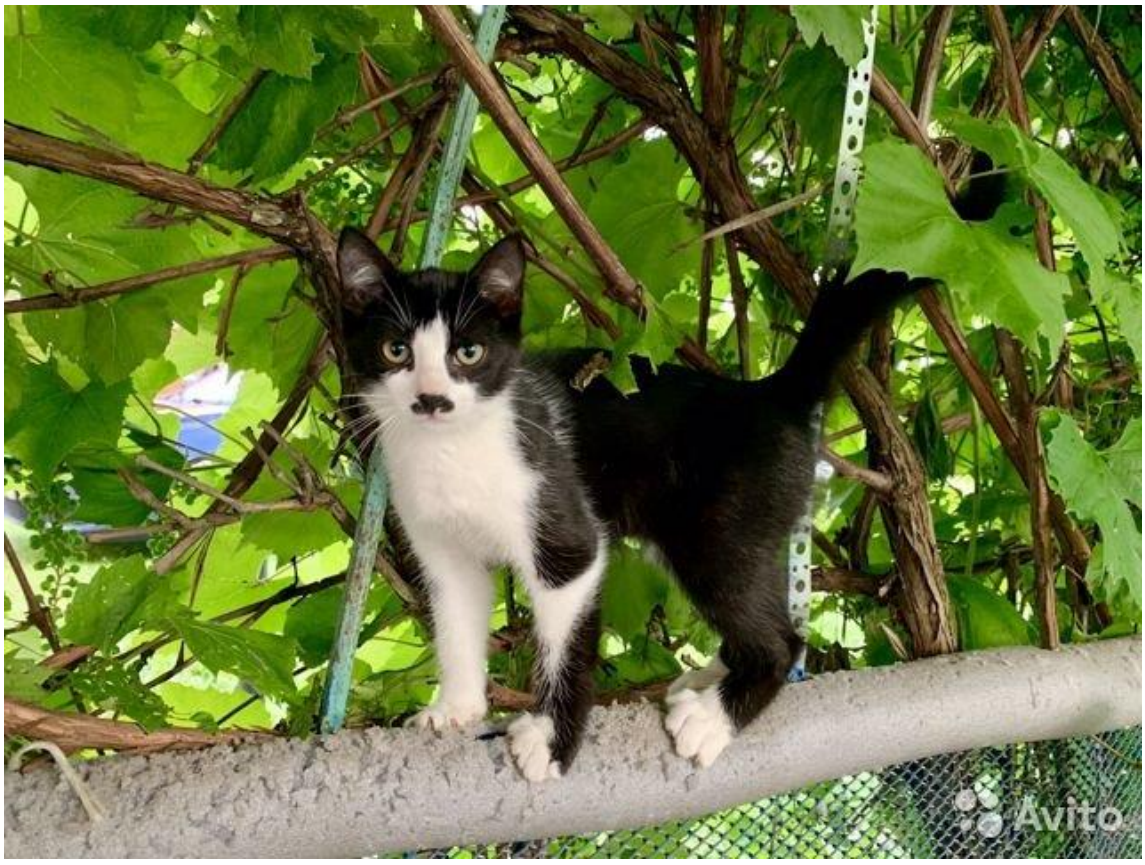

Рисунок 3.9 – Кольоровий оригінал зображення №4

Рисунок 3.10 – Оброблене кольорове зображення №4

З кольоровими зображеннями система показала себе трохи гірше ніж у випадку з чорно-білими зображеннями. На рисунку 3.8 бачимо що використання фільтру Ланцоша краще обробило зображення, деталі контурів більш чіткі, але все ж присутнє спотворення кольорів. На рисунку 3.10 фільтр Ланцоша майже однаковий з роботою нейронної мережі. Можна припустити що набір даних для тренування включав більше зображень з високою деталізацією, ніж з простою, як бачимо у випадку рисунка 3.8, тому і якість низько деталізованих оброблених зображень вийшла низькою. Також треба відмітити низький рівень натренованості мережі, в подальшому можна очікувати покращення.

