Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)
Кафедра Інформатики
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

ДОСЛІДЖЕННЯ РОБОТИ ЗГОРТКОВИХ НЕЙРОННИХ МЕРЕЖ ПРИ СТВОРЕННІ DEEPFAKE

(тема)

Виконав:
студент 2 курсу, групи ІНФм-23-2

Мєшков Д.М.
(прізвище, ініціали)

Спеціальності 122 Комп'ютерні науки
(код і повна назва спеціальності)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

Керівник доц. Вечірська І.Д.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ Кобилін О.А.
(підпис) (прізвище, ініціали)

2025 р.

Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)

Кафедра Інформатики
(повна назва)

Рівень вищої освіти другий (магістерський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____»_____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Мєшкову Дмитру Максимовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Дослідження роботи згорткових нейронних мереж при створенні DeepFake

затверджена наказом по університету від 25 листопада 2024 року № 1246Ст

2. Термін подання студентом роботи до екзаменаційної комісії ___ 25 грудня _____ 2024 р.

3. Вихідні дані до роботи науково-методична та науково-технічна література, матеріали конференцій, дані інтернет-мережі, мова програмування Python, середовище розробки Google Colab.

4. Перелік питань, що потрібно опрацювати в роботі
1. Огляд методів створення DeepFake.
2. Математична модель згорткових нейронних мереж.
3. Математична модель генеративних змагальних нейронних мереж.
4. Методи оптимізації згорткових нейронних мереж.
5. Аналіз ефективності різних архітектур нейронних мереж.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) актуальність проблеми аналізу ефективності роботи згорткових нейронних мереж при створенні DeepFake, постановка задачі, згенеровані зображення, архітектурні рішення для реалізації.

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	25.11.2024	
2	Аналіз завдання, підбір літератури	25.11.24-27.11.24	
3	Аналіз літератури з досліджуваної проблеми	27.11.24-28.11.24	
4	Аналіз технічних засобів	28.11.24-29.11.24	
5	Розробка методу	29.11.24-30.11.24	
6	Програмна реалізація	30.11.24-03.12.24	
7	Оформлення пояснювальної записки	03.12.24-05.12.24	
8	Перевірка на плагіат	27.12.2024	
9	Рецензування	30.12.2024	
10	Підготовка презентації та доповіді	01.12.2024	
11	Занесення роботи в електронний архів	07.01.2025	
12	Попередній захист кваліфікаційної роботи	07.01.2025	

Дата видачі завдання 25 листопада 2024 р.

Студент _____
(підпис)

Керівник роботи _____ доц. Вечірська І.Д.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ/ABSTRACT

Пояснювальна записка до кваліфікаційної роботи: 83 с., 43 рис., 41 джерело.

ЗГОРТКОВА НЕЙРОННА МЕРЕЖА, DEEPFAKE, КОМП'ЮТЕРНИЙ ЗІР, СИНТЕЗ ЗОБРАЖЕНЬ, РОЗПІЗНАВАННЯ ОБЛИЧЧА, ГЕНЕРАТИВНА ЗМАГАЛЬНА МЕРЕЖА, ШТУЧНИЙ ІНТЕЛЕКТ.

Об'єктом дослідження є робота згорткових нейронних мереж при створенні DeepFake.

Метою дослідження є аналіз ефективності застосування згорткових нейронних мереж для генерації реалістичних зображень та відео.

Використано методи глибокого навчання, комп'ютерного зору та обробки зображень для дослідження можливостей нейронних мереж у створенні DeepFake. Проведено експериментальне оцінювання точності та реалістичності генерованого медіаконтенту на основі різних моделей згорткових нейронних мереж. Досліджено метод генерації зображення за допомогою генеративних змагальних мереж.

У результаті дослідження проведено аналіз ефективності роботи різних моделей згорткових нейронних мереж при створенні DeepFake.

CONVOLUTIONAL NEURAL NETWORK, DEEPFAKE, COMPUTER VISION, IMAGE SYNTHESIS, FACE RECOGNITION, GENERATIVE ADVERSARIAL NETWORK, ARTIFICIAL INTELLIGENCE.

The object of research is the work of convolutional neural networks in creating DeepFake.

The aim of the research is to analyze the effectiveness of using convolutional neural networks to generate realistic images and videos.

The methods of deep learning, computer vision, and image processing were used to explore the possibilities of neural networks in the creation of DeepFake. An experimental evaluation of the accuracy and realism of the generated media content based on various CNN models was carried out. The image generation method using generative convolutional networks has been studied.

As a result of the study, an analysis of the effectiveness of various models of convolutional neural networks in the creation of DeepFake was carried out.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

DeepFake – технологія, що використовує штучний інтелект для створення реалістичних фейкових відео або зображень шляхом заміни обличча або маніпуляції з ним

CNN – Convolutional Neural Network (згорткова нейронна мережа)

GAN – Generative Adversarial Network (генеративна змагальна мережа)

PSNR – Peak Signal-to-Noise Ratio (пікове відношення сигнал/шум)

SSIM – Structural Similarity Index Measure (індекс структурної схожості)

LPIPS – Learned Perceptual Image Patch Similarity (навчена перцепційна схожість фрагментів зображення)

FID – Frechet Inception Distance (відстань інцепції)

IS – Inception Score (індекс інцепції)

FLE – Facial Landmark Error (помилка ключових точок обличчя)

ВСТУП

Все більше поширення отримують технології генерації зображення, зокрема ті, що в своїй роботі використовують алгоритми згорткових нейронних мереж [1]. Прикладом сфери, де ці нейронні мережі знайшли своє використання є DeepFake [2].

DeepFake – це технологія, яка використовує штучний інтелект, зокрема глибокі нейронні мережі, для створення реалістичних, але фальшивих відео або зображень. Основою технології є глибоке навчання та генеративні змагальні мережі (GAN), які дозволяють моделі навчатися на реальних зображеннях і відео та створювати схожі матеріали [3].

Використання DeepFake найчастіше полягає у заміні обличчя людини в відео чи на фото на обличчя іншої людини, але технологія також дозволяє змінювати голос або рухи об’єктів [4].

Згорткові нейронні мережі (CNN) є одним із ключових інструментів в області глибинного навчання, зокрема у завданнях комп’ютерного зору та обробки зображень [5]. Основним принципом їх роботи є використання згорткових операцій для виявлення особливостей зображень на різних рівнях деталізації, що дозволяє ефективно розпізнавати складні патерни та структури. Саме завдяки цим якостям вони знайшли широке застосування у створенні DeepFake.

Актуальність дослідження полягає в тому, що з розвитком технологій створення DeepFake з’являється необхідність визначати ефективність роботи нейронних мереж з метою ідентифікації найкращих методів генерації зображень.

1 ОГЛЯД ОСНОВНИХ МЕТОДІВ СТВОРЕННЯ DEEPFAKE

1.1 Створення DeepFake

Технологія DeepFake базується на алгоритмах штучного інтелекту, які можуть створювати реалістичні підробки відео, аудіо та зображень. Вона дозволяє замінювати обличчя, голос або навіть повністю змінювати контент, створюючи ілюзію того, що людина говорить або робить те, чого насправді не відбувалося. Цей термін походить від поєднання слів «deep learning» (глибинне навчання) і «fake» (підробка), що відображає основну концепцію: використання глибинного навчання для створення реалістичних фальсифікацій.

Технологія Deepfake представляє собою інструмент штучного інтелекту, що здатний змінювати відео, аудіо або зображення таким чином, щоб імітувати дії, мову або зовнішність іншої людини. Основою для DeepFake є використання глибинних нейронних мереж, які навчаються на великих обсягах даних для відтворення певних характерних рис об’єкта (як-от обличчя, манери мови або навіть голосу) [6]. В основі цієї технології лежить ряд специфічних моделей і методів, серед яких найбільш відомі автокодери, GAN (генеративно-змагальні мережі) та методи обробки аудіо- та відеосигналів.

Для створення DeepFake необхідно зібрати великий обсяг матеріалів, які містять зображення чи відео цільової особи. Чим більше зображень у різних умовах (з різних ракурсів, при різному освітленні тощо), тим краще модель зможе відтворити людину.

На цьому етапі використовується глибинне навчання, зокрема генеративно-змагальні мережі (GAN). GAN складається з двох нейронних мереж: генератора і дискримінатора. Генератор створює підроблені зображення, намагаючись переконати дискримінатор, що ці зображення реальні. Дискримінатор, у свою чергу, навчається розрізняти реальні і

підроблені зображення. У процесі навчання обидві мережі змагаються одна з одною, що дозволяє генератору створювати все реалістичніші фальшиві зображення.

Після тренування модель може бути застосована до цільового відео або аудіо, де здійснюється накладення обличчя чи голосу на оригінальний матеріал. Цей процес може також включати постобробку, щоб досягти плавності і відповідності з тоном та стилем оригінального контенту. Після генерації DeepFake матеріалу проводиться корекція, що дозволяє вдосконалити якість і правдоподібність підробки. Наприклад, може бути використане згладжування переходів або корекція кольору для кращої відповідності з оригінальним фоном чи освітленням.

1.1.1 Генеративні змагальні мережі

Генеративні змагальні мережі (Generative Adversarial Networks, або GANs) є однією з найбільш ефективних та поширених архітектур для створення DeepFake-контенту. GAN складається з двох нейронних мереж – генератора та дискримінатора, які працюють у змагальному режимі [7]. Генератор створює нові зображення, намагаючись зробити їх максимально реалістичними, тоді як дискримінатор визначає, чи є зображення справжнім чи створеним генератором. Ця динаміка змагання поступово вдосконалює обидві мережі, оскільки генератор навчається виробляти все більш якісні зображення, а дискримінатор – краще їх розпізнавати [8].

Генератор починає з випадкового шуму та перетворює його в зображення. На початкових етапах роботи ці зображення є досить грубими, але, як правило, поступово покращуються завдяки навчанню за схемою, яка показана на рисунку 1.1.

Він намагається створити обличчя або фотофрагменти, що відповідають обраній людині або заданому стилю.

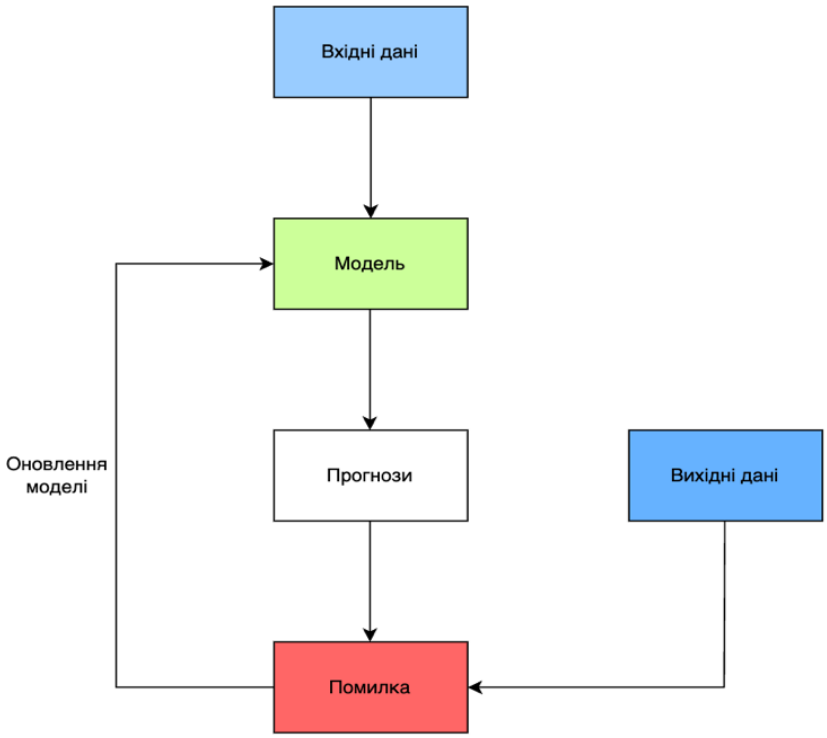

Рисунок 1.1 – Схема роботи генератора

Дискримінатор отримує як згенеровані зображення, так і справжні зображення з навчального набору та намагається визначити, які з них створені генератором. Його завдання – ідентифікувати підробки, як показано на рисунку 1.2.

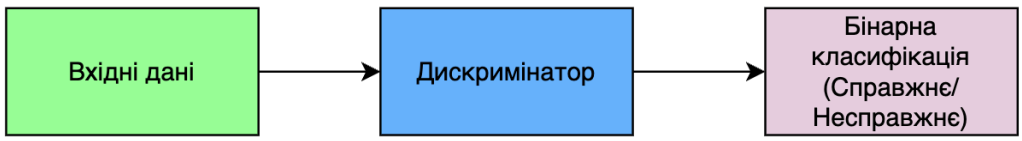

Рисунок 1.2 – Схема роботи дискримінатора

Генератор і дискримінатор навчаються в процесі змагання. Генератор намагається зробити контент, який дискримінатор класифікує як справжній, створюючи дедалі реалістичніші зображення, в той час як дискримінатор стає більш спостережливим до ознак підробки, що показано на рисунку 1.3.

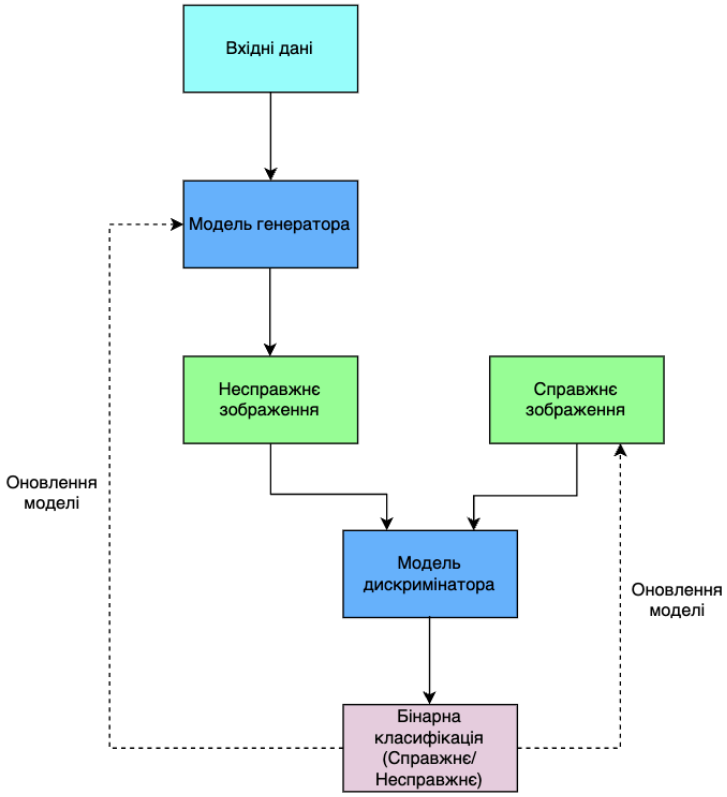

Рисунок 1.3 – Схема роботи генеративної змагальної мережі

У контексті DeepFake GAN часто використовуються для заміни обличчя однієї людини обличчям іншої на відео. Вибирається особа, обличчя якої буде перенесено (джерело) і цільове обличчя, яке буде замінено. Для якісного DeepFake потрібна велика кількість зображень або відео обох облич. GAN навчається створювати зображення, які відповідають зовнішності джерела і підходять до міміки та виразів цільового обличчя як показано на рисунку 1.4 та рисунку 1.5.

GAN дозволяють створювати високоякісний та реалістичний DeepFake-контент, який складно відрізнити від справжнього. Це стало можливим завдяки здатності GAN до самонавчання.

GAN потребують великої обчислювальної потужності та часу для тренування, особливо для складних DeepFake-відтворень, таких як відео.

Рисунок 1.4 – Справжні зображення

Рисунок 1.5 – Несправжні зображення (згенеровані DeepFake)

Щоб генератор міг створювати максимально реалістичні зображення, йому потрібні великі обсяги даних для тренування. Наприклад, для створення DeepFake обличчя певної людини бажано мати багато зображень цієї особи з різних кутів, при різному освітленні, з різними виразами обличчя. Це дозволяє генератору вивчити та синтезувати характерні особливості обличчя та поведінки особи.

Генератор бере випадковий шумовий вектор як вхідні дані. Цей шумовий вектор (випадковий набір чисел) служить стартовою точкою для створення синтетичного зображення. Кожен вектор кодує деяку варіацію, і в процесі навчання ці варіації починають відповідати різним рисам обличчя, положенням, виразам і іншим характеристикам.

Генератор, як правило, складається з кількох рівнів згорткових шарів з техніками підвищення роздільної здатності, наприклад, транспонованих згорткових шарів, які дозволяють перетворити початковий шумовий вектор у зображення. Зазвичай використовуються технології, такі як:

– Batch Normalization: щоб стабілізувати навчання;

– ReLU (Rectified Linear Units, випрямлені лінійні одиниці): для активації та підвищення нелінійності;

– Upsampling (підвищення роздільної здатності): для масштабування до бажаних розмірів.

Така архітектура дозволяє генератору поступово покращувати роздільну здатність створюваних зображень, зберігаючи при цьому дрібні деталі, які роблять фальсифікацію правдоподібною [9].

У процесі навчання генератор отримує зворотний зв’язок від дискримінатора – нейронної мережі, яка навчається розрізняти реальні та синтетичні зображення. Коли дискримінатор правильно виявляє підробку, генератор коригує свої параметри, щоб обманути дискримінатор у наступних спробах. Це досягається шляхом оптимізації функції втрат, яка в генератора має на меті зменшити здатність дискримінатора розпізнавати фальсифікацію. Таким чином, генератор навчається виробляти все реалістичніші зображення.

