МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЧЕРКАСЬКИЙ ДЕРЖАВНИЙ ФАХОВИЙ БІЗНЕС-КОЛЕДЖ
Циклова комісія (кафедра) комп’ютерної інженерії та інформаційних технологій

КВАЛІФІКАЦІЙНА РОБОТА

на тему

ТЕХНОЛОГІЇ ПРОГРАМУВАННЯ СИСТЕМ ДЛЯ СТВОРЕННЯ DEEPFAKE-ІВ

Виконав: студент групи 3П-22
Спеціальності
121 «Інженерія програмного забезпечення»
Денис МАНЖУЛА
Керівник:
Станіслав МАРЧЕНКО

Черкаси 2025

ЧЕРКАСЬКИЙ ДЕРЖАВНИЙ БІЗНЕС-КОЛЕДЖ

Кафедра комп'ютерної інженерії та інформаційних технологій

Спеціальність 121 «Інженерія програмного забезпечення»

Освітня програма Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ

Завідувач кафедри КІ та ІТ

_____ Владислав ХОТУНОВ
(підпис)

«_____»_____2024 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Манжулі Денису Олександровичу
(прізвище, ім'я, по батькові студента)

1. Тема випускної роботи Технології програмування систем для створення deepfake-ів
Керівник роботи Марченко Станіслав Віталійович, спеціаліст І категорії
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом закладу вищої освіти від «07» жовтня 2024 року № 68у.
2. Строк подання студентом випускної роботи 03.06.2025
3. Вихідні дані до випускної роботи мова програмування Мова програмування Python, фреймворк глибинного навчання PyTorch, допоміжні бібліотеки для обробки зображень OpenCV, NumPy, бібліотека для сегментації облич BiSeNet, інструменти для обробки відео FFmpeg.
4. Зміст випускної роботи (перелік питань, які потрібно розробити) огляд предметної області (базові уявлення про технологію deepfake, основні технології програмування систем для створення deepfake, принцип роботи генеративних автоенкодерів у deepfake-системах, огляд існуючих програмних засобів для створення deepfake, постановка задачі на розробку), проєктування та реалізація програмної системи (специфікація вимог до програмного забезпечення, архітектура програмного забезпечення, детальний опис програмного продукту), тестування програмної системи (мета та підхід до тестування, результати тестування моделі, тестування кінцевого результату).
5. Дата видачі завдання 15.09.2024р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Терміни виконання етапів	Примітка про виконання з підписами керівника і студента
1	Вступ	14.10.2024	
2	Розділ 1. Огляд предметної області	9.12.2024	
3	Розділ 2. Проєктування та реалізація програмної системи	10.03.2025	
4	Розділ 3. Тестування програмної системи	28.04.2025	
5	Висновки	12.05.2025	
6	Оформлення кваліфікаційної роботи (чистовий варіант)	26.05.2025	
7	Перевірка кваліфікаційної роботи на наявність ознак плагіату (за 10 днів до захисту)	02.06.2025	
8	Подання кваліфікаційної роботи на затвердження завідувачу кафедри (за 7 днів до захисту)	10.06.2025	

Студент _____ Денис МАНЖУЛА
(підпис)

Керівник роботи _____ Станіслав МАРЧЕНКО
(підпис)

АНОТАЦІЯ

Кваліфікаційна робота присвячена дослідженню сучасних підходів до створення deepfake-зображень і відео із застосуванням технологій глибинного навчання. Основна мета роботи – реалізація повноцінної системи генерації deepfake-контенту на базі фреймворку PyTorch із використанням автоенкодерної архітектури.

У межах проєкту було розглянуто поняття deepfake, типи моделей, що використовуються у цій сфері, а також здійснено огляд актуальних програмних засобів. Було реалізовано конвеєр обробки: починаючи з виділення обличчя на відео, нормалізації та створення навчального датасету, до тренування моделі та генерації відео зі зміненим обличчям.

Система включає в себе модулі для детекції облич, вирівнювання, маскування та реконструкції, а також скрипти для навчання моделі та автоматизованої генерації результатів. Тестування проводилося на власному датасеті та продемонструвало базову здатність моделі до генерації штучних облич з прийнятною візуальною якістю.

Результати роботи можуть бути використані як основа для подальших досліджень у сфері генеративного штучного інтелекту, а також при вивченні методів розпізнавання та протидії deepfake-технологіям.

Ключові слова**:** DEEPFAKE, АВТОЕНКОДЕР, PYTORCH, ГЛИБИННЕ НАВЧАННЯ, ГЕНЕРАЦІЯ ОБЛИЧ, МАШИННЕ НАВЧАННЯ.

ABSTRACT

The qualification thesis is dedicated to the study of modern approaches to generating deepfake images and videos using deep learning technologies. The main objective of the work is to implement a complete deepfake generation system based on the PyTorch framework using an autoencoder architecture.

Within the project, the concept of deepfakes was examined, including the types of models used in this domain, as well as a review of existing software tools. A full processing pipeline was implemented—from face detection and alignment, normalization, and dataset creation to model training and face-swapped video generation.

The system includes modules for face detection, alignment, masking, and reconstruction, as well as scripts for model training and automated result generation. Testing was conducted on a custom dataset and demonstrated the model's basic ability to generate artificial faces with visually acceptable quality.

The results of this work can serve as a foundation for further research in the field of generative artificial intelligence, as well as for the development of deepfake detection and countermeasure techniques.

Keywords: DEEPFAKE, AUTOENCODER, PYTORCH, DEEP LEARNING, FACE GENERATION, MACHINE LEARNING.