ВИСНОВКИ

Результати показали, що генеративна змагальна мережа дуже добре справляється з обробкою зображень, і в випадках масштабування порівняно з методами інтерполяції в більшості випадків працює якісніше з точки зору людського сприйняття, хоча і містить деякі артефакти. Вона не спотворює кольори, а саме зображення не розмивається, як у випадку з такими методами як лінійна інтерполяція. Це можна пояснити тим що функцією втрат було обрано перцептивну функцію втрат, яка за своїм змістом працює з деякими особливостями подібних до людського зорового сприйняття. Однак, не дивлячись на кращі результати також варто зазначити що час обробки зображень набагато більший ніж при використанні методів інтерполяції, і в загальному розумінні використання нейронних мереж при обробці зображень в реальному часі при використанні на приладах з невеликими потужностями майже неможливий. Тобто такий що потребує заздалегідь заданих завдань обробки.

При роботі з чорно-білими зображеннями система працює краще ніж з кольоровими. Окрім випадку низько-деталізованих кольорових зображень, що пов’язане з невеликим початковим датасетом для тренування мережі. Але треба відмітити, що при збільшенні часу навчання мережі також логічно допустити що збільшиться і якість згенерованих зображень без втрати змісту.

СПИСОК ЛІТЕРАТУРИ

1. Generative Adversarial Networks: Generate Images Using Keras, 2018 [Електронний ресурс] // Режим доступу: https://hub.packtpub.com/generative-adversarial-networks-using-keras/
2. Методы оптимизации нейронных сетей [Електронний ресурс] // Режим доступу: https://habr.com/ru/post/318970/
3. Cybenko, G.V. Approximation by Superpositions of a Sigmoidal function / Cybenko G.V. – New York: Springer International, 2006 – 314 с.
4. Alexey Dosovitskiy. Generating images with perceptual similarity metrics based on deep networks. In Advances in Neural Information Processing Systems / Alexey Dosovitskiy and Thomas Brox // pp. 658–666, 2016.
5. Lucas Theis. Generative image modeling using spatial LSTMs. In Advances in Neural Information Processing Systems / Lucas Theis, Matthias Bethge // pp. 1918– 1926, 2015.
6. William T Freeman. Example-based superresolution. IEEE Computer graphics and Applications / William T Freeman, Thouis R Jones, and Egon C Pasztor // 22(2):56–65, 2002
7. Akın Kazakçı. Conceptive artificial intelligence // Insights from design theory. In International Design Conference DESIGN 2014, pp. 1–16, 2014.
8. Rijndael mix columns [Electronic resource] // Wikipedia.- Mode of access: https://en.wikipedia.org/wiki/Rijndael_mix_columns
9. William T Freeman. Example-based superresolution. IEEE Computer graphics and Applications / William T Freeman, Thouis R Jones, and Egon C Pasztor // 22(2):56–65, 2002.
10. Mehdi Mirza. Conditional generative adversarial nets / Mehdi Mirza and Simon Osindero // arXiv:1411.1784, 2014.
11. Ватолин Д., Ратушняк А., Смирнов М., Юкин В. Методы сжатия данных. Устройство архиваторов, сжатие изображений и видео. / Ватолин Д. – Москва.: ДИАЛОГ-МИФИ, 2002. – 384 с.

12. Raymond Yeh. Semantic image inpainting with perceptual and contextual losses / Raymond Yeh, Chen Chen, Teck Yian Lim, Mark Hasegawa-Johnson, and Minh N Do // arXiv:1607.07539, 2016
13. Берг Й., Лёфстрём Й. Интерполяционные пространства. Введение. / Москва: Мир, 1980. – 264 с.
14. Ловягин А. М. Интерполяция. Энциклопедический словарь Брокгауза и Ефрона : в 86 т. (82 т. и 4 доп.). / Ловягин А. М. — СПб., 1890—1907 с.