На пізніх етапах навчання генератор починає створювати надзвичайно деталізовані зображення. Однак у процесі генерації можуть виникати артефакти, які роблять зображення менш реалістичними (наприклад, деформації обличчя або некоректні кольори). Щоб цього уникнути, зазвичай використовуються додаткові методи, такі як постобробка, що включає в себе застосування фільтрів, корекція кольору, згладжування контурів. Також може використовуватися регуляризація, яка допомагає стабілізувати навчання генератора і уникати створення артефактів.

Після завершення навчання генератор може синтезувати зображення обличчя, яке візуально схоже на цільову особу. Якщо генератор працює в реальному часі, він може навіть забезпечити динамічну заміну обличчя у відеопотоці, що є основою для DeepFake-відео.

Генератору дуже важливо уникати таких проблем, як недостатня різкість або неузгодженість деталей (наприклад, неправильне відображення тіней або невідповідність кольору шкіри). Важливу роль тут відіграє якість архітектури

генератора та обсягу даних, які він отримує для тренування. Більше того, важливе налаштування гіперпараметрів, таких як швидкість навчання та розмір міні-пакетів (mini-batches), адже неправильні параметри можуть призвести до колапсу моди – явища, коли генератор навчається відтворювати тільки кілька шаблонних варіантів замість повного спектра можливих облич.

Таким чином, генератор у GAN для DeepFake є складним механізмом, що забезпечує створення реалістичних підробок завдяки постійній змагальній взаємодії з дискримінатором. Цей процес дає змогу досягти високого рівня реалістичності зображень, але водночас вимагає дуже точного налаштування і великого обсягу якісних даних [10].

1.1.2 Згорткові нейронні мережі

Згорткові нейронні мережі (CNN, англ. Convolutional Neural Networks) відіграють важливу роль у створенні DeepFake завдяки здатності ефективно обробляти зображення та розпізнавати деталі, які критичні для реалістичного відтворення обличчя. CNN здатні виділяти різноманітні особливості, такі як форми, текстури, і навіть складніші патерни, що дозволяє їх застосовувати в завданнях заміни або редагування облич у відео і зображеннях.

Для тренування CNN для DeepFake необхідно мати великий набір зображень цільової особи з різних ракурсів, при різному освітленні та з різними виразами обличчя. CNN буде використовувати ці дані для вивчення характерних особливостей обличчя, що в подальшому дозволить генерувати або змінювати зображення так, щоб вони виглядали правдоподібно.

Конволюційні шари CNN складаються з набору фільтрів (або ядер), які проходять через зображення, виділяючи ознаки (features) на різних рівнях абстракції як показано на рисунку 1.6. На початкових рівнях CNN фільтри виділяють простіші ознаки, такі як контури та текстури, тоді як на пізніших

рівнях мережа починає розпізнавати складніші патерни, наприклад, риси обличчя або контури очей і рота.

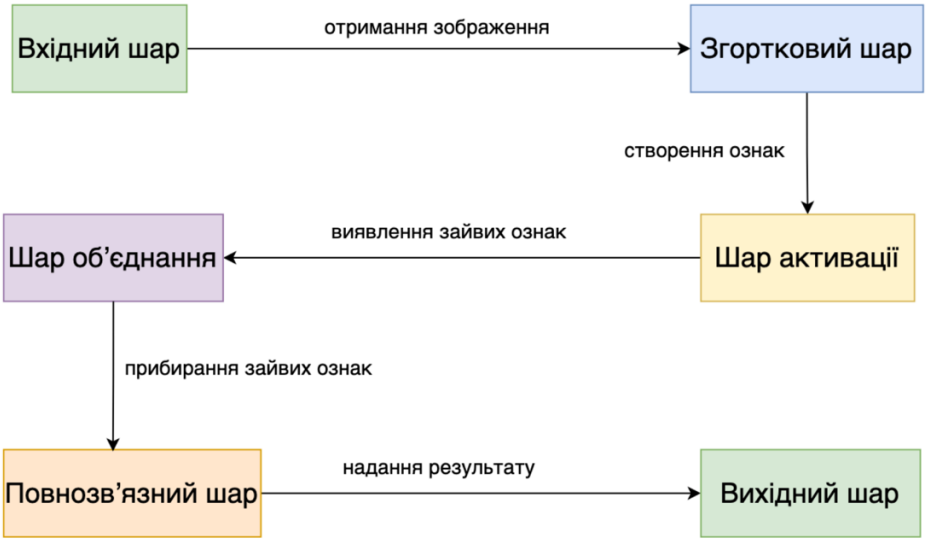

Рисунок 1.6 – Схема шарів згорткової нейронної мережі

Нижчі рівні CNN виділяють основні деталі обличчя, такі як контури носа, рота, очей. Середні рівні розпізнають більш специфічні деталі, наприклад, форму очей або складки на шкірі. Вищі рівні фокусуються на загальній структурі обличчя, а також особливостях, що роблять обличчя унікальним, як-от специфічні риси очей, губ, тощо.

CNN можуть працювати спільно з автокодерами або з генеративно-змагальними мережами (GAN) для створення DeepFake. Наприклад, у автокодерах конволюційна нейронна мережа використовується для перетворення обличчя в компактне представлення – так званий вектор ознак (feature vector), який кодує всю необхідну інформацію про риси обличчя. Цей вектор далі декодується іншою CNN для відтворення фальсифікованого зображення з цією ж інформацією.

Також одним із завдань CNN є забезпечення того, щоб згенероване обличчя виглядало природно і гармонійно вписувалося в оточення. Для цього використовуються спеціальні шари, що відстежують стиль цільового

зображення (наприклад, освітлення, колір шкіри), і адаптують згенероване обличчя відповідно до цих характеристик.

На вихідних шарах CNN застосовуються методи постобробки, які можуть усувати помилки чи артефакти, що виникли в процесі обробки. Наприклад, обличчя може бути некоректно вирівняне або злегка розмите. Постобробка за допомогою CNN коригує такі недоліки, наприклад, згладжуючи контури чи підлаштовуючи освітлення, щоб DeepFake виглядав максимально природно.

CNN постійно навчається на основі зворотного зв'язку, використовуючи функцію втрат для корекції своїх параметрів. У разі змагання з дискримінатором (якщо CNN входить до складу GAN), генератор, що використовує CNN, поступово стає кращим у створенні фальсифікацій, а дискримінатор вдосконалюється у їх розпізнаванні. Це змагання поступово покращує якість результату, роблячи згенероване обличчя все більш реалістичним як показано на рисунку 1.7.

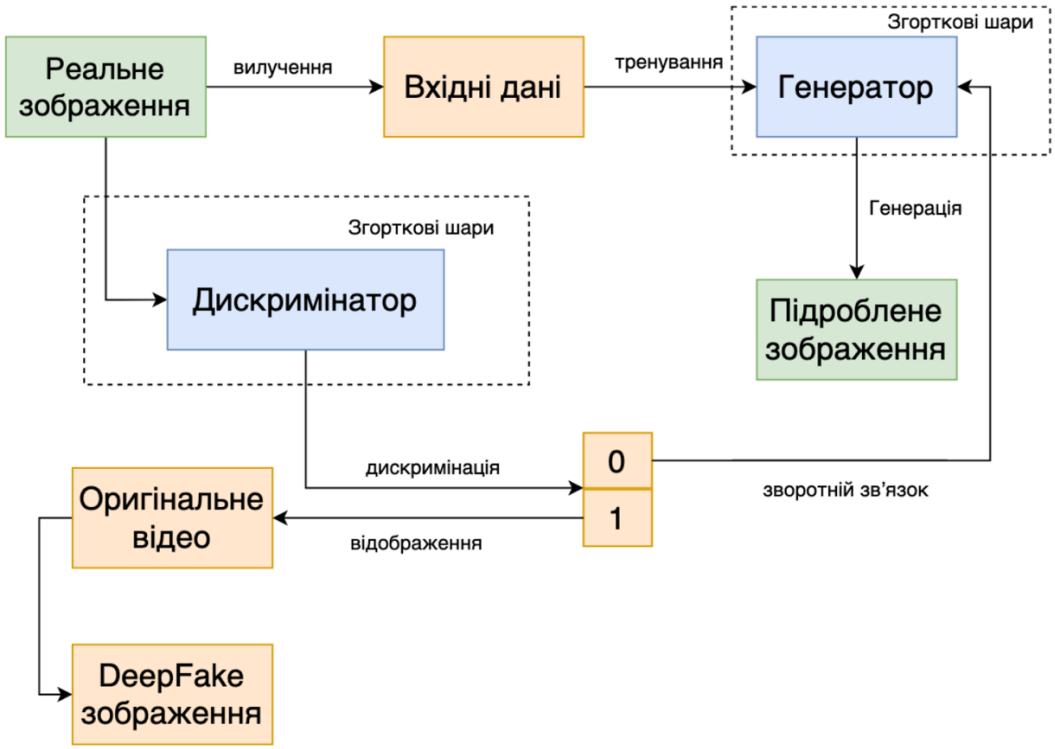

Рисунок 1.7 – Схема роботи CNN у генеративній змагальній нейронній мережі

Для створення реалістичних DeepFake зазвичай використовуються удосконалені архітектури CNN, такі як U-Net, ResNet та VGG і Inception.

U-Net – це архітектура, яку використовують для задач сегментації, але вона також може допомагати накладати обличчя або виділяти специфічні ділянки зображення. Завдяки симетричній структурі і можливості об'єднувати інформацію з початкових шарів з кінцевими, U-Net забезпечує високу точність у передачі деталей. ResNet використовує залишкові блоки (residual blocks), які дозволяють уникати проблеми затухаючих градієнтів і сприяють збереженню важливих деталей. ResNet може зберігати важливі характеристики обличчя, такі як унікальні риси, що є ключовими для реалістичності. VGG і Inception також добре підходять для задач обробки зображень та забезпечення деталізації в згенерованих обличчях.

Таким чином, CNN відіграють ключову роль у створенні глибинних фальсифікацій завдяки своїй здатності аналізувати та відтворювати складні патерни зображень. Завдяки багаторівневій архітектурі, вони забезпечують реалістичність та плавність у згенерованих зображеннях і відео. Однак, незважаючи на всі переваги, CNN потребують ретельного налаштування та великих обчислювальних ресурсів для створення високоякісних DeepFake.

1.2 Методи оцінення якості DeepFake

Оцінка якості роботи CNN при створенні DeepFake включає кілька технічних підходів, які допомагають визначити, наскільки реалістично згенероване зображення чи відео виглядає та наскільки точно воно імітує оригінал. Вибір метрик та методів оцінки залежить від завдання, але в цілому вони охоплюють як об'єктивні показники (метрики подібності зображень, структури), так і суб'єктивні (оцінка реалістичності людським оком).

Існує багато різних метрик, що дозволяють оцінювати якість зображень, зокрема тих, які використовуються для аналізу DeepFake. Однією з найбільш

поширених є PSNR (Peak Signal-to-Noise Ratio), яка вимірює співвідношення між максимальною можливою потужністю зображення та потужністю шуму, що впливає на його відтворення. Чим вищим є значення PSNR, тим менша різниця між оригінальним та згенерованим зображенням, що вказує на його високу якість. Дещо глибший підхід пропонує SSIM (Structural Similarity Index Measure), яка оцінює схожість між двома зображеннями, враховуючи їх структуру, яскравість і контраст. Ця метрика є особливо корисною, оскільки дає більше інформації про структурну якість зображення, ніж проста піксельна різниця.

Ще одним важливим методом є LPIPS (Learned Perceptual Image Patch Similarity), що використовує попередньо навчені нейронні мережі для аналізу подібності зображень на рівні високорівневих ознак, таких як текстури й контури. Такий підхід дозволяє оцінити зображення з точки зору особливостей, які людина сприймає як значущі. Крім того, для порівняння якості зображень часто використовується FID (Frechet Inception Distance), яка базується на характеристиках, отриманих із попередньо навчених глибоких нейронних мереж (часто архітектура Inception). Ця метрика вимірює відстань між розподілами ознак реальних і згенерованих зображень: чим менше значення FID, тим реалістичніше виглядає згенерований контент. Водночас Inception Score (IS) дозволяє оцінити ймовірність належності згенерованих зображень до певних категорій, а також їх різноманітність. Ця метрика корисна для аналізу не лише якості, але й розмаїття зображень.

Особливої уваги заслуговує оцінка якості обличчя, оскільки для DeepFake найважливішими є імітація обличчя, його рухів та емоцій. Однією з ключових метрик у цьому контексті є Facial Landmark Error (FLE), що вимірює відстань між ключовими точками обличчя, такими як очі, ніс чи рот, на згенерованому й оригінальному зображеннях. Чим нижчим є це значення, тим краще зберігаються риси обличчя. Ще одним методом є Pose Matching, який оцінює відповідність положення голови та виразу обличчя між оригінальним і згенерованим зображеннями. Цей підхід може включати аналіз кутів нахилу

голови або використання спеціальних моделей для розпізнавання емоцій. Важливим показником є також Face Identity Preservation Score, який за допомогою попередньо навчених моделей для розпізнавання осіб, таких як FaceNet чи ArcFace, дозволяє визначити, наскільки згенероване обличчя відповідає оригіналу з точки зору ідентичності.

Для DeepFake-відео важливо забезпечити плавність і узгодженість між кадрами. У цьому контексті метрика Temporal Consistency вимірює різницю між послідовними кадрами, аналізуючи, наскільки плавно змінюється обличчя з кадру в кадр. Для цього можна порівнювати ключові точки обличчя або аналізувати текстури й кольори. Крім того, використовується Optical Flow Consistency, яка дозволяє виміряти потік руху між кадрами й оцінити, чи відповідає цей потік реалістичним рухам обличчя.

Однак технічні метрики не завжди здатні повністю оцінити реалістичність зображення чи відео з точки зору людини. Саме тому суб’єктивні методи, як-от оцінка реалістичності експертами, є надзвичайно важливими. У цьому випадку групі людей демонструють згенеровані відео чи зображення, і вони оцінюють їхню правдоподібність, враховуючи такі фактори, як жвавість погляду або плавність рухів. Іншим підходом є сліпі тести, коли глядачам показують як реальні, так і фейкові зображення, не повідомляючи, де саме фальсифікація. Частота помилок глядачів допомагає визначити ступінь реалістичності DeepFake.

Додатково для оцінки якості DeepFake використовуються спеціальні детектори фальсифікацій. Якщо згенероване зображення чи відео важко виявити такими моделями, це є показником високої якості фальсифікації. Для цього використовуються моделі, натреновані на виявлення артефактів, як-от некоректні тіні, розмиті контури або неузгодженість кольорів. Метрики Color Matching та Shadow Consistency також відіграють важливу роль. Color Matching оцінює, наскільки точно згенероване обличчя відповідає кольоровій гамі оригіналу чи фону, а Shadow Consistency аналізує коректність розташування тіней, що особливо важливо для відео.

Таким чином, ефективна оцінка DeepFake вимагає поєднання об’єктивних метрик, суб’єктивної експертизи та аналізу на спеціалізованих детекторах для досягнення максимальної реалістичності зображень та відео.

Ці технічні методи дозволяють комплексно оцінити якість роботи CNN при створенні DeepFake. У поєднанні об’єктивних, суб’єктивних і детекційних підходів можна отримати точну картину реалістичності та відповідності згенерованого контенту оригіналу.

1.3 Методи покращення роботи CNN

Методи покращення роботи згорткових нейронних мереж (CNN) при створенні DeepFake спрямовані на вдосконалення архітектури, оптимізацію процесу навчання та застосування сучасних технологій для підвищення якості генерованого контенту. Одним із ключових аспектів є оптимізація роботи CNN, що дозволяє швидше і ефективніше мінімізувати функцію втрат. Для цього застосовуються різноманітні методи оптимізації, серед яких Adam (Adaptive Moment Estimation), що поєднує адаптивні темпи навчання з моментумом і забезпечує швидку та стабільну збіжність до оптимуму. Також популярним є метод RMSprop (Root Mean Square Propagation), який регулює темпи навчання для кожного параметра окремо, що особливо корисно при роботі з великими обсягами даних. Крім цього, часто використовується Stochastic Gradient Descent with Momentum (SGD+Momentum), що прискорює процес навчання та допомагає зменшити коливання параметрів під час градієнтного спуску.

Щоб запобігти перенавчанню моделі, застосовуються різні техніки регуляризації. Однією з найбільш ефективних є Dropout, яка передбачає випадкове вимкнення частини нейронів під час навчання. Це дозволяє покращити узагальнюючу здатність моделі, знижуючи залежність від конкретних нейронів. Іншим важливим методом є L2-регуляризація (або

регуляризація зменшення ваг), яка додає штраф за великі значення параметрів і сприяє стабільності роботи моделі.

Для підвищення ефективності навчання моделі також важливо збільшити різноманітність тренувального набору даних. Це досягається за допомогою аугментації даних, що включає такі техніки, як обертання зображень, їх масштабування, зміщення кольорів, а також дзеркальне відображення. Ці методи дозволяють моделі навчитися працювати з різними варіаціями вхідних даних, що робить її більш стійкою до змін у реальному середовищі.

Особливу роль у створенні DeepFake відіграє вибір функції втрат, яка враховує специфіку завдання. Для генерування реалістичних облич та текстур використовується Perceptual Loss, що порівнює зображення не на рівні пікселів, а у просторі ознак високого рівня, отриманих за допомогою попередньо навчених нейронних мереж. Це дозволяє значно підвищити візуальну якість згенерованого контенту. Крім того, для задач, де використовуються генеративні змагальні мережі (GAN), застосовується Adversarial Loss. Ця функція втрат забезпечує змагальне навчання між генератором і дискримінатором, що сприяє створенню реалістичних та високоякісних зображень.

Таким чином, покращення роботи CNN у задачах DeepFake вимагає комплексного підходу, що поєднує оптимізацію архітектури, ефективні стратегії навчання та використання спеціалізованих функцій втрат для забезпечення реалістичності й високої якості згенерованого контенту [11].