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ

AI – Artificial Intelligence (штучний інтелект)

API – Application Programming Interface (інтерфейс програмування додатків)

AE – AutoEncoder (автоенкодер)

CNN – Convolutional Neural Networks (згорткові нейронні мережі)

CUDA – Compute Unified Device Architecture (архітектура для обчислень на GPU)

cuDNN – CUDA Deep Neural Network library (бібліотека для глибоких нейронних мереж)

DCGAN – Deep Convolutional Generative Adversarial Network (глибока згорткова генеративна змагальна мережа)

GAN – Generative Adversarial Networks (генеративні змагальні мережі)

GPU – Graphics Processing Unit (графічний процесор)

JAX – Just After eXecution (фреймворк для машинного навчання)

LSTM – Long Short-Term Memory (довгострокова короткочасна пам’ять)

MLOps – Machine Learning Operations (операції машинного навчання)

TPU – Tensor Processing Unit (тензорний процесор)

VAE – Variational AutoEncoder (варіаційний автоенкодер)

BigGAN – Big Generative Adversarial Network (велика генеративна змагальна мережа)

CycleGAN – Cycle-Consistent Adversarial Networks (циклічно-консистентні змагальні мережі)

FOMM – First Order Motion Model (модель руху першого порядку)

GRU – Gated Recurrent Unit (керована рекурентна одиниця)

SAEHD – Sparse AutoEncoder HD (розріджений автоенкодер високої роздільності)

StyleGAN – Style-Based Generator Architecture (генератор на основі стилю)

AWS – Amazon Web Services (хмарні сервіси Amazon)

FFmpeg – Fast Forward Moving Picture Experts Group (мультимедійний фреймворк)

OpenVINO – Open Visual Inference and Neural Network Optimization (оптимізація нейронних мереж)

PIL – Python Imaging Library (бібліотека для обробки зображень)

BN – Batch Normalization (пакетна нормалізація)

FID – Fréchet Inception Distance (відстань Фреше-Інсепшн)

GAP – Global Average Pooling (глобальне усереднене об’єднання)

GMP – Global Max Pooling (глобальне максимальне об’єднання)

IN – Instance Normalization (нормалізація екземпляру)

KL – Kullback-Leibler divergence (дивергенція Кульбака-Лейблера)

LPIPS – Learned Perceptual Image Patch Similarity (навчена перцептивна схожість патчів зображень)

MLP – Multi-Layer Perceptron (багатошаровий персептрон)

SSIM – Structural Similarity Index (індекс структурної схожості)

WGAN-GP – Wasserstein GAN with Gradient Penalty (Вассерштейнова GAN з градієнтним покаранням)

AVI – Audio Video Interleave (формат відеофайлу)

CPU – Central Processing Unit (центральний процесор)

HD – High Definition (висока роздільність)

ЗМІСТ

ВСТУП

Упродовж останніх років технології генеративного штучного інтелекту демонструють стрімкий розвиток, відкриваючи нові можливості для створення високоякісного синтетичного медіаконтенту. Одним із найпомітніших напрямів цього розвитку стали Deepfake-технології, які дозволяють автоматизовано змінювати обличчя, міміку, голос або інші параметри зображення й відео за допомогою моделей глибокого навчання. З одного боку, Deepfake-технології відкривають широкі перспективи для творчої індустрії: кіновиробництва, віртуальних шоу, відеоігор, цифрового маркетингу та освіти. Вони дають змогу відтворювати історичні персонажі, здійснювати «омолодження» акторів, створювати персоналізований контент або збагачувати досвід взаємодії користувачів із цифровими продуктами. З іншого боку, популярність Deepfake-ів викликає занепокоєння в суспільстві через потенційні ризики їхнього зловживання, а саме: створення дезінформаційних матеріалів, компрометуючих відео, шахрайських схем. Поширення таких технологій вимагає від фахівців з інформаційної безпеки, розробників програмного забезпечення та дослідників глибокого розуміння принципів їх роботи, можливих вразливостей та методів виявлення.

Об’єктом дослідження є програмні системи для створення та обробки синтетичного медіаконтенту за допомогою технологій глибокого навчання.

Предметом дослідження виступають сучасні технології програмування, алгоритми машинного навчання та програмні інструменти для побудови систем створення Deepfake-ів, включаючи етапи обробки вхідних даних, навчання нейронних мереж та генерації результатів.

Метою цієї роботи є аналіз існуючих підходів, реалізація експериментального програмного прототипу системи для створення deepfake-ів на основі autoencoder-архітектури та оцінка отриманих результатів.

Для досягнення поставленої мети у роботі було визначено такі основні завдання:

1) провести аналіз існуючих підходів та архітектур, що застосовуються для створення Deepfake-ів;

2) дослідити програмні засоби, бібліотеки та інструменти для реалізації deepfake-систем;

3) спроєктувати та реалізувати прототип системи на основі технологій глибокого навчання;

4) провести експериментальні дослідження та оцінити якість отриманих результатів;

5) визначити перспективи подальшого розвитку та вдосконалення подібних систем.

РОЗДІЛ 1
ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Базові уявлення про технологію deepfake

Deepfake є комплексним поняттям, що поєднує слова «deep learning» (глибоке навчання) та «fake» (підробка). Ця технологія представляє собою метод синтезу медіаконтенту, заснований на використанні генеративних нейронних мереж глибокого навчання для створення реалістичних, але штучно згенерованих зображень, відео- або аудіозаписів.