ДОДАТОК А

Код програми:

```
# Підключаємі модулі
from keras.layers import Dense
from keras.layers.core import Activation
from keras.layers.core import Flatten
from keras.layers import Input
from keras.layers.normalization import BatchNormalization
from keras.layers.convolutional import UpSampling2D
from keras.layers.advanced_activations import LeakyReLU, PReLU
from keras.layers.convolutional import Conv2D, Conv2DTranspose
from keras.models import Model
from keras.layers import add

# Залишкові блоки
def res_block_gen(model, kernal_size, filters, strides):

    gen = model
    model = Conv2D(filters = filters, kernel_size = kernal_size, strides = strides,
    padding = "same")(model)
    model = BatchNormalization(momentum = 0.5)(model)
    # Використовуєм функцію Parametric ReLU
    model = PReLU(alpha_initializer='zeros', alpha_regularizer=None,
    alpha_constraint=None,
    shared_axes=[1,2])(model)
    model = Conv2D(filters = filters, kernel_size = kernal_size, strides = strides,
    padding = "same")(model)
    model = BatchNormalization(momentum = 0.5)(model)
```

```
        model = add([gen, model])

        return model

def up_sampling_block(model, kernal_size, filters, strides):

        model = Conv2D(filters = filters, kernel_size = kernal_size, strides = strides,
        padding = "same")(model)
        model = UpSampling2D(size = 2)(model)
        model = LeakyReLU(alpha = 0.2)(model)

        return model

def discriminator_block(model, filters, kernel_size, strides):

        model = Conv2D(filters = filters, kernel_size = kernel_size, strides = strides,
        padding = "same")(model)
        model = BatchNormalization(momentum = 0.5)(model)
        model = LeakyReLU(alpha = 0.2)(model)

        return model

# Архітектура мережі як показано на рисунку 3.1, а також
https://arxiv.org/pdf/1609.04802.pdf
class Generator(object):

        def __init__(self, noise_shape):
        self.noise_shape = noise_shape
```

```
def generator(self):
gen_input = Input(shape = self.noise_shape)
model = Conv2D(filters = 64, kernel_size = 9, strides = 1, padding =
"same")(gen_input)
model = PReLU(alpha_initializer='zeros', alpha_regularizer=None,
alpha_constraint=None,
shared_axes=[1,2])(model)
gen_model = model

# Використовуєм 16 залишкових блоків
for index in range(16):
model = res_block_gen(model, 3, 64, 1)
model = Conv2D(filters = 64, kernel_size = 3, strides = 1, padding =
"same")(model)
model = BatchNormalization(momentum = 0.5)(model)
model = add([gen_model, model])
# Використовуєм 2 блоки передискредитації(UpSampling)
for index in range(2):
model = up_sampling_block(model, 3, 256, 1)
model = Conv2D(filters = 3, kernel_size = 9, strides = 1, padding =
"same")(model)
model = Activation('tanh')(model)
generator_model = Model(inputs = gen_input, outputs = model)
return generator_model

# Архітектура мережі як показано на рисунку 3.1, а також
https://arxiv.org/pdf/1609.04802.pdf
class Discriminator(object):
```

```
        def __init__(self, image_shape):
        self.image_shape = image_shape
        def discriminator(self):
        dis_input = Input(shape = self.image_shape)
        model = Conv2D(filters = 64, kernel_size = 3, strides = 1, padding =
        "same")(dis_input)
        model = LeakyReLU(alpha = 0.2)(model)
        model = discriminator_block(model, 64, 3, 2)
        model = discriminator_block(model, 128, 3, 1)
        model = discriminator_block(model, 128, 3, 2)
        model = discriminator_block(model, 256, 3, 1)
        model = discriminator_block(model, 256, 3, 2)
        model = discriminator_block(model, 512, 3, 1)
        model = discriminator_block(model, 512, 3, 2)
        model = Flatten()(model)
        model = Dense(1024)(model)
        model = LeakyReLU(alpha = 0.2)(model)
        model = Dense(1)(model)
        model = Activation('sigmoid')(model)
        discriminator_model = Model(inputs = dis_input, outputs = model)
        return discriminator_model


from Network import Generator, Discriminator
import Utils_model, Utils
from Utils_model import VGG_LOSS
from keras.models import Model
from keras.layers import Input
from tqdm import tqdm
import numpy as np
```

```
import argparse
np.random.seed(10)
# Використовуєм 4-кратне зменшення, як було зазначено в розділі 3.2
downscale_factor = 4
#
image_shape = (384,384,3)