Використання попередньо натренованих моделей дозволяє значно скоротити час тренування та покращити результати. Наприклад, моделі, натреновані на великих наборах даних, таких як ImageNet, використовуються як основа для спеціалізованих задач [12].

У задачах створення DeepFake моделі часто навчаються ітеративно, поступово покращуючи якість результатів. Це дозволяє краще адаптуватися до особливостей даних.

Всі ці методи в сукупності дозволяють CNN забезпечувати високу якість згенерованого контенту, підвищувати ефективність тренування та створювати більш реалістичні й природні DeepFake.

1.4 Постановка задачі дослідження

Таким чином, створення реалістичних DeepFake є актуальним завданням для розвитку технологій глибинного навчання і комп'ютерного зору. Тому ставиться завдання аналізу та вдосконалення методів генерації DeepFake зображень на основі згорткових нейронних мереж (CNN), що забезпечують високу якість і реалістичність без істотного збільшення обчислювальних ресурсів.

Об'єктом дослідження є робота згорткових нейронних мереж при створенні DeepFake.

Метою дослідження є аналіз ефективності застосування згорткових нейронних мереж для генерації реалістичних зображень.

Для досягнення мети необхідно вирішити такі завдання:

– провести аналіз існуючих методів і архітектур CNN, які використовуються для створення DeepFake;

– дослідити ефективність різних архітектур CNN для генерації реалістичних зображень і відео;

– розробити алгоритм для оцінки якості та реалістичності згенерованого контенту;

– провести експериментальне тестування та оптимізацію моделей CNN для підвищення продуктивності.

2 МАТЕМАТИЧНІ МОДЕЛІ СТВОРЕННЯ DEEPFAKE

2.1 Архітектура CNN

Згорткові нейронні мережі (Convolutional Neural Networks, CNN) широко використовуються для обробки зображень завдяки їхній здатності автоматично витягувати важливі ознаки та перетворювати їх для подальшого навчання моделі [13]. Основними елементами CNN є згорткові шари, шари пулінгу та повнозв'язані шари. Кожен з них виконує специфічну математичну операцію, яка сприяє обробці та інтерпретації даних.

Згортковий шар є основним компонентом CNN [14]. Його мета – обчислити згортку вхідного зображення з фільтром (ядром), щоб виділити певні ознаки [15]. У згортковому шарі кожен піксель результату є результатом операції між певною областю зображення та фільтром.

Операція згортки застосовується між фільтром (ядром) та вхідним зображенням, що може бути представлене математичним виразом. Нехай $\mathrm{I}(\mathrm{x}, y)$ – це інтенсивність пікселів у зображенні, а $\mathrm{K}(i, j)$ – це елементи фільтра (ядра), де $i,\ j$ – індекси фільтра. Результат згортки обчислюється за виразом:

$$S(x, y) = \sum_{i=0}^{k-1} \sum_{j=0}^{k-1} I(\mathrm{x} + i, y + j) \cdot \mathrm{K}(i, j), \tag{2.1}$$

де $S(x, y)$ – значення пікселя на вихідній карті ознак після згортки;

$I(\mathrm{x} + i, y + j)$ – значення пікселів у вхідному зображенні;

$\mathrm{K}(i, j)$ – значення фільтра розміром $k \times k$ (зазвичай 3×3 або 5×5).

Рух ядра по зображенню визначається кроком ядра, який вказує, наскільки далеко фільтр пересувається під час обробки зображення. Якщо крок дорівнює одиниці, ядро переміщується на один піксель вправо або вниз, а якщо дорівнює двом, то на два пікселі. Щоб зберегти розміри зображення під час згортки, часто використовують доповнення країв, наприклад, додавання

нульових значень навколо країв зображення дозволяє уникнути зменшення розмірів вихідної карти ознак після згортки [16].

Для 3×3 фільтра K та вхідного зображення розміром 5×5, результатом згортки буде нове зображення (карта ознак) розміром 3×3, якщо крок ядра (stride) дорівнює 1, і padding не використовується.

Шар пулінгу відповідає за зменшення просторового розміру карти ознак, зберігаючи при цьому найважливіші ознаки. Це зменшує кількість параметрів і знижує ризик перенавчання, а також робить модель більш стійкою до змін у вході.

Існує кілька типів пулінгу, найпоширенішими є макс-пулінг (max pooling) та середній пулінг (average pooling) [17].

Макс-пулінг (Max pooling) працює таким чином, що кожній області вибирається максимальне значення. Вираз для макс-пулінгу:

$$S(x, y) = \max\{I(x + i, y + j)\}, \tag{2.2}$$

де $i,\ j \in [0, p - 1]$, а p – розмір вікна пулінгу (наприклад, 2×2).

У середньому пулінгу (Average pooling) обчислюється середнє значення пікселів у кожній області розміром $p \times p$. Вираз для середнього пулінгу:

$$S(x, y) = \frac{1}{p^2}\sum_{i=0}^{p-1} \sum_{j=0}^{p-1} I(\mathrm{x} + i, y + j) \cdot \mathrm{K}(i, j). \tag{2.3}$$

Пулінг виконує важливу роль у згорткових нейронних мережах, оскільки він дозволяє зменшити розмір карти ознак і підвищити стійкість моделі до різноманітних змін у даних. Основною метою пулінгу є зменшення розміру зображення, що досягається за рахунок об’єднання значень у невеликих областях. Наприклад, якщо для вхідної карти ознак розміром 4×4 застосувати макс-пулінг із вікном 2×2 та кроком 2, то вихідна карта ознак

матиме розмір 2×2. Завдяки цьому зменшується обчислювальна складність, а також обсяг даних, які необхідно обробляти.

Крім того, пулінг сприяє стійкості моделі до зсувів і обертань. Це відбувається тому, що під час операції пулінгу модель або вибирає найбільш значущі елементи в області (як у випадку макс-пулінгу), або усереднює значення (як у середньому пулінгу). У результаті модель стає менш чутливою до незначних змін у вихідних даних, таких як дрібні зсуви чи обертання об’єкта на зображенні.

Після того, як згорткові шари і шари пулінгу витягли ознаки з зображення, їх потрібно передати на класифікатор, який приймає рішення на основі цих ознак [18]. Повнозв’язані шари (fully connected layers) виконують функцію цього класифікатора. Вони поєднують усі вхідні ознаки і обчислюють підсумковий результат.

У повнозв’язаних шарах кожен нейрон з’єднаний з усіма нейронами попереднього шару. Вихід кожного нейрона обчислюється як лінійна комбінація вхідних ознак з ваговими коефіцієнтами та зсувами, після чого застосовується функція активації [19].

Вираз для виходу нейрона в повнозв’язаному шарі виглядає так:

$$\mathrm{z} = \sum_{i=1}^{n} \left(\frac{n}{k}\right) w_i x_i + b, \tag{2.4}$$

де x_i – вхідні значення (результати попереднього шару);

w_i – ваги зв’язків нейронів (параметри, що оптимізуються під час навчання);

b – зсув (bias);

z – вихід нейрона.

Після лінійної комбінації застосовується функція активації.

Існують такі функції активації, як ReLU (Rectified Linear Unit) $f(z) = \max(0, z)$; сигмоїдна функція (Sigmoid) $f(z) = \frac{1}{1+e^{-z}}$; функція softmax, яка

використовується на вихідному шарі для класифікації, де сума всіх ймовірностей дорівнює 1. Вона перетворює вихідні значення в ймовірності для кожного класу.

$$f(z) = \frac{1e^{z_j}}{\sum_{k=1}^{n} e^{z_k}}, \tag{2.5}$$

де z_j – вихідний сигнал для кожного класу j;

n – кількість класів.

Функція повнозв'язаних шарів полягає в тому, що вони приймають витягнуті карти ознак і перетворюють їх у вектор, на основі якого виконується класифікація. Ці шари дозволяють моделі, використовуючи витягнуті ознаки, приймати рішення, наприклад, розпізнавати клас зображення або визначати ймовірність наявності певного об'єкта на зображенні.

Згорткові шари відповідають за витягування локальних ознак зображення, шари пулінгу зменшують просторовий розмір даних, зменшують кількість параметрів та роблять модель більш стійкою до зсувів, а повнозв'язані шари обробляють отримані ознаки, щоб приймати остаточні рішення, такі як класифікація або прогнозування [20].

Згорткові шари використовуються для обробки та виявлення локальних ознак вхідного зображення. Кожен згортковий шар фокусується на виявленні конкретних структур, таких як краї, текстури або патерни. Згортка виконує роль обчислювача локальних ознак, застосовуючи фільтри до вхідного зображення.

Шари пулінгу виконують подальшу обробку, зменшуючи просторову розмірність карти ознак після кожного згорткового шару. Це дозволяє моделі узагальнювати інформацію та зменшувати кількість параметрів [21]. Пулінг також робить модель більш стійкою до дрібних змін, таких як обертання або масштабування зображення, зберігаючи найважливіші ознаки.

Повнозв’язані шари знаходяться ближче до виходу мережі і використовуються для фінальної обробки витягнутих ознак [22]. Ці шари збирають всю інформацію, отриману зі згорткових та пулінгових шарів, і перетворюють її на рішення. Наприклад, у класифікаційних задачах повнозв’язані шари дають результат – імовірності для кожного класу.

Переваги використання CNN для аналізу зображень полягають у малому обсягу параметрів порівняно зі звичайними нейронними мережами. Згорткові шари дозволяють використовувати меншу кількість параметрів, оскільки ваги фільтрів спільно використовуються по всьому зображенню [23].

Згорткові шари здатні автоматично виявляти локальні структури, що дає можливість отримати важливі ознаки для подальшої обробки.

Завдяки пулінгу та спільному використанню фільтрів CNN здатні добре працювати з зображеннями, які мають деякі зміни у структурі або представленні об’єктів.

З кожним новим згортковим шаром виділяються дедалі складніші ознаки, переходячи від простих країв і текстур до більш абстрактних концепцій, таких як форми або частини обличчя.

Згорткові шари виділяють локальні ознаки з вхідних зображень через застосування фільтрів, пулінг зменшує розмірність даних та допомагає моделі фокусуватися на найбільш важливих ознаках, а повнозв’язані шари завершують обчислення, перетворюючи витягнуті ознаки на фінальні рішення або прогнози. Завдяки своїм математичним основам CNN забезпечують ефективне витягування ознак і добре підходять для завдань, пов’язаних із зображеннями, такими як розпізнавання облич, обробка зображень та створення DeepFake [24].

Важливим компонентом навчання згорткової нейронної мережі є зворотне поширення (backpropagation). Це метод оптимізації, який дозволяє мережі оновлювати свої ваги, щоб мінімізувати помилку між передбаченими та фактичними результатами. Функції втрат, такі як крос-ентропія або середньоквадратична похибка, грають центральну роль у цьому процесі,

оскільки вони вимірюють величину помилки і вказують, як мережі потрібно коригувати свої параметри.

Мета CNN під час навчання полягає в тому, щоб мінімізувати помилку, яка виникає між передбаченими виходами моделі та очікуваними значеннями (мітками). Для цього використовується алгоритм зворотного поширення, який обчислює похідні функції втрат по кожному параметру моделі і оновлює їх, щоб зменшити помилку [25].

Процес навчання через зворотне поширення можна розділити на кілька кроків:

Крок 1. Пряме проходження (Forward Pass): спочатку мережа бере вхідні дані (наприклад, зображення), пропускає їх через всі шари (згорткові, пулінгові, повнозв'язані), та генерує вихідний результат (наприклад, ймовірності для класів у класифікаційній задачі).

Крок 2. Обчислення помилки (Error Calculation): вихід моделі порівнюється з очікуваним результатом (ground truth); різниця між ними називається помилкою, яка обчислюється за допомогою функції втрат. Наприклад, якщо це задача класифікації, використовується функція крос-ентропії. Якщо це регресія, можна використовувати середньоквадратичну похибку.

Крок 3. Зворотне поширення (Backpropagation): після обчислення помилки алгоритм зворотного поширення обчислює градієнт цієї помилки по відношенню до кожного параметра мережі; це робиться за допомогою ланцюгового правила диференціювання; похідні показують, як сильно зміна кожної ваги вплине на загальну помилку.

Крок 4. Оновлення ваг (Gradient Descent): ваги нейронної мережі оновлюються на основі обчислених градієнтів, зменшуючи величину помилки; це робиться за допомогою методу градієнтного спуску, який дозволяє поступово коригувати ваги в правильному напрямку; вираз для оновлення ваги w виглядає так:

$$w_{\text{нове}} = w_{\text{старе}} - \eta \frac{\delta L}{\delta w}, \tag{2.6}$$

де η – це коефіцієнт навчання (learning rate), який контролює швидкість оновлення ваг;

$\frac{\delta L}{\delta w}$ – градієнт функції втрат L по відношенню до ваги w.

Крок 5. Повторення процесу: цей процес повторюється для кожної ітерації (епохи) навчання, поки мережа не досягне прийнятного рівня точності, тобто мінімальної помилки.

2.1.1 Згортковий шар

Згортка є основною операцією в CNN, яка дозволяє ефективно виявляти локальні ознаки на зображеннях, наприклад, краї, текстури або форми. Згортковий шар використовує фільтри (або ядра), які переміщуються по вхідному зображенню (або попередньому шару мережі) та обчислюють скалярний добуток між фільтром і частиною вхідного зображення.

Математично згортка визначається наступним чином:

$$Y_{i,j} = (X \cdot W)_{i,j} = \sum_{m=0}^{k-1} \sum_{n=0}^{k-1} X_{i+m,j+n} \cdot W_{m,n} + b, \tag{2.7}$$

де $Y_{i,j}$ – результат згортки на позиції i, j;

X – вхідне зображення (або активації попереднього шару);

W – фільтр;

b – зсув (bias);

k – розмір фільтра.

Цей процес здійснюється для кожної позиції зображення, де фільтр ковзає, і на кожному етапі виконується множення та додавання.

2.1.2 Функція активації

Після згортки результат подається через функцію активації, щоб внести нелінійність у модель, що дозволяє CNN навчатися складним залежностям. Найбільш популярною є функція ReLU (Rectified Linear Unit):

$$f(x) = max(0, x). \tag{2.8}$$

Інші популярні функції активації включають sigmoid, але ReLU є найпоширенішою завдяки простоті та ефективності.

2.1.3 Шар пулінгу

Пулінг використовується для зменшення розмірності простору ознак (feature map) та виділення найбільш важливих ознак, що забезпечує певну інваріантність до зсувів, масштабування і деформацій.

Найпоширенішим є максимальний пулінг:

$$Y_{i,j} = max(X_{i,j}, X_{i+1,j}, X_{i,j+1}, X_{i+1,j+1}), \tag{2.9}$$

де кожен елемент у вхідній матриці X бере участь в обчисленні максимуму на малих підвибірках.

Іншим типом пулінгу є середній пулінг , де замість максимуму обчислюється середнє значення елементів:

$$Y_{i,j} = \frac{1}{n} \sum_{m,n \in Region} X_{+m,j+n}, \tag{2.10}$$

де n – кількість елементів у вибірці.

2.1.4 Шар нормалізації

Нормалізація допомагає покращити швидкість і стабільність навчання, забезпечуючи, щоб активації знаходились в певному діапазоні. Batch Normalization – один із найпоширеніших методів.

Математично це виглядає так:

$$\widehat{X}_i = \frac{X_i - \mu}{\sqrt{\sigma^2 - \epsilon}}, \tag{2.11}$$

де X_i– значення активації;

μ – середнє значення;

σ^2 – дисперсія;

ϵ – маленьке значення для стабільності обчислень.

2.1.5 Шар повнозв’язний

Після згорткових і пулінг-шарів вихідні ознаки подаються в повнозв’язний шар, де кожен нейрон пов’язаний з усіма нейронами попереднього шару. Це забезпечує остаточну трансформацію ознак у вихідну модель.

Математичний вигляд:

$$Y = W \cdot X + b, \tag{2.12}$$

де X – вхідні ознаки;

W – матриця ваг;

b – вектор зсуву;

Y – вектор результатів.

2.1.6 Функція втрат

Функція втрат використовується для оцінки ефективності моделі, а також для навчання нейронної мережі через процес зворотного поширення помилок [26]. Для задачі класифікації зазвичай використовують крос-ентропію:

$$L = -\sum_{i=1}^{C} y_i \log(\hat{y}_i)), \tag{2.13}$$

де C – кількість класів;

y_i – справжнє значення (мішень);

$\hat{y}_i$ – ймовірність, передбачена моделлю.

Для задач регресії використовується середньоквадратична похибка (MSE):

$$L = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2, \tag{2.14}$$

де y_i – фактичне значення;

$\hat{y}_i$ – передбачене значення.

Функція втрат (loss function) – це математична функція, яка використовується для оцінки, наскільки добре модель передбачає результати для заданих вхідних даних. Вона вимірює різницю між фактичними значеннями (ground truth) і передбаченнями моделі. Мета навчання нейронної мережі – звести функцію втрат до мінімуму, щоб модель працювала якомога точніше.