Термін «deepfake» вперше з'явився в 2017 році завдяки користувачу Reddit під псевдонімом «deepfakes», який опублікував програмне забезпечення з відкритим первинним кодом для створення фальшивих відеороликів. З того часу технологія стрімко розвивалася, привернувши увагу як дослідників, так і широкої громадськості через свої можливості та потенційні ризики.

Основою технології deepfake є генеративні змагальні мережі (Generative Adversarial Networks, GANs), запропоновані Іаном Гудфеллоу у 2014 році. GAN-и складаються з двох нейронних мереж: генератора та дискримінатора, які навчаються в змагальному процесі. Генератор намагається створити реалістичні зображення, а дискримінатор намагається відрізнити справжні зображення від штучно згенерованих.

Технологія deepfake базується на принципах глибокого навчання, зокрема на використанні згорткових нейронних мереж (Convolutional Neural Networks, CNNs) та автоенкодерів. Автоенкодери складаються з двох частин: енкодера, який стискає вхідні дані до низькорозмірного представлення (латентного простору), та декодера, який відновлює оригінальні дані з цього представлення [20], принцип роботи зображено на рисунку 1.1.

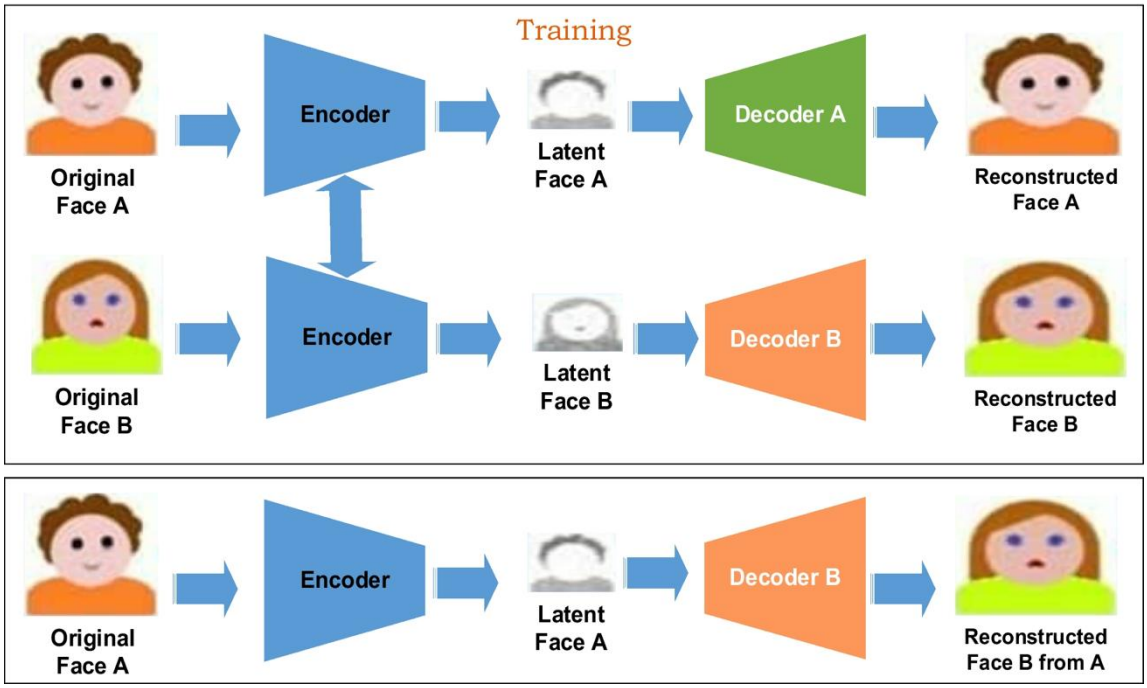


Рисунок 1.1 – Принцип роботи автоенкодеру [20]

Процес створення deepfake-зображень та відео включає кілька ключових етапів [21]:

1. Збір та підготовка даних. Необхідно зібрати великий обсяг тренувальних даних, включаючи фотографії або відео цільової особи та особи-джерела. Якість та кількість даних критично впливають на результат.

2. Попередня обробка. Дані проходять етап очищення, включаючи виявлення облич, їх вирівнювання, нормалізацію освітлення та приведення до стандартного розміру.

3. Навчання моделі. Нейронна мережа навчається на підготовлених даних, вивчаючи особливості обох облич та способи їх перетворення.

4. Генерація. Навчена модель використовується для створення нового контенту, де обличчя з вихідного відео замінюється на цільове обличчя.

5. Постобробка. Згенерований контент може потребувати додаткового налаштування для підвищення реалістичності.

Фотореалістичність є однією з ключових характеристик якісної підміни обличчя в медіаконтенті, оскільки згенероване зображення повинно виглядати природно. Не менш важливою стає темпоральна когерентність: у відео

послідовні кадри повинні бути узгоджені між собою. Крім того, важливі характеристики цільової особи повинні зберігатися і система повинна коректно обробляти різні умови освітлення.

Рисунок 1.2 – Приклад якісного deepfake-зображення [29]

Сучасні deepfake технології можна класифікувати за кількома критеріями [6]. За типом медіаконтенту розрізняють заміну обличчя (face swap), перенесення мімічних рухів (face reenactment), синтез мовлення та управління всім тілом (full body puppetry). За способом навчання можуть застосовуватись технології як на основі навчання з учителем, так і технології навчання без учителя або частково контрольоване навчання (semi-supervised learning).

Технологія deepfake знаходить застосування у різних галузях. Позитивні застосування вона має в кіноіндустрії (цифрове відтворення акторів), освіті (створення віртуальних викладачів), медицині (реконструкція облич) та історичних дослідженнях (відтворення історичних персонажів). У той же час, технологія має й проблематичні застосування через дезінформацію та фальшиві новини, шахрайство та кіберзлочинність, політичні маніпуляції та порнографічний контент без згоди.