# Об’єднана мережа
def get_gan_network(discriminator, shape, generator, optimizer, vgg_loss):

    discriminator.trainable = False
    gan_input = Input(shape=shape)
    x = generator(gan_input)
    gan_output = discriminator(x)
    gan = Model(inputs=gan_input, outputs=[x,gan_output])
    gan.compile(loss=[vgg_loss, "binary_crossentropy"],
    loss_weights=[1., 1e-3],
    optimizer=optimizer)
    return gan

# Використовуєм стартові параметри за замовченням бібліотеки
def train(epochs, batch_size, input_dir, output_dir, model_save_dir,
number_of_images, train_test_ratio):

    x_train_lr, x_train_hr, x_test_lr, x_test_hr =
    Utils.load_training_data(input_dir, '.jpg', number_of_images,
    train_test_ratio)
    loss = VGG_LOSS(image_shape)
    batch_count = int(x_train_hr.shape[0] / batch_size)
```

```
shape = (image_shape[0]//downscale_factor,
image_shape[1]//downscale_factor, image_shape[2])
generator = Generator(shape).generator()
discriminator = Discriminator(image_shape).discriminator()
optimizer = Utils_model.get_optimizer()
generator.compile(loss=loss.vgg_loss, optimizer=optimizer)
discriminator.compile(loss="binary_crossentropy", optimizer=optimizer)
gan = get_gan_network(discriminator, shape, generator, optimizer,
loss.vgg_loss)
loss_file = open(model_save_dir + 'losses.txt' , 'w+')
loss_file.close()
for e in range(1, epochs+1):
print ('-'*15, 'Epoch %d' % e, '-'*15)
for _ in tqdm(range(batch_count)):
rand_nums = np.random.randint(0, x_train_hr.shape[0], size=batch_size)
image_batch_hr = x_train_hr[rand_nums]
image_batch_lr = x_train_lr[rand_nums]
generated_images_sr = generator.predict(image_batch_lr)
real_data_Y = np.ones(batch_size) -
np.random.random_sample(batch_size)*0.2
fake_data_Y = np.random.random_sample(batch_size)*0.2
discriminator.trainable = True
d_loss_real = discriminator.train_on_batch(image_batch_hr, real_data_Y)
d_loss_fake = discriminator.train_on_batch(generated_images_sr,
fake_data_Y)
discriminator_loss = 0.5 * np.add(d_loss_fake, d_loss_real)
rand_nums = np.random.randint(0, x_train_hr.shape[0], size=batch_size)
image_batch_hr = x_train_hr[rand_nums]
image_batch_lr = x_train_lr[rand_nums]
```

```
        gan_Y = np.ones(batch_size) - np.random.random_sample(batch_size)*0.2
        discriminator.trainable = False
        gan_loss = gan.train_on_batch(image_batch_lr, [image_batch_hr,gan_Y])
        print("discriminator_loss : %f" % discriminator_loss)
        print("gan_loss :", gan_loss)
        gan_loss = str(gan_loss)
        loss_file = open(model_save_dir + 'losses.txt' , 'a')
        loss_file.write('epoch%d : gan_loss = %s ; discriminator_loss = %f\n' %(e,
        gan_loss, discriminator_loss)
        )
        loss_file.close()
        if e == 1 or e % 5 == 0:
        Utils.plot_generated_images(output_dir, e, generator, x_test_hr, x_test_lr)
        if e % 500 == 0:
        generator.save(model_save_dir + 'gen_model%d.h5' % e)
        discriminator.save(model_save_dir + 'dis_model%d.h5' % e)

if __name__== "__main__":

        parser = argparse.ArgumentParser()
        parser.add_argument('-i', '--input_dir', action='store', dest='input_dir',
        default='./data/' ,
        help='Path for input images')
        parser.add_argument('-o', '--output_dir', action='store', dest='output_dir',
        default='./output/' ,
        help='Path for Output images')
        parser.add_argument('-m', '--model_save_dir', action='store',
        dest='model_save_dir', default='./model/' ,
        help='Path for model')
```

```
        parser.add_argument('-b', '--batch_size', action='store', dest='batch_size',
        default=64,
        help='Batch Size', type=int)
        parser.add_argument('-e', '--epochs', action='store', dest='epochs',
        default=1000 ,
        help='number of iteratios for trainig', type=int)
        parser.add_argument('-n', '--number_of_images', action='store',
        dest='number_of_images', default=1000 ,
        help='Number of Images', type= int)
        parser.add_argument('-r', '--train_test_ratio', action='store',
        dest='train_test_ratio', default=0.8 ,
        help='Ratio of train and test Images', type=float)
        values = parser.parse_args()

train(values.epochs,
values.batch_size,
values.input_dir,
values.output_dir,
values.model_save_dir,
values.number_of_images, values.train_test_ratio)
```