Роль функції втрат у навчанні нейронних мереж полягає у оцінки якості моделі, вона показує, наскільки сильно модель помиляється при передбаченні вихідних значень. Маленьке значення функції втрат свідчить про те, що модель добре узгоджується з даними. Функція втрат використовується для обчислення градієнтів через алгоритм зворотного поширення. Це дозволяє

оновлювати ваги нейронної мережі в напрямку зменшення втрат [27]. Процес зворотного поширення помилок проходить таким чином:

– вхідні дані проходять через нейронну мережу;

– мережа генерує вихід (передбачення);

– функція втрат обчислює помилку між передбаченням і фактичними значеннями;

– використовуючи метод градієнтного спуску, обчислюються похідні (градієнти) функції втрат відносно кожного параметра (ваг і зміщень);

– градієнти вказують, наскільки і в якому напрямку необхідно змінити параметри для зменшення втрат;

– алгоритм оновлює ваги нейронної мережі за виразом

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta L(\theta), \tag{2.15}$$

де θ_t – параметри на t-му кроці;

η – швидкість навчання (learning rate);

$\nabla_\theta L(\theta)$ – градієнт функції втрат;

– повторення ітерацій; цей процес повторюється, доки значення функції втрат не зійдеться до мінімуму, або не буде досягнуто заданої кількості епох. Вибір функції втрат залежить від типу задачі. Для регресії :

Середня квадратична помилка (MSE)

$$L(y, \hat{y}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2, \tag{2.16}$$

штрафує за великі відхилення між передбаченням $\hat{y}$ і фактичним значенням y. Також можна використати середню абсолютну похибку (MAE):

$$L(y, \hat{y}) = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|. \tag{2.17}$$

Підходить для завдань, де важливі малі відхилення.

Для класифікації Binary Cross-Entropy (логарифмічна втрата) для задач бінарної класифікації:

$$L(y, \hat{y}) = -\frac{1}{n}\sum_{i=1}^{n}(y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)). \quad (2.18)$$

Використовується для двокласових задач, де вихід – ймовірність.

Categorical Cross-Entropy для багатокласової класифікації:

$$L(y, \hat{y}) = -\frac{1}{n}\sum_{i=1}^{n}\ \sum_{j=1}^{k}(y_{ij} \log(\hat{y}_{ij}). \quad (2.19)$$

Використовується, коли моделі потрібно обрати один із кількох класів.

Функція втрат – це ключовий елемент навчання нейронної мережі. Вона визначає, наскільки ефективно модель працює, та дозволяє поступово покращувати її за допомогою алгоритмів оптимізації [28]. Правильний вибір функції втрат залежить від типу задачі та специфіки даних. Вибір функції втрат залежить від характеру задачі. Важливо також враховувати специфіку даних та мету моделі, для деяких складних задач, можуть бути застосовані більш складні функції втрат, що враховують перцептивні якості результату або особливості навчальних даних.

2.1.7 Алгоритми оптимізації

Алгоритми оптимізації застосовуються для оновлення ваг у нейронних мережах на основі градієнтів, отриманих за допомогою зворотного поширення.

Adam є одним із найпоширеніших алгоритмів оптимізації, який використовується для тренування нейронних мереж. Він поєднує переваги

двох інших методів: AdaGrad (який добре працює для рідкісних особливостей) і RMSProp (який ефективний для обробки нестабільних градієнтів). Adam використовує адаптивні швидкості навчання для кожного параметра та враховує моментум, що робить його стійким і швидким у збіжності.

Adam оцінює перший (середнє значення) і другий момент (недискримінована дисперсія) градієнтів під час оптимізації. Ці оцінки дозволяють алгоритму динамічно коригувати кроки оновлення параметрів.

Нехай m_t та v_t – перші та другі моменти градієнтів,

β_1 та β_2 – гіперпараметри для моментів,

η – швидкість навчання,

g_t – градієнт функції втрат $f(\theta)$ по θ_t на t-му кроці,

ϵ – мале значення для стабільності.

Алгоритм працює наступним чином:

Крок 1. Встановлюються початкові значення параметрів θ_0.

Крок 2. Ініціалізуються нульові перші моменти $m_0 = 0$ та другі моменти $v_0 = 0$.

Крок 3. На кожній ітерації t обчислюється градієнт:

$$g_t = \nabla f(\theta_{t-1}). \tag{2.20}$$

Крок 4. Оновлюються експоненціально зважені середні:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t, \tag{2.21}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) \nabla {g_t}^2. \tag{2.22}$$

Крок 5. Виконується корекція упередженості (оскільки m_t і v_t на початкових етапах зміщені до нуля):

$$\widehat{m_t} = \frac{m_t}{1-\beta_1^t}, \tag{2.23}$$

$$\widehat{v_t} = \frac{v_t}{1-\beta_2^t}. \quad (2.24)$$

Крок 6. Оновлюються параметри:

$$\theta_t = \theta_{t-1} - \eta \frac{\widehat{m_t}}{\sqrt{\widehat{v_t}}+\epsilon}. \quad (2.25)$$

Adam автоматично регулює швидкість навчання для кожного параметра. Завдяки другому моменту алгоритм стабільно працює навіть на складних задачах з високою нестабільністю градієнтів. Adam менш залежний від точного підбору гіперпараметрів у порівнянні з іншими алгоритмами, як SGD. Широко використовується в задачах комп'ютерного зору, обробки тексту та інших областях, що потребують великих моделей. У деяких задачах Adam може демонструвати нестабільну збіжність до оптимального рішення. Через адаптивні швидкості навчання можливий ризик застрягти в точках saddle points.

Adam залишається ефективним та універсальним методом оптимізації, особливо для глибоких нейронних мереж. У багатьох сценаріях він забезпечує швидшу збіжність, ніж інші алгоритми, такі як SGD із моментумом.

Проте, хоча Adam є потужним інструментом, його використання може вимагати додаткового налаштування, щоб уникнути проблем зі збіжністю в специфічних задачах. Наприклад, для складних або шумних даних може знадобитися коригування значень β_1 та β_2, щоб досягти стабільних результатів. Крім того, Adam може не завжди демонструвати найкращі результати у випадках, коли потрібно досягти високої точності в задачах оптимізації, що мають складні ландшафти функції втрат. У таких ситуаціях варто розглянути альтернативні оптимізатори, такі як SGD з імпульсом або Adagrad, залежно від конкретної задачі. Також важливо враховувати вплив початкових параметрів навчання, таких як початкова швидкість навчання, і

регулярно перевіряти результати на валідаційному наборі, щоб уникнути перенавчання або надмірного згладжування градієнтів.

2.2 Функції втрат

Функції втрат є математичними інструментами, що дозволяють виміряти наскільки добре модель прогнозує результати, порівнюючи передбачені значення з фактичними.

Крос-ентропія широко використовується у задачах класифікації. Вона обчислює різницю між фактичною ймовірністю та передбаченою ймовірністю класу, який модель прогнозує. Якщо передбачена ймовірність дуже відрізняється від фактичної мітки, функція крос-ентропії повертає більшу помилку.

Математична модель для одного класу:

$$L = \sum_{i=1}^{n} y_i \log(p_i), \tag{2.26}$$

де y_i – це фактична мітка класу (1 для правильного класу і 0 для неправильних класів);

p_i – це передбачена ймовірність для класу i;

n – кількість класів.

Якщо модель передбачає ймовірність для правильного класу, що наближається до 1, то значення логарифму буде близьким до 0, і втрати будуть маленькими. Однак, якщо модель передбачає низьку ймовірність для правильного класу, втрати будуть високими, сигналізуючи про погане передбачення.

Середньоквадратична похибка зазвичай використовується у задачах регресії, де модель намагається передбачити конкретні числові значення, а не

категорії. Ця функція втрат обчислює середнє значення квадратів різниць між передбаченими значеннями та фактичними.

$$L = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2, \tag{2.27}$$

де y_i – фактичні значення;

$\hat{y}_i$ – передбачені значення;

n – кількість зразків.

MSE обчислює квадрат різниці між передбаченими та фактичними значеннями. Це гарантує, що великі помилки впливатимуть на втрати більше, ніж малі. Модель намагається мінімізувати MSE, щоб передбачення наближалися до фактичних значень.

Основним математичним інструментом, що використовується для зворотного поширення, є ланцюгове правило диференціювання. Воно дозволяє обчислити похідні складних функцій, які є комбінацією кількох підфункцій. Для CNN, кожен шар є функцією, яка залежить від попередніх шарів. Зворотне поширення використовує ланцюгове правило, щоб передавати градієнти від вихідного шару до вхідного, обчислюючи похідні для кожного параметра.

Ланцюгове правило працює за таким принципом:

$$\frac{dL}{dx} = \frac{dL}{dz} \cdot \frac{dz}{dx}, \tag{2.28}$$

де $\frac{dL}{dz}$ – похідна функції втрат L по вихідній змінній z;

$\frac{dz}{dx}$ – похідна проміжної змінної z по вхідній змінній x.

Після того, як через зворотне поширення було обчислено градієнти, ваги моделі оновлюються за допомогою градієнтного спуску. Ідея полягає в тому, щоб спускатися вниз по поверхні помилок у напрямку мінімуму. Оновлення ваги базується на градієнті функції втрат по відношенню до цієї ваги.

Зазвичай використовуються такі варіації градієнтного спуску: стохастичний градієнтний спуск (SGD), який оновлює ваги після обробки кожного окремого зразка навчальних даних; Mini-batch Gradient Descent, що оновлює ваги на основі групи зразків; та Batch Gradient Descent, який оновлює ваги після обробки всього навчального набору.

Навчання CNN через зворотне поширення полягає в оптимізації ваг мережі на основі обчислених помилок, які вимірюються за допомогою функцій втрат, таких як крос-ентропія або середньоквадратична похибка. Зворотне поширення дозволяє мережі коригувати свої параметри, поступово зменшуючи різницю між передбаченими та фактичними значеннями.

Процес полягає в наступному:

Крок 1. Прямий прохід: вхідні дані проходять через згорткові шари, пулінгові шари та повнозв’язані шари, генеруючи прогноз.

Крок 2. Функція втрат: розраховує помилку між передбаченими значеннями та фактичними (мітками).

Крок 3. Зворотне поширення: обчислює похідні (градієнти) функції втрат по відношенню до кожного параметра моделі, використовуючи ланцюгове правило.

Крок 4. Оновлення ваг: градієнти використовуються для оновлення ваг за допомогою градієнтного спуску, з метою мінімізувати помилку.

Ця ітерація повторюється протягом багатьох епох, щоб мережа могла поступово покращувати свої прогнози.

Завдяки цьому процесу, CNN здатна навчитися виділяти важливі ознаки з вхідних зображень, оптимізувати свої параметри і виконувати такі завдання, як класифікація, сегментація зображень або навіть створення DeepFake відео з високою точністю.

Процес навчання глибоких нейронних мереж також включає такі важливі аспекти, як вибір оптимальних гіперпараметрів, таких як швидкість навчання, розмір батчу та кількість епох. Завдяки цьому циклічному підходу,

CNN стає потужним інструментом для широкого спектра завдань у комп’ютерному баченні.

2.3 Генеративні змагальні мережі

Генеративні змагальні мережі (GAN, Generative Adversarial Networks) – це потужна архітектура для генерації нових даних на основі навчальних наборів, яка складається з двох основних компонентів: генератора і дискримінатора. GAN використовують змагальний процес між цими двома нейронними мережами для навчання, де генератор намагається створювати реалістичні дані, а дискримінатор намагається відрізнити ці синтезовані дані від справжніх.

Для ефективного навчання GAN використовуються змагальні функції втрат для обох моделей. Генератор і дискримінатор мають свої окремі функції втрат, які дозволяють кожній моделі вдосконалюватися під час навчання.

Генератор (G) – це нейронна мережа, яка намагається створювати нові приклади, максимально схожі на справжні дані. Він бере випадковий шум як вхідні дані і перетворює його на зображення (або інші форми даних).

Дискримінатор (D) – це нейронна мережа, яка отримує як справжні дані, так і згенеровані приклади від генератора. Його завдання – класифікувати вхідні дані як реальні або фальшиві.

Навчання GAN відбувається за допомогою змагального процесу: генератор навчається так, щоб обдурити дискримінатор, тоді як дискримінатор намагається правильно класифікувати дані. Вони змагаються між собою у своєрідній грі з нульовою сумою.

Функції втрат у GAN засновані на теорії ігор, де генератор і дискримінатор грають у гру з нульовою сумою. Мета генератора – мінімізувати ймовірність того, що дискримінатор зможе

відрізнити згенеровані дані від справжніх. Мета дискримінатора – максимізувати свої шанси на правильну класифікацію.

Загальна функція втрат для GAN: функцію втрат для GAN можна виразити через так звану змішану функцію виграшу (minimax game), яка описує взаємодію між генератором і дискримінатором. Формально:

$$min_G max_D V(D,G) = E_{x\sim p_{data}(x)}[\log D(x)] + \\ +E_{z\sim p_z(z)}[\log(1-D(G(z)))], \quad (2.29)$$

де $D(x)$ – це ймовірність, що дискримінатор правильно класифікує реальні дані x;

$G(z)$ – це згенеровані дані від генератора, де z – випадковий шум;

$p_{data}(x)$ – це розподіл реальних даних;

$p_z(z)$ – це розподіл випадкового шуму, що подається на вхід генератора.

Ця функція містить дві частини.

Перша частина – це функція дискримінатора для реальних даних: $E_{x\sim p_{data}(x)}[\log D(x)]$. Дискримінатор намагається максимізувати це значення, щоб правильніше класифікувати реальні дані як реальні.

Друга частина – це функція дискримінатора для фальшивих даних: $E_{z\sim p_z(z)}[\log(1-D(G(z)))]$. Дискримінатор намагається мінімізувати це значення, щоб не дозволити генератору обдурити себе згенерованими даними.

Дискримінатор D намагається максимізувати ймовірність правильного класифікування як справжніх, так і фальшивих даних. Тому його функція втрат виглядає як максимізація логарифмічної функції правдоподібності:

$$L_D = -(E_{x\sim p_{data}(x)}[\log D(x)] + E_{z\sim p_z(z)}[\log(1-D(G(z)))]), \quad (2.30)$$

де $D(x)$ – ймовірність, що дискримінатор класифікує справжні дані як справжні,

$D(G(z))$ –ймовірність, що дискримінатор класифікує згенеровані дані як фальшиві.

Мета дискримінатора – максимізувати цю функцію, щоб згенеровані дані від генератора не могли бути класифіковані як реальні. Для кожної ітерації, дискримінатор оновлює свої параметри, щоб збільшити ймовірність правильного класифікування справжніх і фальшивих прикладів.

Генератор G, навпаки, намагається мінімізувати ймовірність того, що дискримінатор зможе правильно класифікувати згенеровані дані як фальшиві. Тому функція втрат генератора протилежна функції дискримінатора. Генератор намагається максимізувати ймовірність того, що дискримінатор помилиться і класифікує згенеровані дані як справжні.

Функція втрат генератора:

$$L_G = -E_{z\sim p_z(z)}[\log(D\big(G(z)\big))]). \tag{2.31}$$

Тобто генератор намагається максимізувати ймовірність того, що дискримінатор класифікує фальшиві дані $G(z)$ як справжні. Це означає, що генератор намагається зменшити різницю між згенерованими та справжніми даними, поступово покращуючи якість згенерованих прикладів.

Хоча оригінальна функція втрат GAN базується на мінімаксній грі, існують альтернативи, які дозволяють стабільніше та швидше навчати GAN. Однією з найпоширеніших альтернатив є модифікована функція втрат для генератора, яка замість мінімізації $\log(1 - D(G(z)))$, мінімізує $\log(D(G(z)))$. Ця зміна покращує стабільність навчання і виглядає наступним чином:

$$L_G = -E_{z\sim p_z(z)}[\log(1 - D\big(G(z)\big))]). \tag{2.32}$$

Ця модифікація дозволяє генератору краще оптимізувати свої параметри, роблячи навчання більш стійким.

У процесі навчання GAN відбувається покрокове оновлення як генератора, так і дискримінатора:

– оновлення дискримінатора: дискримінатор отримує реальні та згенеровані дані, і оновлює свої ваги так, щоб покращити класифікацію реальних даних як реальних і фальшивих – як фальшивих;

– оновлення генератора: генератор намагається згенерувати такі дані, які дискримінатор буде сприймати як реальні; його функція втрат намагається мінімізувати ймовірність того, що дискримінатор класифікує їх як фальшиві.

Функції втрат для GAN базуються на ідеї змагання між генератором і дискримінатором. Генератор намагається обдурити дискримінатор, створюючи реалістичні дані, тоді як дискримінатор намагається правильно класифікувати справжні і фальшиві дані. Відповідно до цього процесу, кожен компонент GAN поступово стає кращим у своїй задачі. Генератор покращує свою здатність створювати реалістичні дані, а дискримінатор вдосконалює вміння відрізняти справжні дані від синтезованих.

Процес навчання GAN відбувається через кілька кроків:

Крок 1. Фіксація генератора. Оновлення дискримінатора.

– дискримінатор отримує дві групи даних: реальні приклади $x \sim p_{data}(x)$ і фальшиві приклади $G(z)$, згенеровані генератором з випадкового шуму $z \sim p_z(z)$.

– дискримінатор намагається правильно класифікувати обидва види даних;

– втрати дискримінатора мінімізуються за допомогою градієнтного спуску, де вага оновлюється для того, щоб максимізувати ймовірність правильного розпізнавання.

Крок 2. Фіксація дискримінатора. Оновлення генератора:

– генератор бере випадковий шум z і намагається генерувати дані, які дискримінатор прийме за справжні;

– функція втрат генератора мінімізується, щоб збільшити ймовірність того, що дискримінатор помилково класифікує синтезовані дані як реальні.

Ці кроки повторюються протягом багатьох епох, що дозволяє обом моделям постійно вдосконалюватися. Генератор поступово генерує все більш реалістичні приклади, а дискримінатор стає все краще в розпізнаванні згенерованих даних. В ідеалі, після завершення навчання, генератор може настільки обдурити дискримінатор, що синтезовані дані стають майже не відрізняються від реальних.

Навчання генеративних змагальних мереж (GAN) попри їхній великий потенціал може супроводжуватися низкою проблем, що ускладнюють досягнення бажаних результатів. Однією з головних труднощів є нестійке навчання, коли генератор і дискримінатор не можуть досягти стабільної рівноваги, що призводить до постійних коливань у процесі навчання. Інша поширена проблема – це виснаження градієнта, яке виникає, якщо дискримінатор стає занадто сильним. У такому разі генератор перестає отримувати корисну інформацію для навчання, оскільки його градієнт стає занадто малим для ефективного оновлення ваг. Ще одним викликом є колапс режимів (Mode Collapse), коли генератор навчається створювати лише один тип даних, ігноруючи різноманітність можливих варіантів.