Розвиток deepfake-технологій також стимулював створення методів детекції фальшивого контенту. Дослідники розробляють різноманітні підходи

для виявлення deepfake, включаючи аналіз артефактів стиснення, детекцію неконсистентностей у фізіологічних сигналах та використання спеціалізованих нейронних мереж для класифікації [21]. Етичні аспекти використання deepfake технологій є предметом активних дискусій у науковому співтоваристві та суспільстві. Багато країн розглядають або вже прийняли законодавство, що регулює використання таких технологій [8].

1.2 Основні технології програмування систем для створення deepfake

Створення систем для генерації deepfake потребує не лише глибокого розуміння сучасних технологій машинного навчання, а й практичного володіння відповідними інструментами програмування. Технологічний стек таких систем включає фреймворки глибокого навчання, бібліотеки комп'ютерного зору, інструменти для роботи з медіаконтентом та спеціалізовані архітектури нейронних мереж.

Фреймворк TensorFlow залишається одним з найбільш затребуваних фреймворків у галузі розробки deepfake-систем. Розроблений командою Google, він надає розробникам потужний інструментарій для створення та навчання складних нейронних мереж. Особливо привабливими є можливості розподіленого навчання на кількох GPU або TPU, що критично важливо для роботи з великими моделями. Крім того, TensorFlow пропонує широкий спектр готових шарів та операцій, що значно прискорює процес розроблення [6]. Для мобільних додатків доступний TensorFlow Lite, а для вебзастосувань – TensorFlow.js, що робить фреймворк універсальним рішенням для різних платформ.

PyTorch, створений командою Facebook, завоював особливу популярність серед дослідників завдяки своїй гнучкості та динамічним обчислювальним графам [6]. На відміну від статичних графів, динамічні дозволяють змінювати архітектуру мережі під час виконання, що особливо корисно при експериментах з новими підходами. Інтуїтивний інтерфейс програмування робить PyTorch

привабливим для новачків, а потужна екосистема, включаючи torchvision та torchaudio, забезпечує всі необхідні інструменти для роботи з мультимедійним контентом.

Keras, який тепер інтегрований у TensorFlow як високорівневий API, значно спрощує процес створення нейронних мереж. Його простий та інтуїтивний інтерфейс робить його особливо корисним для швидкого прототипування deepfake-моделей, дозволяючи розробникам зосередитися на логіці моделі, а не на технічних деталях реалізації.

Однією з най популярніших бібліотек комп'ютерного зору є OpenCV. Це справжній фундамент для роботи з зображеннями та відео у deepfake-системах, він надає весь необхідний інструментарій для детекції та трекінгу облич, виконання геометричних перетворень зображень, фільтрації та покращення якості контенту. Особливо важливими є можливості роботи з відеопотоками та калібрування камер, що дозволяє створювати системи реального часу.

Бібліотека dlib спеціалізується на більш специфічних задачах комп'ютерного зору та машинного навчання. Вона має особливу цінність для детекції «Facial Landmarks» приклад на рисунку 1.3, що є критично важливим для точного накладання deepfake-ефектів. Крім того, dlib надає потужні інструменти для розпізнавання облич, трекінгу об'єктів та вирівнювання зображень.

MediaPipe від Google революціонізувала роботу з медіаконтентом, надаючи готові рішення для багатьох типових задач. Особливо вражають можливості детекції та трекінгу облич у реальному часі, сегментації зображень та розпізнавання жестів і поз. Оптимізація для мобільних пристроїв робить MediaPipe ідеальним вибором для створення мобільних додатків з deepfake-функціональністю.

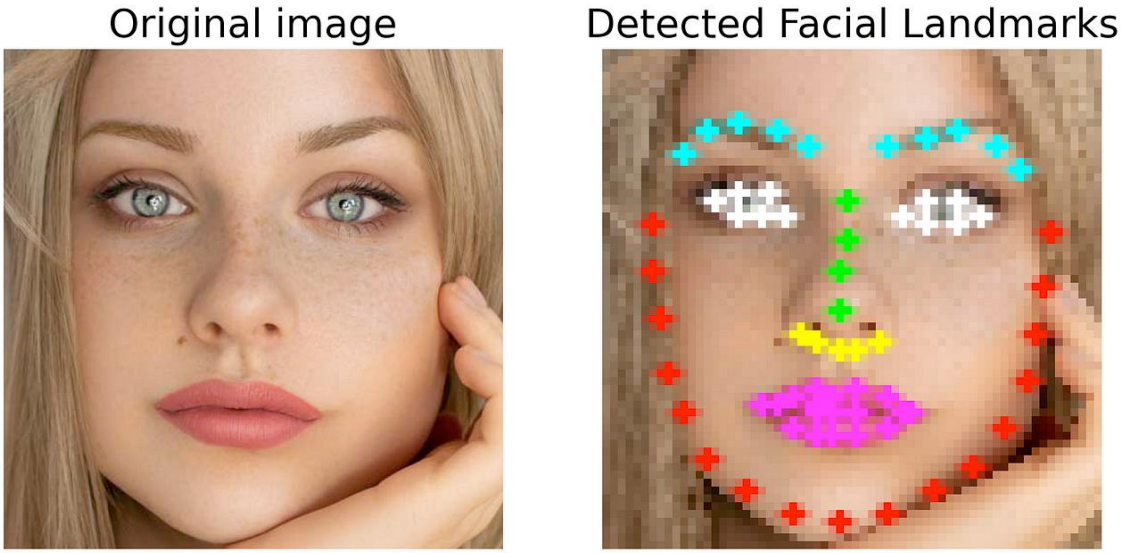


Рисунок 1.3 – Приклад детекції «Facial Landmarks» [30]

Вибір архітектур нейронних мереж для deepfake переважно невеликий. Однією з опцій є автоенкодери, які лежать в основі багатьох deepfake-систем і складаються з трьох основних компонентів. Енкодер відповідає за перетворення вхідного зображення у стиснуте латентне представлення, що містить найважливіші особливості оригінального контенту. Латентний простір служить своєрідним «місцем зустрічі» різних облич, де відбувається трансформація. Декодер потім відновлює зображення з цього латентного представлення, але вже з новими характеристиками цільового обличчя. Варіаційні автоенкодери додають елемент стохастичності до цього процесу, що покращує різноманітність та реалістичність згенерованих зображень.

Альтернативою є генеративні змагальні мережі (GAN), які революціонізували область генерації зображень, створивши принципово новий підхід до навчання. Ідея полягає в змаганні між генератором, який намагається створити максимально реалістичні фальшиві зображення, та дискримінатором, який намагається відрізнити справжні зображення від штучних, виявилася надзвичайно ефективною. У процесі такого змагального навчання обидві мережі постійно покращуються, досягаючи вражаючих результатів.

Серед популярних GAN-архітектур для deepfake особливе місце займає DCGAN, який використовує згорткові шари для покращення якості генерації [3].

Основна інновація DCGAN полягає у використанні операцій згортки з кроком (stride convolutions) замість субдискретизації за максимумом (max pooling) для зменшення розміру карт ознак (feature maps). Це дозволяє мережі самостійно навчитися оптимальним способам зменшення та збільшення дискретизації, що значно покращує якість генерованих зображень. Архітектура також широко використовує пакетну нормалізацію (batch normalization) у всіх шарах, окрім вихідного шару генератора та вхідного шару дискримінатора, що стабілізує процес навчання та запобігає внутрішньому зміщенню коваріат (internal covariate shift).

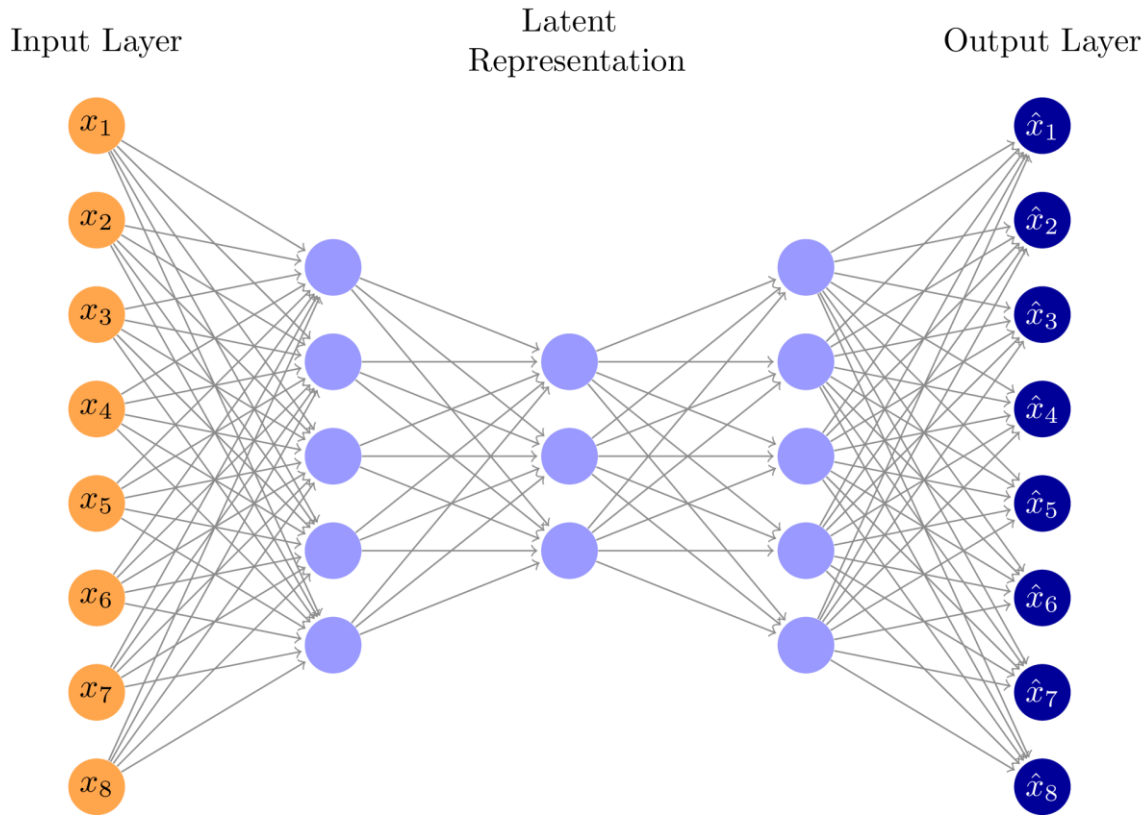


Рисунок 1.4 – Схематичне зображення архітектури автоенкодера

StyleGAN дозволяє точно контролювати стиль та деталі генерації, що особливо важливо для створення реалістичних портретів [3]. Центральною інновацією StyleGAN є відображальна мережа (mapping network), яка перетворює традиційний прихований код (latent code) в проміжний прихований простір (intermediate latent space). Цей проміжний простір має значно кращі властивості для інтерполяції та редагування, оскільки усуває зав’язаність між різними

атрибутами. Замість того, щоб подавати шум безпосередньо в генератор, StyleGAN спочатку обробляє його через повнозв'язну мережу, створюючи більш придатні для маніпуляцій представлення.