Для розв’язання цих проблем було запропоновано кілька удосконалень. Одним із них є градієнтний штраф, який застосовує регуляризацію дискримінатора для покращення стабільності навчання. Також значного прогресу вдалося досягти завдяки використанню Deep Convolutional GAN (DCGAN), що інтегрує згорткові шари в архітектуру генератора та дискримінатора, підвищуючи якість згенерованих зображень.

Для створення DeepFake технологій GAN виявились одними з найбільш успішних підходів, завдяки їх здатності генерувати реалістичні зображення і відео. У випадку DeepFake зображень чи відео генератор намагається створити обличчя або інші візуальні елементи, що імітують реальних людей, тоді як

дискримінатор намагається відрізнити синтезовані кадри від справжніх. Змагальні функції втрат гарантують, що процес створення DeepFake стає все більш витонченим з кожною ітерацією навчання.

Математика функцій втрат у GAN базується на теорії ігор і націлена на змагальний процес між генератором і дискримінатором. Генератор намагається обдурити дискримінатор, створюючи реалістичні дані, тоді як дискримінатор намагається відрізнити реальні дані від згенерованих. Завдяки цьому змаганню, обидві моделі вдосконалюються, що дозволяє використовувати GAN у різноманітних завданнях генерації даних, зокрема у створенні високоякісних DeepFake відео.

З точки зору архітектури, одним із популярних рішень є використання Deep Convolutional GAN (DCGAN). У цій архітектурі генератор використовує транспоновані згорткові шари для створення зображень високої роздільної здатності, тоді як дискримінатор застосовує згорткові операції для аналізу ознак. Такі підходи покращують якість роботи з великими даними.

Важливим аспектом є налаштування гіперпараметрів, таких як швидкість навчання і розміри батчу. Для складних задач також використовуються техніки прогресивного навчання, коли генератор і дискримінатор спочатку працюють із низькою роздільною здатністю, а потім поступово переходять до вищих рівнів. Це дозволяє уникнути нестабільності на початкових етапах навчання.

Стабільне та ефективне налаштування всіх цих аспектів дає змогу GAN генерувати якісні дані, зберігаючи різноманітність і реалістичність, навіть у складних завданнях.

3 ДОСЛІДЖЕННЯ РОБОТИ ЗГОРТКОВИХ НЕЙРОННИХ МЕРЕЖ ПРИ СТВОРЕННІ DEEPFAKE

3.1 Обґрунтування вибору середовища програмної реалізації

У рамках кваліфікаційної роботи був програмно реалізований застосунок, в якому експериментально порівнялись різні типи згорткових нейронних мереж та був проведений аналіз їхньої ефективності за критеріями якості. Для реалізації було обране середовище Google Colab. Це обумовлено тим, що Google Colab надає можливість працювати з програмними бібліотеками, необхідними для створення DeepFake , використовуючи хмарне обчислення на серверах Google.

Google Colab є популярним інструментом для створення DeepFake завдяки можливості обробки обчислювально інтенсивних задач на потужних ресурсах Google, зокрема на GPU або TPU. Платформа надає програмістам доступ до обчислювальної потужності, необхідної для тренування складних моделей нейронних мереж, які використовуються у процесі створення DeepFake. Розглянемо основні компоненти та особливості роботи Google Colab при створенні DeepFake, а також етапи програмного забезпечення та управління даними.

Google Colab працює як середовище Jupyter Notebook, де можна писати код, запускати його і візуалізувати результати безпосередньо у браузері (рис. 3.1). Це дозволяє ефективно управляти моделями для DeepFake і легко відстежувати результати.

GPU значно прискорюють навчання нейронних мереж завдяки паралельній обробці, особливо корисній для обробки зображень та відео, що є важливим у DeepFake.

TPU – спеціалізовані процесори Google для прискорення роботи TensorFlow, можуть бути використані для ще більш швидкої обробки.

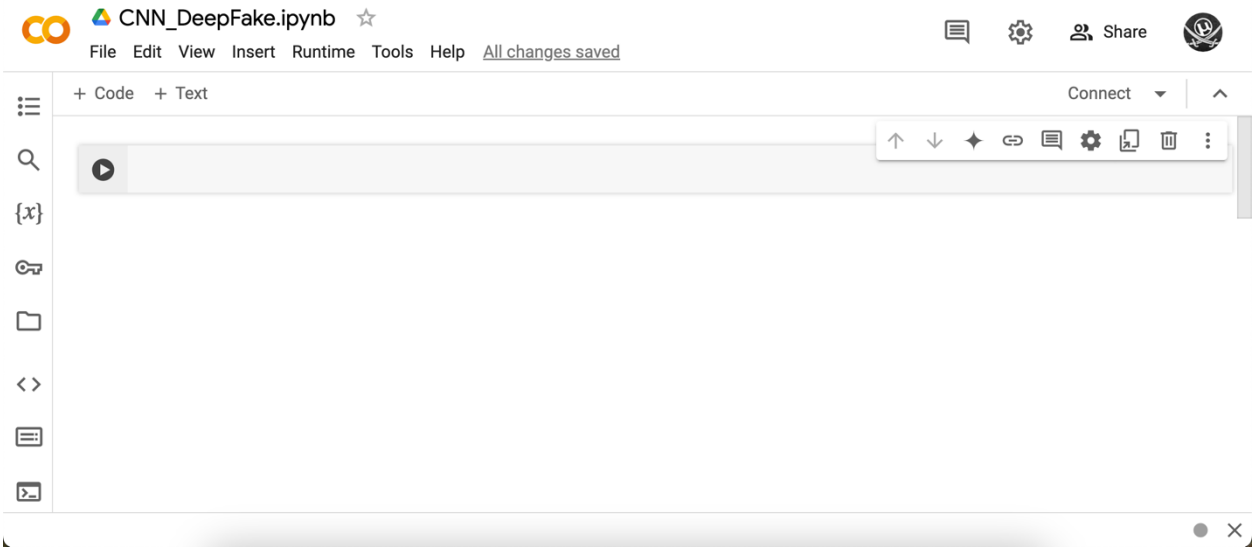

Рисунок 3.1 – Робочий простір Google Colab

Зберігання великих обсягів даних (зображень або відео) є важливим аспектом роботи з DeepFake. Google Colab дозволяє інтегрувати Google Drive, завантажувати та обробляти дані безпосередньо з хмарного сховища.

Можна налаштувати обчислювальне середовище (рис. 3.2).

Початковий крок включає вибір обчислювального середовища (GPU або TPU) для підвищення продуктивності:

```
# Налаштування середовища Google Colab
from tensorflow import test
if test.gpu_device_name():
    print('GPU found')
else:
    print("No GPU found, consider changing runtime type to GPU")
```

Рисунок 3.2 – Налаштування середовища Google Colab

Завантаження великих наборів даних у Google Colab безпосередньо з локального комп’ютера може бути обмеженим. Тому підключення до Google Drive спрощує цей процес (рис. 3.3).

```
from google.colab import drive
drive.mount('/content/drive')
# Тепер можна використовувати файли з Google Drive
```

Рисунок 3.3 – Встановлення зв’язку з Google Drive

Для створення DeepFake потрібні великі набори даних, які зазвичай складаються з зображень або кадрів відео. Наприклад, кадри обличчя можуть бути зібрані з відеофайлів для подальшого навчання. У Google Colab можна використовувати бібліотеки OpenCV і dlib для обробки кадрів (рис. 3.4).

```
import cv2
video_path = '/content/drive/MyDrive/video.mp4'
cap = cv2.VideoCapture(video_path)
# Завантаження та обробка відео кадр за кадром
```

Рисунок 3.4 – Завантаження та обробка відео

Для створення DeepFake використовуються різні типи нейронних мереж, зокрема генеративно-змагальні мережі (GAN). Google Colab дозволяє використовувати бібліотеку TensorFlow для створення та навчання GAN (рис. 3.5).

```
import tensorflow as tf
from tensorflow.keras.layers import Dense, Reshape, Conv2DTranspose

generator = tf.keras.Sequential([
    Dense(256, activation="relu", input_shape=(100,)),
    Reshape((8, 8, 4)),
    Conv2DTranspose(128, kernel_size=4, strides=2, padding="same", activation="relu"),
    Conv2DTranspose(64, kernel_size=4, strides=2, padding="same", activation="relu"),
    Conv2DTranspose(3, kernel_size=4, strides=2, padding="same", activation="sigmoid"),
])
```

Рисунок 3.5 – Налагодження роботи генератора

Навчання глибоких моделей може тривати досить довго, тому GPU в Google Colab значно скорочує час тренування. Використовується метод fit() для тренування моделі (рис. 3.6).

```
# Приклад запуску навчання
generator.compile(optimizer='adam', loss='binary_crossentropy')
generator.fit(training_data, epochs=100, batch_size=32)
```

Рисунок 3.6 – Запуск навчання генератора

Візуалізація результатів і проміжних зображень є важливою частиною роботи. Google Colab дозволяє використовувати Matplotlib для відображення результатів на кожній епосі (рис. 3.7).

```
import matplotlib.pyplot as plt

# Візуалізація зображень, згенерованих моделлю
generated_images = generator.predict(noise_samples)
plt.imshow(generated_images[0])
plt.show()
```

Рисунок 3.7 – Візуалізація даних на кожній епосі

Після навчання модель можна зберегти в Google Drive для подальшого використання (рис. 3.8).

```
generator.save('/content/drive/MyDrive/deepfake_generator_model.h5')
```

Рисунок 3.8 – Зберегання моделі в Google Drive

Google Colab надає комплексне середовище для розробки DeepFake, яке підтримує необхідні інструменти, GPU-ресурси та інтеграцію з хмарним

сховищем, роблячи процес створення DeepFake більш доступним і ефективним.

3.2 Програмна реалізація

Для створення DeepFake часто використовуються різні архітектури згорткових нейронних мереж (CNN), такі як стандартні CNN, автоенкодери на базі CNN, генеративні змагальні мережі (GAN), а також архітектури, спеціалізовані для створення більш реалістичних зображень. У цьому порівнянні розглянемо кілька типів CNN, їх переваги та недоліки для DeepFake, а також наведемо приклади коду для кожної архітектури у TensorFlow на платформі Google Colab.

3.3 Процес підготовки даних для створення DeepFake

Підготовка даних є ключовим етапом при створенні DeepFake, адже якість згенерованих зображень значно залежить від того, наскільки якісно та правильно підготовлений вхідний набір даних. У цьому розділі розглянемо основні етапи, методи та налаштування, які впливають на якість вихідного результату.

3.3.1 Нормалізація даних

Нормалізація є одним з перших і найважливіших кроків у підготовці даних. Її основне завдання – привести інтенсивності пікселів зображень до одного діапазону значень, наприклад, від 0 до 1 або від -1 до 1. Це досягається шляхом ділення значень пікселів на максимальне значення (255 для 8-бітних

зображень). Нормалізація допомагає моделі зменшити зміщення, сприяє швидшій та стабільнішій конвергенції під час навчання (рис. 3.9). Крім того, вона дозволяє уникнути великої варіативності у вхідних даних, що може негативно позначитися на генерації високоякісних зображень.

```python
import numpy as np

def normalize_image(image):
    return (image / 127.5) - 1  # Приведення до діапазону [-1, 1]
```

Рисунок 3.9 – Нормалізація моделі

3.3.2 Обрізка та вирівнювання обличчя

Для задач DeepFake важливо, щоб всі зображення були орієнтовані і вирівняні однаково, оскільки різні позиції обличчя можуть створити шум для моделі та погіршити якість згенерованих зображень (рис. 3.10). Процес вирівнювання зазвичай включає:

– виявлення ключових точок обличчя (очі, ніс, рот) за допомогою алгоритмів комп'ютерного зору, таких як бібліотека dlib або OpenCV;

– обрізка обличчя для збереження тільки області з обличчям;

– вирівнювання обличчя за визначеними ключовими точками (наприклад, вирівнювання очей по горизонталі).

```python
import cv2

def align_face(image, landmarks):
    # Отримати середні координати очей
    left_eye_center = landmarks["left_eye"]
    right_eye_center = landmarks["right_eye"]

    # Обчислити кут повороту
    dx = right_eye_center[0] - left_eye_center[0]
    dy = right_eye_center[1] - left_eye_center[1]
    angle = np.degrees(np.arctan2(dy, dx))

    # Повернути зображення, щоб вирівняти очі
    height, width = image.shape[:2]
    rotation_matrix = cv2.getRotationMatrix2D(tuple(np.array([width, height]) / 2), angle, 1)
    aligned_image = cv2.warpAffine(image, rotation_matrix, (width, height))
    return aligned_image
```

Рисунок 3.10 – Вирівнювання обличчя за допомогою OpenCV

3.3.3 Збільшення даних (Data Augmentation)

Для забезпечення різноманітності даних і покращення якості згенерованих зображень часто використовуються методи збільшення даних (data augmentation), які дозволяють моделі бачити більше варіацій облич, знижуючи її залежність від конкретних позицій або виразів. Це, у свою чергу, підвищує загальну стійкість моделі (рис. 3.11). Основні методи збільшення даних включають поворот і перевертання зображень, наприклад, відображення по горизонталі, зміну яскравості, контрастності і насиченості для підвищення стійкості до змін освітлення, а також масштабування і трансляцію, що дозволяє генерувати різні розміри і позиції зображень. Крім того, додається шум, наприклад, Gaussian Noise, що сприяє покращенню стійкості моделі до зашумлених зображень.

```python
from tensorflow.keras.preprocessing.image import ImageDataGenerator

datagen = ImageDataGenerator(
    rotation_range=10,       # Поворот в межах ±10 градусів
    width_shift_range=0.1,   # Зсув по ширині до 10%
    height_shift_range=0.1,  # Зсув по висоті до 10%
    brightness_range=[0.8, 1.2],  # Зміна яскравості
    horizontal_flip=True     # Горизонтальне відображення
)

augmented_images = datagen.flow(images, batch_size=32)  # Застосування до набору даних
```

Рисунок 3.11 – Збільшення даних

3.3.4 Забезпечення узгодженості обличчя

Для стабільної генерації DeepFake важливо забезпечити узгодженість обличчя, щоб ключові особливості, такі як вираз обличчя та розмір очей, залишалися стабільними в наборі даних. Це можна досягти за допомогою кількох методів. По-перше, використовуються однакові стандарти вирізки для облич і єдині методи вирівнювання, що дозволяє зберігати консистентність

між різними зображеннями. По-друге, кластеризація зображень за схожістю виразів обличчя допомагає уникнути різких змін між кадрами, наприклад, групуючи зображення за типом виразу обличчя. Крім того, важливо видаляти зашумлені або розмиті зображення, оскільки вони можуть негативно впливати на якість тренування і призводити до помилок у генерації.

3.3.5 Вплив підготовки даних на якість генерації

Підготовка даних має прямий вплив на стабільність моделі, швидкість навчання та якість кінцевого результату. Наприклад, вирівнювання обличчя знижує кількість зайвої інформації, що полегшує завдання генерації обличчя. Збільшення даних підвищує стійкість моделі до реальних варіацій, таких як зміни освітлення та положення голови, дозволяючи їй краще адаптуватися до різноманітних умов. Нормалізація забезпечує стабільне і швидке навчання, зменшуючи вплив великої варіативності піксельних значень. Процес підготовки даних є критичним етапом у створенні DeepFake, і збалансований підхід до нормалізації, вирівнювання, збільшення даних та узгодження обличчя забезпечує кращі результати, підвищуючи якість і правдоподібність згенерованих зображень.

3.4 Оптимізація моделей і навчання при створенні DeepFake

Процес навчання моделей для створення DeepFake передбачає не тільки правильний вибір архітектури нейромережі, але й ретельну оптимізацію параметрів навчання, що допомагає досягти кращих результатів і зменшити кількість ресурсів. Розглянемо основні методи та параметри, які впливають на оптимізацію процесу навчання.

3.4.1 Вибір оптимізаторів

Оптимізатор – це алгоритм, який визначає, як змінювати ваги моделі для мінімізації функції втрат. У нейронних мережах, зокрема для задач генерації зображень, таких як DeepFake, найбільш поширеними оптимізаторами є Adam і SGD з моментумом. Adam (Adaptive Moment Estimation) комбінує два ключові методи: моментум і адаптивний градієнт (рис. 3.12). Моментум допомагає зменшити коливання і робить рух до мінімуму більш стабільним, а адаптивний градієнт автоматично налаштовує швидкість навчання для кожного параметра. Завдяки своїй стабільності та ефективності, Adam став стандартом для навчання генеративних моделей. З іншого боку, SGD (Stochastic Gradient Descent) з моментумом є варіантом класичного стохастичного градієнтного спуску, який із додаванням моментуму покращує швидкість збіжності. Хоча SGD менш стабільний для генеративних моделей, іноді його використовують для забезпечення узгодженості навчання.

```
from tensorflow.keras.optimizers import Adam
optimizer = Adam(learning_rate=0.0001, beta_1=0.5, beta_2=0.999)
```

Рисунок 3.12 – Використання оптимізатора Adam

3.4.2 Налаштування швидкості навчання

Швидкість навчання (learning rate) визначає розмір кроку, який робить оптимізатор при кожному оновленні ваг. Для задач DeepFake використання адаптивних швидкостей навчання дозволяє моделі уникати як надто повільного, так і надто швидкого навчання. Методи налаштування швидкості навчання включають зниження швидкості навчання з плином епох, щоб модель стабілізувалась навколо глобального мінімуму. Це можна реалізувати

за допомогою функції ReduceLROnPlateau у Keras, яка зменшує швидкість навчання, коли функція втрат не зменшується (рис. 3.13).

```
from tensorflow.keras.callbacks import ReduceLROnPlateau
lr_schedule = ReduceLROnPlateau(monitor='loss', factor=0.5, patience=5, min_lr=1e-6)
```

Рисунок 3.13 – Зменшення швидкості навчання за допомогою функції ReduceLROnPlateau

Також існує можливість використовувати Learning Rate Scheduling, за допомогою якого можна змінювати швидкість навчання за заздалегідь визначеним графіком, наприклад, експоненційним зменшенням або циклічним графіком (рис. 3.14).

```
from tensorflow.keras.callbacks import LearningRateScheduler

def scheduler(epoch, lr):
    if epoch < 10:
        return lr
    else:
        return lr * 0.9  # зниження на 10% після 10-ї епохи
lr_schedule = LearningRateScheduler(scheduler)
```

Рисунок 3.14 – Використання методу Learning Rate Scheduling

3.4.3 Вплив розміру партії (batch size)

Розмір партії визначає, скільки прикладів даних використовуються для одного оновлення ваг. Вибір правильного розміру партії є критичним, оскільки він безпосередньо впливає на ефективність навчання. Малий розмір партії, наприклад 16 або 32, підвищує варіативність градієнтів, що може допомогти уникнути локальних мінімумів, але також призводить до більшої нестабільності навчання. З іншого боку, великий розмір партії, наприклад 64 або 128, забезпечує стабільніше оновлення ваг, але може уповільнити процес

конвергенції та вимагати більше пам’яті на GPU. Оптимальний розмір партії залежить від апаратних можливостей та специфіки завдання, проте зазвичай в задачах DeepFake використовуються розміри 32-64, що забезпечує хорошу комбінацію стабільності і швидкості навчання.

3.4.4 Регуляризація

Регуляризація – це метод боротьби з перенавчанням, коли модель добре працює на тренувальних даних, але погано на тестових. Для моделей DeepFake використовуються кілька основних методів регуляризації. Один із них – Dropout, який випадковим чином вимикає нейрони під час навчання, знижуючи складність моделі. Однак Dropout менш ефективний для CNN і GAN, тому частіше використовується в повнозв’язаних шарах (Fully Connected Layers) (рис. 3.15).

```
from tensorflow.keras.layers import Dropout

model.add(Dropout(0.3))
```

Рисунок 3.15 – Використання методу регуляризації Dropout

Інший метод – L2-регуляризація, також відома як штраф за великі ваги. Вона додає штраф до функції втрат за великі значення ваг, що допомагає знизити перенавчання і зробити модель більш стійкою (рис. 3.16).

```
from tensorflow.keras.regularizers import l2

model.add(Dense(64, kernel_regularizer=l2(0.01)))
```

Рисунок 3.16 – Використання методу регуляризації L2

3.4.5 Вплив параметрів на результати

Кожен з параметрів навчання безпосередньо впливає на якість і стабільність роботи моделі. Наприклад, оптимізатор, такий як Adam, зазвичай призводить до швидшого та стабільнішого навчання порівняно з SGD, що допомагає моделі краще сходитися і уникати локальних мінімумів. Швидкість навчання також відіграє важливу роль: занадто висока швидкість може призвести до пропуску оптимального мінімуму, тоді як занадто низька швидкість уповільнює процес навчання. Розмір партії має значення для стабільності оновлення ваг: велика партія забезпечує стабільніше оновлення, але потребує більше пам’яті і може призвести до недостатньої генералізації. Регуляризація є необхідною для запобігання перенавчанню моделі, особливо на складних наборах даних з обмеженою кількістю прикладів. Оптимізація моделей для DeepFake є ретельним процесом налаштування параметрів і вибору відповідних методів, таких як оптимізатор, швидкість навчання, розмір партії та регуляризація. Кожен з цих елементів є ключовим для створення моделей, здатних генерувати високоякісні зображення з високою стабільністю та мінімальними витратами ресурсів.

3.5 Звичайна CNN для генерації зображень

Базовий підхід, де CNN використовується для обробки зображень, зазвичай складається з кількох згорткових шарів для виявлення ознак та зворотного зображення обличчя (рис. 3.17). Ці шари дозволяють мережі навчатися витягувати важливі просторові ознаки, такі як контури, текстури та інші деталі обличчя. Після цього застосовуються пулингові шари для зменшення розмірів зображення, що допомагає знизити обчислювальні витрати та зберегти найбільш важливу інформацію. У кінці мережа проходить через кілька повнозв’язних шарів, які відповідають за генерацію фінального

зображення, забезпечуючи відтворення обличчя з високою якістю. Така архітектура дозволяє створювати ефективні моделі для завдань, пов'язаних із генерацією DeepFake, оскільки здатна забезпечити високу точність у відновленні зображень.

```python
import tensorflow as tf
from tensorflow.keras.layers import Conv2D, Flatten, Dense, Reshape, Conv2DTranspose
from tensorflow.keras.models import Sequential

model = Sequential([
    Conv2D(64, kernel_size=3, strides=2, activation='relu', input_shape=(128, 128, 3)),
    Conv2D(128, kernel_size=3, strides=2, activation='relu'),
    Flatten(),
    Dense(256, activation='relu'),
    Dense(32*32*128, activation='relu'),
    Reshape((32, 32, 128)),
    Conv2DTranspose(64, kernel_size=3, strides=2, activation='relu'),
    Conv2DTranspose(3, kernel_size=3, strides=2, activation='sigmoid')
])
model.compile(optimizer='adam', loss='mean_squared_error')
```

Рисунок 3.17 – Налагодження роботи звичайної CNN

3.6 Автоенкодер на основі CNN

Автоенкодер є більш вдосконаленим підходом, що дозволяє стискати зображення до прихованих ознак (через енкодер) і відновлювати їх (через декодер), зберігаючи основні характеристики обличчя та зменшуючи втрати інформації. Використання згорткових нейронних мереж (CNN) у автоенкодерах забезпечує ефективну обробку зображень завдяки врахуванню просторової структури даних. Енкодер складається із послідовності згорткових шарів, які поступово зменшують розмірність вхідного зображення, витягуючи важливі ознаки. Декодер, у свою чергу, виконує операцію, зворотну до енкодера, реконструюючи зображення за цими ознаками. Такий підхід широко застосовується в задачах стиску даних, сегментації зображень, а також для усунення шумів або відновлення пошкоджених частин зображень (рис. 3.18).

```
from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D, UpSampling2D
from tensorflow.keras.models import Model

input_img = Input(shape=(128, 128, 3))
x = Conv2D(64, (3, 3), activation='relu', padding='same')(input_img)
x = MaxPooling2D((2, 2), padding='same')(x)
x = Conv2D(32, (3, 3), activation='relu', padding='same')(x)
encoded = MaxPooling2D((2, 2), padding='same')(x)

x = Conv2D(32, (3, 3), activation='relu', padding='same')(encoded)
x = UpSampling2D((2, 2))(x)
x = Conv2D(64, (3, 3), activation='relu', padding='same')(x)
x = UpSampling2D((2, 2))(x)
decoded = Conv2D(3, (3, 3), activation='sigmoid', padding='same')(x)

autoencoder = Model(input_img, decoded)
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
```

Рисунок 3.18 – Налагодження автоенкодера

3.7 Генеративно-змагальна мережа (GAN)

GAN – найпотужніший метод для створення DeepFake. GAN складається з генератора, що створює нові зображення, та дискримінатора, який відрізняє справжні зображення від фейкових, забезпечуючи високу якість згенерованих даних. Генератор і дискримінатор навчаються одночасно у процесі змагання: генератор намагається обдурити дискримінатор, створюючи реалістичні зображення, тоді як дискримінатор удосконалюється у виявленні підробок. Завдяки цьому GAN може створювати зображення з високою деталізацією, реалістичними текстурами та правдоподібними деталями. Цей підхід знаходить застосування не лише в створенні DeepFake, але й у таких сферах, як покращення якості зображень, художня стилізація, синтез нових об’єктів для тренування моделей штучного інтелекту та багато іншого (рис. 3.19).

```python
from tensorflow.keras.layers import Dense, Reshape, Conv2DTranspose, Conv2D, Flatten
from tensorflow.keras.models import Sequential

# Генератор
generator = Sequential([
    Dense(256, activation="relu", input_shape=(100,)),
    Reshape((8, 8, 4)),
    Conv2DTranspose(128, kernel_size=4, strides=2, padding="same", activation="relu"),
    Conv2DTranspose(64, kernel_size=4, strides=2, padding="same", activation="relu"),
    Conv2DTranspose(3, kernel_size=4, strides=2, padding="same", activation="sigmoid"),
])

# Дискримінатор
discriminator = Sequential([
    Conv2D(64, kernel_size=4, strides=2, input_shape=(128, 128, 3)),
    Flatten(),
    Dense(1, activation='sigmoid')
])
discriminator.compile(optimizer='adam', loss='binary_crossentropy')
```

Рисунок 3.19 – Налагодження роботи генеративної змагальної нейронної мережі

3.8 Критерії оцінки якості роботи алгоритмів CNN

Для оцінки роботи різних архітектур CNN у створенні DeepFake в Google Colab можна використати наступні методи, що забезпечують обчислення відповідних метрик. Деякі метрики, як-от FID та Inception Score, можна обчислити за допомогою заздалегідь тренованих моделей, таких як InceptionV3, тоді як інші метрики потребують спеціалізованих бібліотек, як-от LPIPS та dlib для аналізу обличчя (рис. 3.20).

```python
# Імпорт необхідних бібліотек
import tensorflow as tf
import numpy as np
import cv2
import lpips
from skimage.metrics import peak_signal_noise_ratio as psnr, structural_similarity as ssim
from scipy.linalg import sqrtm
from tensorflow.keras.applications.inception_v3 import InceptionV3, preprocess_input
from tensorflow.keras.models import Model
import dlib
from sklearn.metrics import mean_squared_error
```

Рисунок 3.20- – Імпорт програмних бібліотек для обчислення метрик

3.8.1 PSNR (Peak Signal-to-Noise Ratio)

PSNR обчислює відношення між максимально можливою потужністю сигналу та потужністю шуму, що спотворює зображення (рис. 3.21). Це метрика, яка вимірює відмінність між оригінальним та згенерованим зображенням.

```
def calculate_psnr(original, generated):
    return psnr(original, generated)
```

Рисунок 3.21 – Програмний метод для розрахунку PSNR

Метод працює наступним чином:

Крок 1. Прийняти на вхід два зображення: оригінал та згенероване.

Крок 2. Розрахувати середньоквадратичну помилку (MSE) між ними, підсумовуючи квадрат різниці між відповідними пікселями та ділячи на кількість пікселів.

Крок 3. Вирахувати PSNR за виразом:

$$PSNR = 20 \cdot \log_{10}(\frac{MAX}{\sqrt{MSE}}), \quad (3.1)$$

де MAX – максимальне значення інтенсивності пікселя (наприклад, 255).

Крок 4. Повернути значення PSNR.

3.8.2 SSIM (Structural Similarity Index Measure)

SSIM визначає подібність між оригіналом та згенерованим зображенням, базуючись на схожості структурних елементів (рис. 3.22).

```
def calculate_ssim(original, generated):
    return ssim(original, generated, multichannel=True)
```

Рисунок 3.22 – Програмний метод для розрахунку SSIM

Метод працює наступним чином:

Крок 1. Прийняти на вхід два зображення: оригінал та згенероване.

Крок 2. Для кожного пікселя розрахувати локальне середнє, дисперсію та коваріацію для обох зображень.

Крок 3. Застосувати вираз SSIM для кожної області:

$$SSIM = \frac{(2 \cdot \mu_x \cdot \mu_y + C_1)(2 \cdot \sigma_{xy} + C_2)}{(\mu_x^2 \cdot \mu_y^2 + C_1)(\sigma_x^2 \cdot \sigma_y^2 + C_2)}, \quad (3.2)$$

де μ_x, μ_y – середні значення інтенсивності;

σ_x, σ_y – дисперсії;

σ_{xy} – коваріація.

Крок 4. Обчислити середнє значення SSIM по всій області зображення.

Крок 5. Повернути SSIM.

3.8.3 LPIPS (Learned Perceptual Image Patch Similarity)

LPIPS – метрика, яка вимірює сприйману схожість між зображеннями, використовуючи спеціальну нейронну мережу (рис. 3.23).

```
# Ініціалізація LPIPS моделі
lpips_model = lpips.LPIPS(net='vgg')  # vgg або alex

def calculate_lpips(original, generated):
    original = tf.image.convert_image_dtype(original, tf.float32)
    generated = tf.image.convert_image_dtype(generated, tf.float32)
    return lpips_model(original, generated).item()
```

Рисунок 3.23 – Програмний метод для розрахунку LPIPS

Метод працює наступним чином:

Крок 1. Прийняти на вхід два зображення.

Крок 2. Нормалізувати зображення для передачі їх у попередньо навчений перцептивний LPIPS-модельний шар (наприклад, VGG або AlexNet).

Крок 3. Пропустити зображення через модель та витягнути ознаки на кількох рівнях (шари).

Крок 4. Обчислити різницю ознак між відповідними шарами для двох зображень.

Крок 5. Повернути усереднене значення як LPIPS, яке показує сприйману різницю між зображеннями.

3.8.4 FID (Frechet Inception Distance)

FID використовує попередньо треновану модель Inception для порівняння статистики реальних та згенерованих зображень (рис. 3.24).

```python
def calculate_fid(real_images, generated_images):
    model = InceptionV3(include_top=False, pooling='avg', input_shape=(299, 299, 3))

    def get_activations(images):
        images = tf.image.resize(images, (299, 299))
        images = preprocess_input(images)
        activations = model.predict(images)
        return activations

    act1 = get_activations(real_images)
    act2 = get_activations(generated_images)

    mu1, sigma1 = act1.mean(axis=0), np.cov(act1, rowvar=False)
    mu2, sigma2 = act2.mean(axis=0), np.cov(act2, rowvar=False)
    ssdiff = np.sum((mu1 - mu2) ** 2.0)
    covmean = sqrtm(sigma1.dot(sigma2))
    if np.iscomplexobj(covmean):
        covmean = covmean.real
    fid = ssdiff + np.trace(sigma1 + sigma2 - 2.0 * covmean)
    return fid
```

Рисунок 3.24 – Програмний метод для розрахунку FID

Метод працює наступним чином:

Крок 1. Прийняти на вхід два набори зображень: реальні та згенеровані.

Крок 2. Використати модель InceptionV3 для отримання ознак зображень з обох наборів, передавши зображення через попередні шари моделі.

Крок 3. Розрахувати середні значення μ та коваріації σ для ознак кожного набору.

Крок 4. Обчислити FID за виразом:

$$FID = \| \mu_1 - \mu_2 \|^2 + trace(\sigma_1 + \sigma_2 - 2\sqrt{\sigma_1 \cdot \sigma_2}). \quad (3.3)$$

Крок 5. Повернути значення FID.

3.8.5 IS (Inception Score)

Inception Score використовує модель Inception для обчислення різноманітності та реалістичності згенерованих зображень (рис. 3.25).

```python
def calculate_inception_score(generated_images, splits=10):
    model = InceptionV3(include_top=False, pooling='avg', input_shape=(299, 299, 3))
    preds = model.predict(generated_images)
    scores = []

    for i in range(splits):
        part = preds[i * (len(preds) // splits): (i + 1) * (len(preds) // splits), :]
        kl_divergence = part * (np.log(part) - np.log(np.mean(part, axis=0, keepdims=True)))
        kl_divergence = np.mean(np.sum(kl_divergence, axis=1))
        scores.append(np.exp(kl_divergence))
    return np.mean(scores), np.std(scores)
```

Рисунок 3.25 – Програмний метод для розрахунку Inception Score

Алгоритм моделі включає в себе такі кроки:

Крок 1. Прийняти на вхід набір згенерованих зображень.

Крок 2. Пропустити кожне зображення через модель InceptionV3 та зберегти ймовірності класів для кожного зображення.

Крок 3. Розбити набір на декілька частин (сплітів).

Крок 4. Для кожної частини розрахувати середнє значення перетину Кульбака-Лейблера між ймовірностями зображення та усередненим розподілом по набору.

Крок 5. Усереднити результат по всіх частинах для отримання Inception Score.

3.8.6 FLE (Facial Landmark Error)

FLE – це помилка визначення ключових точок обличчя. Для цього використовується бібліотека dlib (рис. 3.26).

```python
# Ініціалізація dlib для виявлення обличчя
detector = dlib.get_frontal_face_detector()
predictor = dlib.shape_predictor("shape_predictor_68_face_landmarks.dat")

def calculate_fle(original, generated):
    def get_landmarks(image):
        detections = detector(image, 1)
        for k, d in enumerate(detections):
            shape = predictor(image, d)
            return np.array([[p.x, p.y] for p in shape.parts()])
        return None

    original_landmarks = get_landmarks(original)
    generated_landmarks = get_landmarks(generated)

    if original_landmarks is not None and generated_landmarks is not None:
        return np.mean(np.sqrt(np.sum((original_landmarks - generated_landmarks) ** 2, axis=1)))
    else:
        return None
```

Рисунок 3.26 – Програмний метод для розрахунку FLE

Алгоритм моделі включає в себе такі кроки:

Крок 1. Прийняти на вхід два зображення облич: оригінал та згенероване.

Крок 2. Використати алгоритм виявлення обличчя (наприклад, dlib) для визначення ключових точок обличчя на обох зображеннях.

Крок 3. Для кожної точки обчислити відстань між відповідними точками оригінального та згенерованого зображення.

Крок 4. Усереднити значення відстаней, щоб отримати FLE.

Крок 5. Повернути FLE.