CycleGAN вирішує задачу непарного перетворення доменів, дозволяючи навчатися без паралельних наборів даних [3]. CycleGAN вирішує фундаментальну проблему машинного навчання, яка полягає в необхідності парних даних для навчання моделей перетворення зображень. Традиційні підходи вимагали точно відповідних пар зображень «до» та «після» трансформації, що часто неможливо отримати в реальних умовах.

Архітектурна концепція CycleGAN базується на використанні двох генераторів: G для перетворення з домену X у домен Y, та F для зворотного перетворення з Y в X. Кожен генератор має відповідний дискримінатор: D_x для розрізнення реальних та згенерованих зображень домену X, і D_y для домену Y. Така симетрична архітектура забезпечує двонаправлене навчання без потреби в парних даних.

BigGAN зосереджується на високоякісній генерації великих зображень з високою деталізацією [3]. BigGAN спеціально розроблений для генерації великих, високоякісних зображень з надзвичайною деталізацією, встановлюючи нові стандарти якості в області генеративних моделей. Назва «Big» відображає як розмір моделі, так і розмір зображень, які вона може генерувати.

Механізми уваги значно покращують якість deepfake-контенту за рахунок можливості фокусування на найважливіших частинах зображення. Це дозволяє зберігати критичні деталі при трансформації та покращує темпоральну консистентність у відеороликах, що робить результат більш переконливим.

Для практичної реалізації deepfake-систем добре зарекомендували себе спеціалізовані архітектури згорткових нейромереж. Архітектура U-Net стала в цьому контексті стандартом де-факто для задач сегментації та трансформації зображень. Її унікальна енкодер-декодер структура зі skip-зв'язками дозволяє ефективно зберігати деталі вихідного зображення, що критично важливо для створення реалістичних результатів. Ця архітектура особливо ефективна для

pixel-to-pixel перетворень, коли потрібно змінити кожен піксель зображення відповідно до заданої трансформації [23].

Залишкові нейронні мережі, на зразок ResNet, вирішили фундаментальну проблему затухаючих градієнтів, що дозволило створювати значно глибші нейронні мережі. Залишкові зв'язки дають можливість навчати мережі з сотнями прошарків, що критично важливо для складних задач обробки зображень. Часто ResNet використовуються як backbone-мережа для інших, більш спеціалізованих архітектур [7].

MobileNet-архітектури були спеціально оптимізовані для мобільних та вбудованих систем. Використання відокремлюваних згорток з урахуванням глибини (depthwise separable convolutions) дозволяє значно зменшити кількість параметрів без суттєвої втрати якості, що робить можливою обробку в реальному часі навіть на обмежених ресурсах мобільних пристроїв.

Іншими технологіями оптимізації та прискорення обчислень є CUDA та cuDNN. Для цього вони задіюють апаратні можливості графічних процесорів компанії NVIDIA. Ці технології дозволяють виконувати паралельні обчислення та використовувати оптимізовані операції для нейронних мереж, що призводить до значного прискорення як процесу навчання, так і інференсу готових моделей. Додатково фреймворк TensorRT від NVIDIA спеціально розроблений для оптимізації моделей у продакшн-середовищі. Він дозволяє виконувати квантизацію моделей, об'єднувати операції для підвищення ефективності та динамічно формувати батчі залежно від навантаження, що критично важливо для реальних додатків.

OpenVINO від Intel забезпечує аналогічну оптимізацію, але зосереджується на процесорах Intel. Кросплатформенна підтримка та оптимізація для CPU та інтегрованих GPU робить його привабливим вибором для систем, де використання дискретних графічних карт неможливе або недоцільне.

У ході навчання моделей для deepfake-систем також можуть використовуватись інструменти для роботи з даними: FFmpeg для роботи з медіафайлами, бібліотека Pillow для роботи з зображеннями, NumPy та SciPy

формують математичну основу. У різних сценаріях задіюються хмарні технології та сервіси, на зразок Google Colab чи професійних хмарних платформ AWS SageMaker, Google Cloud AI Platform, Azure Machine Learning тощо. Допоміжними інструментами для розроблення та налагодження коду можуть стати Weights & Biases (моніторинг експериментів машинного навчання) [10,19], TensorBoard (інтегрований інструмент візуалізації для TensorFlow) та ін.

Розроблення deepfake-систем неможливе без урахування аспектів безпеки та етики. Differential Privacy [6] допомагає захистити приватність тренувальних даних, гарантуючи, що модель не «запам’ятає» специфічні деталі окремих зображень. Методика Federated Learning дозволяє навчати моделі без централізації даних, що особливо важливо при роботі з чутливим контентом. Змагальне навчання (Adversarial Training) підвищує стійкість моделей до спеціально створених атак, що критично важливо для захисту від зловмисного використання. Технології додавання водяних знаків дозволяють вбудовувати невидимі позначки, які допомагають ідентифікувати згенерований контент та відстежувати його походження.

Отже, сучасні deepfake-системи представляють собою складний симбіоз передових технологій машинного навчання, оптимізованих обчислювальних рішень та етичних практик розробки. Успішна реалізація таких систем вимагає не лише технічної експертизи, а й глибокого розуміння соціальних та етичних наслідків їх використання.