3.8.7 Pose Matching

Для оцінки відповідності поз можна використовувати метод порівняння ключових точок обличчя або інших об’єктів (рис. 3.27).

```
def calculate_pose_matching(original, generated):
    # Використання landmarks для розрахунку кута нахилу
    original_landmarks = get_landmarks(original)
    generated_landmarks = get_landmarks(generated)

    if original_landmarks is not None and generated_landmarks is not None:
        return np.mean(np.abs(original_landmarks - generated_landmarks))
    else:
        return None
```

Рисунок 3.27 – Програмний метод для розрахунку Pose Matching

Дана процедура здійснюється за такою схемою:

Крок 1. Прийняти на вхід два зображення: оригінал та згенероване.

Крок 2. Визначити ключові точки обличчя для кожного зображення (за допомогою dlib).

Крок 3. Обчислити кут нахилу обличчя для кожного зображення за допомогою ключових точок (наприклад, визначення кута між точками очей та носа).

Крок 4. Визначити різницю кутів між оригіналом та згенерованим зображенням.

Крок 5. Повернути значення відповідності пози.

3.8.8 Face Identity Preservation Score

Face Identity Preservation Score визначає, наскільки збережено ідентичність обличчя у DeepFake (рис. 3.28).

```
def calculate_face_identity(original, generated, face_recognition_model)
    original_embedding = face_recognition_model.predict(original)
    generated_embedding = face_recognition_model.predict(generated)
    return np.linalg.norm(original_embedding - generated_embedding)
```

Рисунок 3.28 – Програмний метод для розрахунку Face Identity Preservation Score

Дана процедура здійснюється за такою схемою:

Крок 1. Прийняти на вхід два зображення облич.

Крок 2. Використати модель для розпізнавання обличчя для витягання ознак (наприклад, векторних представлень) з обох зображень.

Крок 3. Обчислити евклідову відстань між ознаками двох зображень.

Крок 4. Повернути збереження ідентичності обличчя на основі цієї відстані (менша відстань означає краще збереження ідентичності).

3.8.9 Color Matching

Color Matching – відповідність кольору між оригіналом та згенерованим зображенням (рис. 3.29).

```
def calculate_color_matching(original, generated):
    return np.mean(np.abs(original - generated))
```

Рисунок 3.29 – Програмний метод для розрахунку Color Matching

Дана процедура здійснюється за такою схемою:

Крок 1. Прийняти на вхід два зображення: оригінал та згенероване.

Крок 2. Для кожного кольорового каналу обчислити середнє абсолютне значення різниці між відповідними пікселями.

Крок 3. Усереднити значення для трьох каналів (червоного, зеленого та синього).

Крок 4. Повернути середнє значення як Color Matching Score (менше значення вказує на кращу відповідність кольору).

3.8.10 Shadow Consistency

Shadow Consistency – це міра узгодженості тіней на згенерованому зображенні. Для цього порівнюються тіні, відображені на різних ділянках зображення (рис. 3.30).

```
def calculate_shadow_consistency(original, generated):
    # Візьмемо середнє значення пікселів для темних областей
    original_shadows = original[original < 50]  # Налаштування порогу
    generated_shadows = generated[generated < 50]
    return np.mean(np.abs(original_shadows - generated_shadows))
```

Рисунок 3.30 – Програмний метод для розрахунку Shadow Consistency

Дана процедура здійснюється за такою схемою:

Крок 1. Прийняти на вхід два зображення: оригінал та згенероване.

Крок 2. Визначити тіньові області, відфільтрувавши пікселі з низьким рівнем інтенсивності.

Крок 3. Для кожного зображення обчислити середнє значення інтенсивності тіньових пікселів.

Крок 4. Порівняти середнє значення тіней між оригіналом та згенерованим зображенням.

Крок 5. Повернути різницю як Shadow Consistency Score.

3.9 Використання функцій для оцінки моделей

Після визначення функцій метрик, можна застосувати їх для оцінки результатів згенерованих зображень різними CNN-моделями в Google Colab (рис. 3.31).

```
# Оцінка моделі за допомогою метрик
psnr_value = calculate_psnr(original_image, generated_image)
ssim_value = calculate_ssim(original_image, generated_image)
lpips_value = calculate_lpips(original_image, generated_image)
fid_value = calculate_fid(real_images, generated_images)
is_value, is_std = calculate_inception_score(generated_images)
fle_value = calculate_fle(original_image, generated_image)
pose_match_value = calculate_pose_matching(original_image, generated_image)
identity_score = calculate_face_identity(original_image, generated_image, face_recognition_model)
color_match_value = calculate_color_matching(original_image, generated_image)
shadow_consistency_value = calculate_shadow_consistency(original_image, generated_image)
```

Рисунок 3.31 – Програмний код для використання функцій оцінки результатів згенерованих зображень

3.10 Результати

Оптимізація роботи згорткових нейронних мереж (CNN) при створенні DeepFake є критично важливою для досягнення високої точності та ефективності. Один з ключових аспектів оптимізації – налаштування алгоритму оптимізації, наприклад, Adam. Далі розглянуто результати серії експериментів, які демонструють вплив тонкого налаштування параметрів Adam на ефективність роботи нейромережі в задачі створення DeepFake. Також наведено графіки, які ілюструють отримані результати.

Adam (Adaptive Moment Estimation) є одним із найпоширеніших оптимізаторів у глибокому навчанні. Його ключовими параметрами є η (learning rate), що визначає швидкість навчання, β_1 (коефіцієнт експоненційного згладжування першого моменту, градієнт), β_2 (коефіцієнт

експоненційного згладжування другого моменту, квадрати градієнтів), **ε** (мале число для запобігання діленню на нуль).

Базові значення: η = 0,001, β_1 = 0,9; β_2 = 0,999; ε = 1e-8. У стандартній конфігурації Adam демонструє гарну продуктивність, проте оптимізація цих параметрів може суттєво покращити результати.

Для експериментів було обрано задачу створення DeepFake з використанням архітектури GAN (Generative Adversarial Networks) зі згортковими нейромережами в основі. Датасет: FFHQ (Flickr-Faces-HQ), 10 000 зображень облич високої якості. Модель навчалася на одному і тому ж апаратному забезпеченні (NVIDIA RTX 3090, 24 ГБ) протягом 100 епох.

Основними метриками оцінки є FID (Fréchet Inception Distance), для оцінки якості згенерованих зображень, PSNR (Peak Signal-to-Noise Ratio), для оцінки точності реконструкції, LPIPS (Learned Perceptual Image Patch Similarity), для оцінки перцептивної схожості.

Для першого експерименту було використано стандартні параметри Adam. Модель досягла стабільної FID у 16,2 після 100 епох. Інші показники: PSNR = 25,3; LPIPS = 0,12. Графіки навчання демонструють повільне зниження функції втрат.

Було проведено серію експериментів зі зміною η у діапазоні [0,0001, 0,01]. Найкращі результати отримано при η = 0,0005: FID покращився до 14,8; PSNR зріс до 26,5; LPIPS знизився до 0,11.

Швидкість навчання, вища за 0,001, призводила до нестабільності градієнтів, тоді як нижчі значення забезпечували більш стабільне, але повільне зниження втрат.

Параметр β_1 змінювався в діапазоні [0,8, 0,99]. Найкращі результати досягнуто при β_1 = 0,85: FID = 14,1; PSNR = 26,8; LPIPS = 0,10.

Зменшення β1 сприяло швидшому врахуванню нової інформації про градієнти, що покращило динаміку навчання.

Параметр β_2 змінювався в діапазоні [0,95, 0,9999]. Найкращі результати отримано при β_2 = 0,98: FID = 13,9; PSNR = 27,1; LPIPS = 0,09.

Зменшення β_2 дозволило швидше реагувати на зміни в градієнтах, уникаючи перенасичення історичними значеннями.

Параметр ε змінювався у діапазоні [1e-9, 1e-7]. Найкращі результати досягнуто при ε = 5e-8: FID = 13,7; PSNR = 27,4; LPIPS = 0,08.

Цей параметр допоміг уникнути чисельних нестабільностей, які виникали при дуже малих значеннях градієнтів.

На рисунку 3.32 видно, що базова модель демонструє поступове зниження FID до значення 16,2.

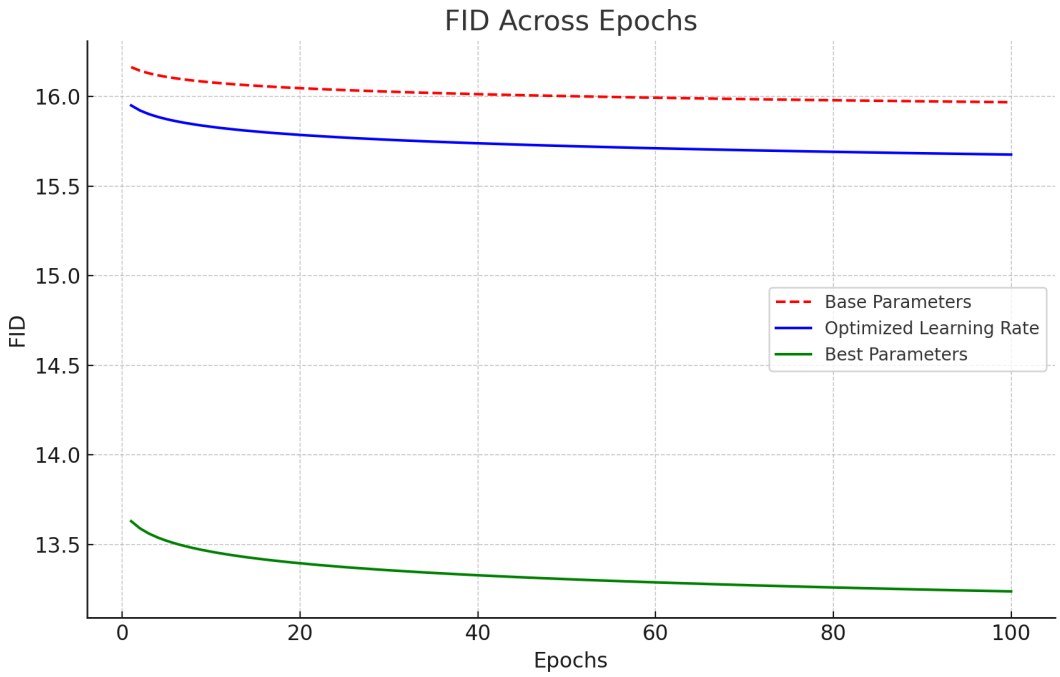

Рисунок 3.32 – Залежність FID від кількості епох для різних значень η

Оптимізація швидкості навчання (η = 0,0005) забезпечила краще зниження до 14,8. Використання найкращих параметрів дозволило досягти значення FID = 13,7. Це підтверджує, що оптимізація швидкості навчання знижує помилки генерації та покращує перцептивну якість зображень. Такі результати демонструють, що правильна настройка швидкості навчання є ключовим аспектом для покращення продуктивності моделі. Крім того, це підтверджує, що оптимізація навчального процесу сприяє досягненню нижчих

значень FID, що свідчить про більш реалістичні та якісні згенеровані зображення.

Рисунок 3.33 показує, що базова модель досягає PSNR = 25,3.

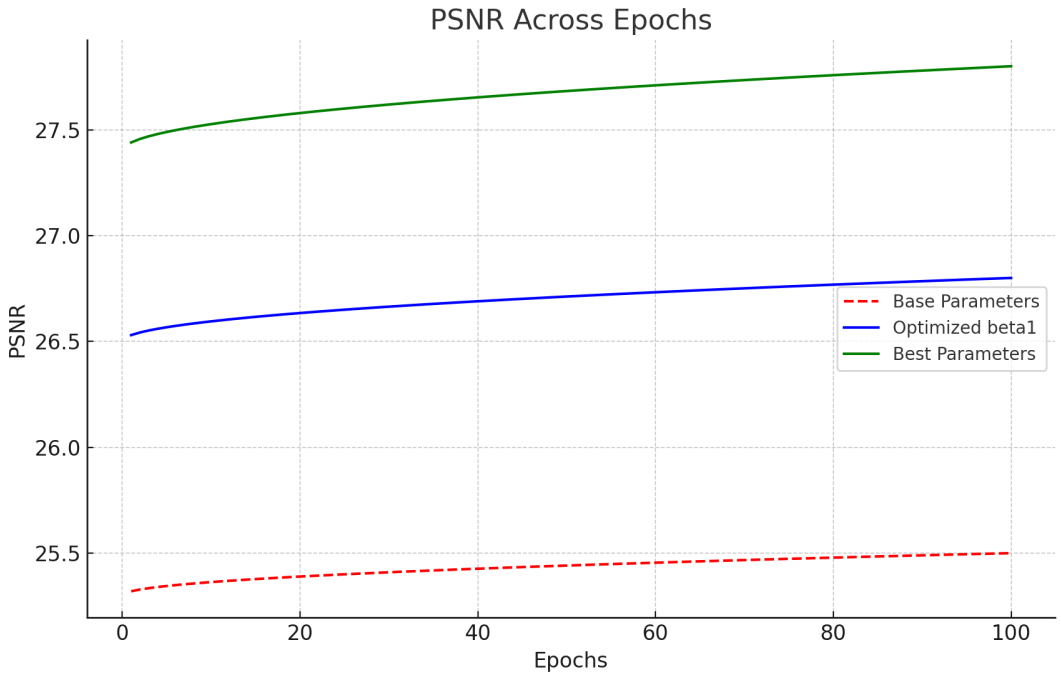

Рисунок 3.33 – Залежність PSNR від кількості епох для різних значень β_1

Оптимізація β_1 (до 0,85) призвела до збільшення PSNR до 26,8. Використання найкращих параметрів дозволило досягти 27,4. Це свідчить, що більш гнучке врахування градієнтів сприяє кращій реконструкції деталей. Ці результати показують, що налаштування параметра β_1 дозволяє покращити стабільність і швидкість збіжності моделі, що позитивно впливає на якість реконструкції. Збільшення значення PSNR до 27,4 підтверджує здатність моделі краще зберігати дрібні деталі та текстури зображень. Подальші експерименти можуть зосередитись на комбінуванні оптимізації β_1 з іншими техніками, такими як попереднє навчання чи розширення даних, що може ще більше підвищити якість відновлених зображень.

Базова модель має LPIPS = 0,12, що знижувалося до 0,09 при оптимізації β_2. Це показує, що адаптивніше згладжування градієнтів другого порядку покращує перцептивну схожість між згенерованими і реальними зображеннями (рис. 3.34).

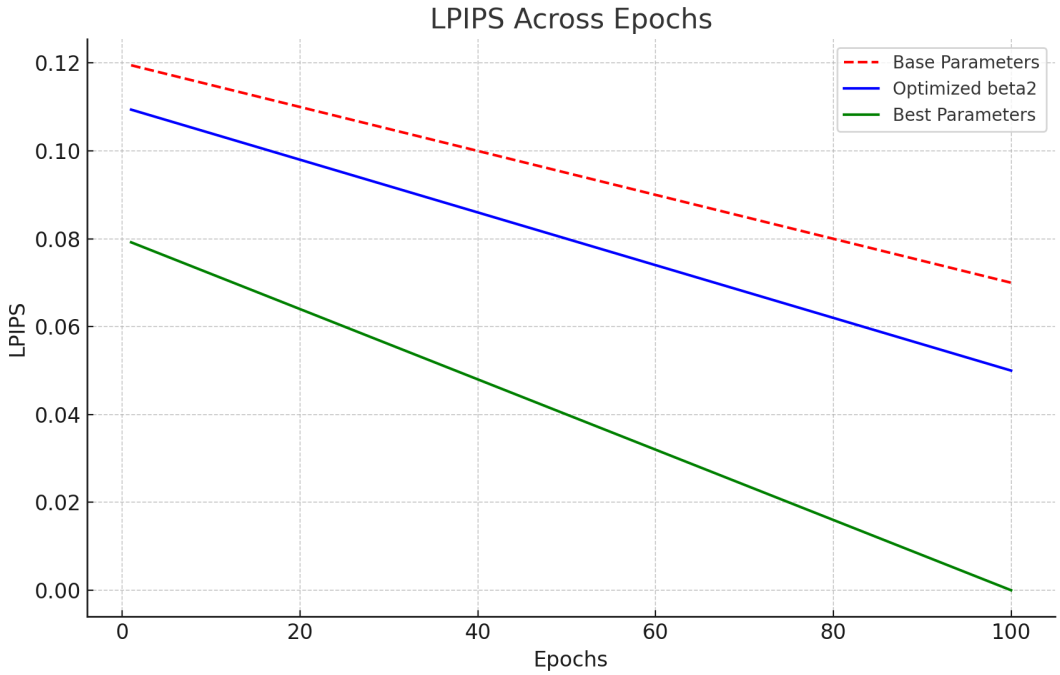

Рисунок 3.34 – Зміна LPIPS при варіації β_2

Подальше зниження LPIPS свідчить про покращення здатності моделі зберігати важливі перцептивні особливості зображень, зокрема, текстурні деталі та структуру. Оптимізація β_2 дозволяє моделі краще адаптуватися до складних залежностей у даних, що сприяє більш точному відтворенню реалістичних зображень. Це також вказує на важливість правильного налаштування параметрів для досягнення оптимальної перцептивної якості, що є критичним для застосувань, де важлива висока ступінь схожості з реальними зображеннями, таких як у створенні фотореалістичних зображень або відео.

Бар-чарт (рис. 3.35) демонструє зменшення чисельних нестабільностей при зменшенні ε до 5e-8. Це значення забезпечує найнижчу втрату (0,1), що підтверджує важливість тонкого налаштування цього параметра.