1.3 Принцип роботи генеративних автоенкодерів у deepfake-системах

Генеративні автоенкодери представляють собою фундаментальну архітектуру нейронних мереж, яка лежить в основі багатьох сучасних deepfake-систем. Ці мережі поєднують принципи стиснення інформації та генеративного моделювання, дозволяючи створювати реалістичні трансформації зображень облич.

Автоенкодер складається з двох основних компонентів: енкодера та декодера. Енкодер відображає вхідні дані x у латентний простір z, а декодер відновлює дані з латентного представлення. Математично це можна записати як енкодер $z = E(x)$ та декодер $x' = D(z)$. Мета навчання полягає в мінімізації функції втрат реконструкції (формула 1.1):

$$L_{rec} = \|x - D(E(x))\|^2. \quad (1.1)$$

У контексті deepfake систем, автоенкодери навчаються відображати зображення облич у спільний латентний простір, що дозволяє здійснювати трансформації між різними особами.

Варіаційні автоенкодери (VAE) [18] розширюють базову концепцію автоенкодерів, додаючи статистичну регуляризацію до латентного простору. Замість детермінованого відображення, VAE навчається параметрам розподілу в латентному просторі.

Енкодер у VAE виводить два вектори: μ (середнє значення) та σ (стандартне відхилення). Латентна змінна семплується з нормального розподілу $z \sim N(\mu, \sigma^2)$. Функція втрат VAE включає два компоненти (формула 1.2):

$$L_{VAE} = L_{rec} + \beta \cdot L_{KL}, \qquad L_{KL} = KL(q(z|x) \,||\, p(z)), \quad (1.2)$$

де L_{KL} – дивергенція Кульбака-Лейблера між апріорним та апостеріорним розподілами. Параметр β контролює баланс між якістю реконструкції та регуляризацією латентного простору.

У контексті задачі підміни облич (face swap) використовуються спеціалізовані архітектури. Сіамські (siamese) автоенкодери використовуються для навчання спільного представлення двох різних облич [28]. Архітектура включає спільний енкодер E, який навчається виділяти загальні риси облич; два декодери D_A та D_B для кожної цільової особи; спільний латентний простір для обох облич. Процес навчання математично записується так. Обробка зображення

особи A: $x_A \rightarrow E(x_A) \rightarrow D_A\big(E(x_A)\big) \rightarrow x'_A$; зображення особи B – $x_B \rightarrow E(x_B) \rightarrow D_B\big(E(x_B)\big) \rightarrow x'_B$; для генерації deepfake-зображення особи A обробляється в послідовності $x_A \rightarrow E(x_A) \rightarrow D_B\big(E(x_A)\big) \rightarrow x'_B$.

Сучасні deepfake-системи інтегрують механізми уваги (attention) для покращення якості генерації. Самоувага (self-attention) дозволяє мережі фокусуватися на різних частинах вхідного зображення:

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V, \quad (1.3)$$

де Q, K, V – матриці запитів, ключів та значень відповідно. Просторова увага (spatial attention) концентрується на просторових областях, визначаючи які піксельні локації є найбільш важливими:

$$A_{spatial} = sigmoid\Big(Conv\Big(concat\big(AvgPool(F), MaxPool(F)\big)\Big)\Big), \quad (1.4)$$

де $AvgPool(F)$, $MaxPool(F)$ – операції середнього та максимального пулінгу відповідно, $concat$ – операція конкатенації, $Conv$ – згорткова операція.

Канальна увага (channel attention) виділяє важливі канали ознак:

$$A_{channel} = sigmoid\Big(MLP\big(GAP(F)\big) + MLP\big(GMP(F)\big)\Big), \quad (1.5)$$

де $GAP(F)$, $GMP(F)$ – операції глобального середнього та максимального пулінгу відповідно, MLP – багатошаровий перцептрон. Канальна увага (channel attention) визначає, які типи візуальної інформації є найбільш релевантними для поточного завдання генерації.

Архітектура U-Net для задачі підміни облич (face swap) зображена на рис. 1.5. На етапі роботи енкодер (contracting path) вхідне зображення обробляється низкою згорткових блоків для зменшення просторової розмірності та збільшення кількості каналів. Bottleneck-блок є найглибшим рівнем мережі та

відповідає за максимальне стиснення інформації. Декодер (expansive path) містить деконволюційні прошарки для відновлення розмірності, skip-з'єднання з енкодера для збереження деталей зображення.

На рис. 1.6 показано базову архітектуру генеративної змагальної мережі (GAN) [24]. Поєднання автоенкодерів з GAN-архітектурою створює потужні генеративні змагальні автоенкодери (generative adversarial autoencoders) [11]. Така архітектура включає автоенкодер (генератор), дискримінатор зображень та дискримінатор латентного простору. Функція втрат у такому разі набуватиме вигляду

$$L_{total} = L_{rec} + \lambda_{adv} \cdot L_{adv} + \lambda_{latent} \cdot L_{latent}, \quad (1.6)$$

де L_{rec} – втрати реконструкції, L_{adv} – змагальні втрати для згенерованих зображень, L_{latent} – втрати для латентного простору.