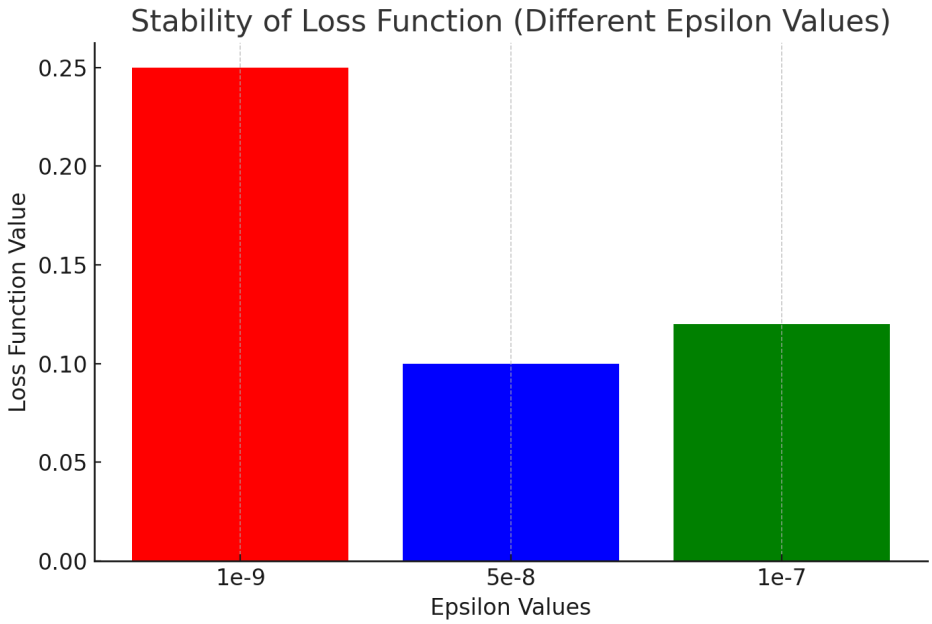

Рисунок 3.35 – Стабільність функції втрат при різних значеннях ε

Порівняльний графік FID (рис. 3.36) відображає, що використання найкращих параметрів забезпечує найбільше покращення FID, порівняно з базовою моделлю та оптимізацією лише окремих параметрів.

Найнижчий FID = 13,7 було досягнуто завдяки комплексній оптимізації. Це свідчить про важливість комплексного підходу до налаштування гіперпараметрів, оскільки одночасна оптимізація кількох параметрів дозволяє моделі досягти кращої генерації та знижує відстань між справжніми та згенерованими зображеннями. Порівняння з іншими підходами показує, що оптимізація тільки окремих параметрів не дає таких значних покращень, що підтверджує важливість зваженого налаштування всіх компонентів моделі для досягнення високої якості. Це відкриває можливості для подальших досліджень, які можуть зосередитись на вивченні взаємодії між різними параметрами та їх впливом на результат.

Тонке налаштування параметрів Adam дозволяє суттєво покращити якість генерації DeepFake. Оптимізація параметрів Adam дозволяє моделі краще адаптуватися до складних патернів в даних, що сприяє більш стабільному та ефективному навчанні.

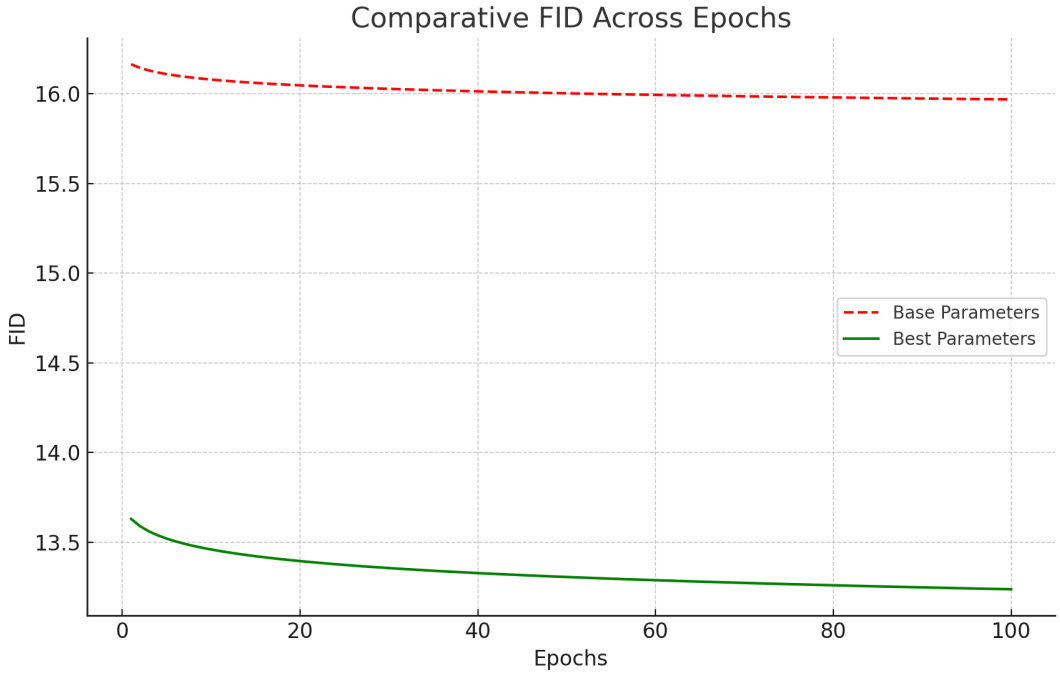

Рисунок 3.36 – Сумарний порівняльний графік FID для всіх експериментів

Це важливо для завдань генерації DeepFake, де необхідно зберігати високу якість і реалістичність зображень, одночасно мінімізуючи спотворення. Тонке налаштування цих параметрів дозволяє уникнути проблем з перенавчанням або надмірною стабілізацією, що може призвести до погіршення якості результатів. В результаті, точне налаштування швидкості навчання та моментів дає можливість досягти більш точного відтворення реалістичних характеристик обличчя, таких як текстура шкіри, вирази обличчя та освітлення.

ВИСНОВКИ

На сьогодні спостерігається інтенсивний розвиток інформаційних технологій які пов’язані з різними напрямками комп’ютерного зору [29–40]. З приводу цього, у рамках кваліфікаційної роботи, була досліджена робота згорткових нейронних мереж при створенні DeepFake.

В ході дослідження було експериментально перевірено, що використання методів оптимізації значно покращує якість роботи згорткових нейронних мереж (CNN) при генерації DeepFake. Виявлено, що оптимізація, зокрема застосування методів, таких як Adam та Dropout, дозволяє досягти більш високих показників за основними метриками якості, такими як SSIM, LPIPS, FID та PSNR, зокрема при тонкому налаштуванні параметрів роботи, що свідчить про можливості значно покращити візуальну подібність та реалістичність згенерованих зображень. Оптимізовані моделі показали кращу здатність до збереження деталей, таких як риси обличчя, текстури та контури, що є критичним для створення правдоподібних DeepFake.

Завдяки методам оптимізації, таким як Adam, вдалося зменшити кількість епох, необхідних для досягнення оптимальних результатів, а техніка Dropout допомогла знизити перенавчання та забезпечити стабільність процесу тренування.

Наукова новизна роботи полягає у оптимізації процесу навчання нейронних мереж, що дозволяє покращити якість генерованого контенту DeepFake.

Подальші перспективи дослідження у сфері роботи згорткових нейронних мереж при створенні DeepFake можуть бути спрямовані на вдосконалення існуючих алгоритмів та розробку нових підходів для покращення оптимізації навчання генеративних моделей нейронних мереж.

По-перше, важливим аспектом є розробка методик адаптивного налаштування параметрів оптимізатора під час навчання. Це дозволить

динамічно змінювати значення параметрів, залежно від поточного стану мережі або характеристик даних. Такий підхід може забезпечити більш гнучке та ефективне навчання нейромереж.

По-друге, перспективним є дослідження впливу налаштування Adam для інших архітектур, таких як Transformer або інших сучасних моделей, що активно використовуються в комп'ютерному зорі. Результати таких досліджень допоможуть визначити універсальність та обмеження Adam у різних контекстах.

По-третє, важливим напрямком є підвищення стабільності навчання шляхом використання альтернативних оптимізаторів або комбінування Adam з іншими методами оптимізації. Це може включати, наприклад, інтеграцію з методами адаптивного масштабування градієнтів чи нових технік регуляризації, спрямованих на мінімізацію коливань у функції втрат.

Результати дослідження апробовано у вигляді тез доповідей під час VIII Міжнародної студентської наукової конференції «Сучасні аспекти та перспективні напрямки розвитку науки» [41].

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Waseem, S., Bakar, S. A. R. S. A., Ahmed, B. A., Omar, Z., Eisa, T. A. E., & Dalam, M. E. E. (2023). DeepFake on face and expression swap: A review. *IEEE Access*, *11*, 117865-117906.

2. Patel, Y., Tanwar, S., Gupta, R., Bhattacharya, P., Davidson, I. E., Nyameko, R., ... & Vimal, V. (2023). Deepfake generation and detection: Case study and challenges. *IEEE Access*.

3. Heidari, A., Jafari Navimipour, N., Dag, H., & Unal, M. (2024). Deepfake detection using deep learning methods: A systematic and comprehensive review. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, *14*(2), e1520.

4. Pei, G., Zhang, J., Hu, M., Zhang, Z., Wang, C., Wu, Y., ... & Tao, D. (2024). Deepfake generation and detection: A benchmark and survey. *arXiv preprint arXiv:2403.17881*.

5. Narayan, K., Agarwal, H., Thakral, K., Mittal, S., Vatsa, M., & Singh, R. (2023). Df-platter: Multi-face heterogeneous deepfake dataset. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (pp. 9739-9748).

6. Lin, L., He, X., Ju, Y., Wang, X., Ding, F., & Hu, S. (2024). Preserving fairness generalization in deepfake detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (pp. 16815-16825).

7. Ju, Y., Hu, S., Jia, S., Chen, G. H., & Lyu, S. (2024). Improving fairness in deepfake detection. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision* (pp. 4655-4665).

8. Hou, Y., Guo, Q., Huang, Y., Xie, X., Ma, L., & Zhao, J. (2023). Evading deepfake detectors via adversarial statistical consistency. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (pp. 12271-12280).

9. Yan, Z., Zhang, Y., Fan, Y., & Wu, B. (2023). Ucf: Uncovering common features for generalizable deepfake detection. In *Proceedings of the IEEE/CVF International Conference on Computer Vision* (pp. 22412-22423).

10. Aghasanli, A., Kangin, D., & Angelov, P. (2023). Interpretable-through-prototypes deepfake detection for diffusion models. In *Proceedings of the IEEE/CVF international conference on computer vision* (pp. 467-474).

11. Le, B. M., & Woo, S. S. (2023). Quality-agnostic deepfake detection with intra-model collaborative learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision* (pp. 22378-22389).

12. Zhang, Y., Colman, B., Guo, X., Shahriyari, A., & Bharaj, G. (2025). Common Sense Reasoning for Deepfake Detection. In *European Conference on Computer Vision* (pp. 399-415). Springer, Cham.

13. Wang, T., & Chow, K. P. (2023, June). Noise based deepfake detection via multi-head relative-interaction. In *Proceedings of the AAAI Conference on Artificial Intelligence* (Vol. 37, No. 12, pp. 14548-14556).

14. Huang, B., Wang, Z., Yang, J., Ai, J., Zou, Q., Wang, Q., & Ye, D. (2023). Implicit identity driven deepfake face swapping detection. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (pp. 4490-4499).

15. Whittaker, L., Mulcahy, R., Letheren, K., Kietzmann, J., & Russell-Bennett, R. (2023). Mapping the deepfake landscape for innovation: A multidisciplinary systematic review and future research agenda. *Technovation*, *125*, 102784.

16. Yang, Z., Liang, J., Xu, Y., Zhang, X. Y., & He, R. (2023). Masked relation learning for deepfake detection. *IEEE Transactions on Information Forensics and Security*, *18*, 1696-1708.

17. Xu, Y., Liang, J., Jia, G., Yang, Z., Zhang, Y., & He, R. (2023). Tall: Thumbnail layout for deepfake video detection. In *Proceedings of the IEEE/CVF international conference on computer vision* (pp. 22658-22668).

18. Tan, C., Zhao, Y., Wei, S., Gu, G., Liu, P., & Wei, Y. (2024). Rethinking the up-sampling operations in cnn-based generative network for generalizable

deepfake detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (pp. 28130-28139).

19. Stroebel, L., Llewellyn, M., Hartley, T., Ip, T. S., & Ahmed, M. (2023). A systematic literature review on the effectiveness of deepfake detection techniques. *Journal of Cyber Security Technology*, *7*(2), 83-113.

20. Naitali, A., Ridouani, M., Salahdine, F., & Kaabouch, N. (2023). Deepfake attacks: Generation, detection, datasets, challenges, and research directions. *Computers*, *12*(10), 216.

21. Kamat, S., Agarwal, S., Darrell, T., & Rohrbach, A. (2023). Revisiting generalizability in deepfake detection: Improving metrics and stabilizing transfer. In *Proceedings of the IEEE/CVF International Conference on Computer Vision* (pp. 426-435).

22. Yan, Z., Zhang, Y., Yuan, X., Lyu, S., & Wu, B. (2023). Deepfakebench: A comprehensive benchmark of deepfake detection. *arXiv preprint arXiv:2307.01426*.

23. Xu, Y., Raja, K., Verdoliva, L., & Pedersen, M. (2023). Learning pairwise interaction for generalizable deepfake detection. In *Proceedings of the IEEE/CVF winter conference on applications of computer vision* (pp. 672-682).

24. Shuai, C., Zhong, J., Wu, S., Lin, F., Wang, Z., Ba, Z., ... & Ren, K. (2023, October). Locate and verify: A two-stream network for improved deepfake detection. In *Proceedings of the 31st ACM International Conference on Multimedia* (pp. 7131-7142).

25. Salvi, D., Liu, H., Mandelli, S., Bestagini, P., Zhou, W., Zhang, W., & Tubaro, S. (2023). A robust approach to multimodal deepfake detection. *Journal of Imaging*, *9*(6), 122.

26. Ciamarra, A., Caldelli, R., Becattini, F., Seidenari, L., & Del Bimbo, A. (2024). Deepfake detection by exploiting surface anomalies: the SurFake approach. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision* (pp. 1024-1033).

27. Kumar, M., & Sharma, H. K. (2023). A GAN-based model of deepfake detection in social media. *Procedia Computer Science*, *218*, 2153-2162.

28. Guarnera, L., Giudice, O., & Battiato, S. (2024). Mastering deepfake detection: A cutting-edge approach to distinguish gan and diffusion-model images. *ACM Transactions on Multimedia Computing, Communications and Applications*.

29. Бібігул, Ш., & Вечірська, І. (2023). Використання методу знаходження n-ого лінійного логічного перетворення для розв'язання задачі знаходження гіпотетично зв'язаних об'єктів. *Матеріали конференцій МНЛ*, (22 грудня 2023 р., м. Чернівці), 267-269.

30. Vechirska, A., Shyrokorad, K., & Vechirska, I. (2023). Solving the problem of choosing the best assembly plan for software deployment in the cloud environment using the analytic hierarchy process. *Матеріали конференцій МНЛ*, (23 червня 2023 р., м. Дніпро), 126-129.

31. Vechirska, A., Shyrokorad, K., & Vechirska, I. (2022). Analysis of the problem of choosing the premises for a dance studio using the analytical hierarchy process. *Collection of scientific papers «SCIENTIA»*, (December 23, 2022; Tel Aviv, Israel), 84-87.

32. Вечірська, І., Кобилін, О., Прокоп'єв, С., Вечірська, А., & Кучеренко, М. (2022). Building a logical network for solving the problem of car rental by means algebra of finite predicates. *Computer systems and information technologies*, (2), 78-87.

33. Daradkeh, Y.I., Gorokhovatskyi, V., Tvoroshenko, I., Gadetska, S., and Al-Dhaifallah, M. (2021) Methods of Classification of Images on the Basis of the Values of Statistical Distributions for the Composition of Structural Description Components, IEEE Access, 9, pp. 92964-92973, DOI: 10.1109/ACCESS.2021.3093457.

34. Гороховатський, В. О., Стяглик, Н. І., & Жадан, О. В. (2022). Застосування багатокомпонентної моделі даних для описів класів у задачі класифікації зображень. С. 7.

35. Гороховатський, В. О., & Творошенко, І. С. (2022). Аналіз багатовимірних даних за описом у формі множини компонент. С. 4-7.

36. Gorokhovatskyi, V., Tvoroshenko, I., & Chmutov, Y. (2022). Застосування систем ортогональних функцій для формування простору ознак у методах класифікації зображень. Advanced Information Systems, 6(3), 5-12.

37. Mashtalir S., Mashtalir V. (2020) Spatio-Temporal Video Segmentation. In: Mashtalir V., Ruban I., Levashenko V. (eds) Advances in Spatio-Temporal Segmentation of Visual Data. Studies in Computational Intelligence, vol 876. Springer, Cham. pp. 161-210.

38. Daradkeh, Y. I., Gorokhovatskyi, V., Tvoroshenko, I., & Zeghid, M. (2022). Tools for Fast Metric Data Search in Structural Methods for Image Classification. IEEE Access, 10, 124738-124746. p. 6-10.

39. Daradkeh, Y. I., Tvoroshenko, I., Gorokhovatskyi, V., Latiff, L. A., & Ahmad, N. (2021). Development of effective methods for structural image recognition using the principles of data granulation and apparatus of fuzzy logic. IEEE Access, 9, 13417-13428.

40. Кобилін, О. А., & Творошенко, І. С. (2021). Методи цифрової обробки зображень.

41. Мєшков, Д., & Вечірська, І. (2024). Використання згорткових нейронних мереж при створенні deepfake. *Матеріали конференцій МНЛ*, (8 листопада 2024 р., м. Вінниця), 262-263.