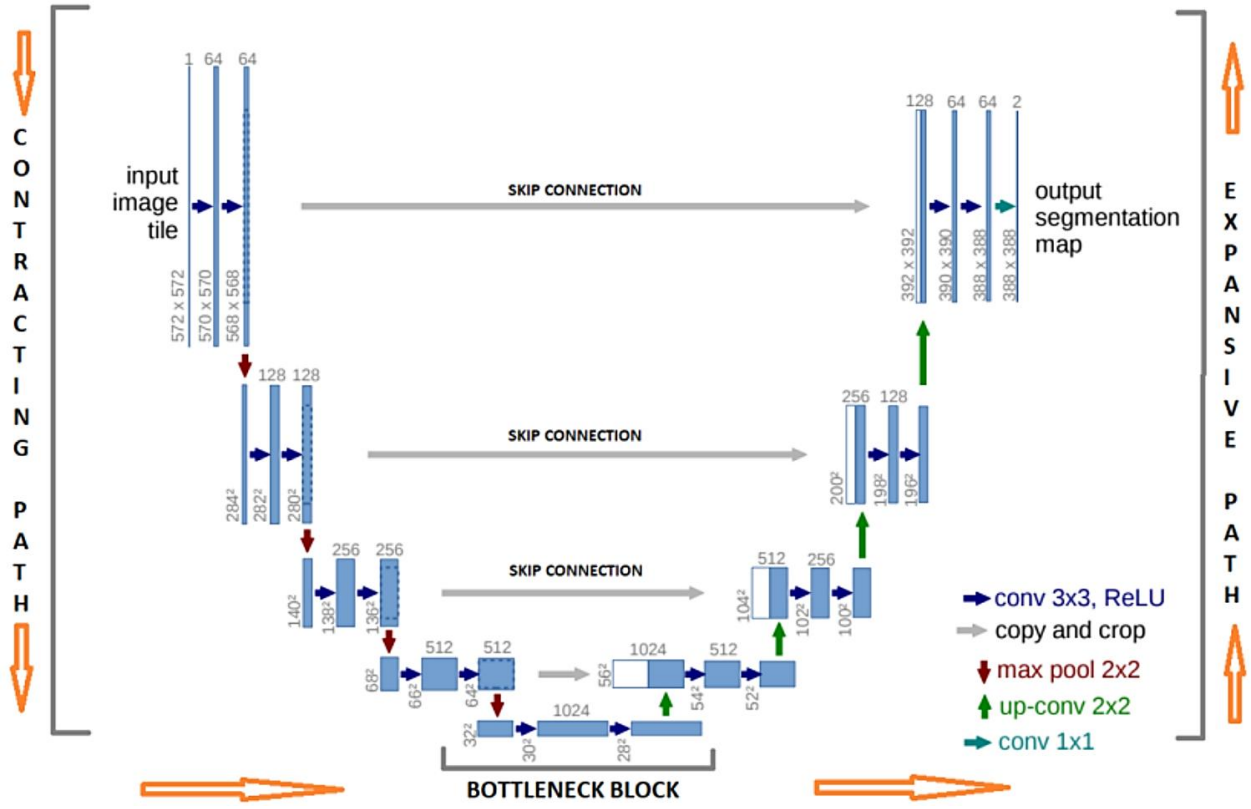


Рисунок 1.5 – Архітектура U-Net для задачі підміни облич [29]:

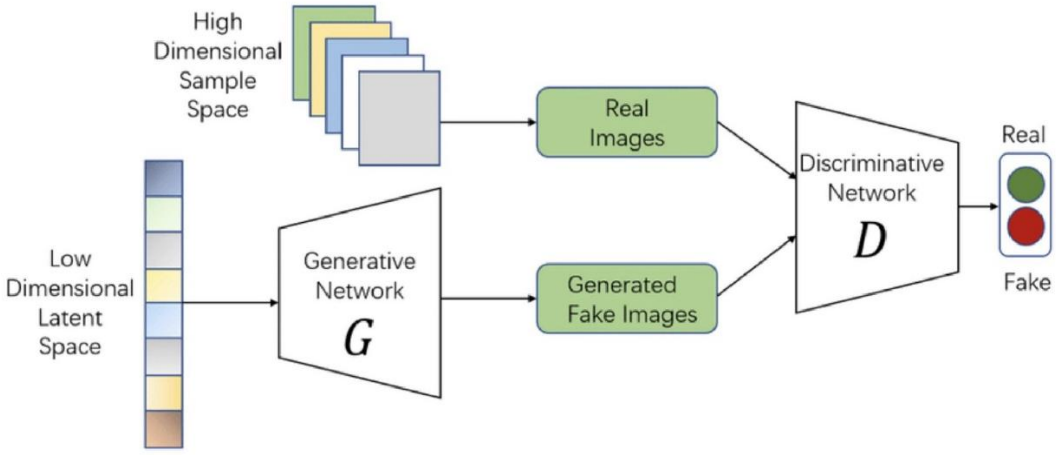


Рисунок 1.6 – Генеративна змагальна мережа

Іншим варіантом є умовні автоенкодери [18], які дозволяють контролювати процес генерації через додаткову умовну інформацію (ідентичність, поза, вираз обличчя, освітлення). Для deepfake-відео критично важливо забезпечити темпоральну консистентність, функція втрат для якої може бути записана так:

$$L_{temp} = \sum \left\| D\left(E(x_t)\right) - D\left(E\left(x_{\{t+1\}}\right)\right) \right\|^2, \quad (1.7)$$

де $E(x_t), E\left(x_{\{t+1\}}\right)$ – кодування кадрів у моменти часу t та t+1 відповідно, D – декодер. Ця функція втрат забезпечує темпоральну консистентність між послідовними кадрами відео, мінімізуючи різницю між декодованими представленнями сусідніх кадрів.

Іншими способами побудови deepfake-зображень можуть бути рекурентні автоенкодери (використовують LSTM/GRU для моделювання часових залежностей) [31] та мультидоменні автоенкодери, на зразок StarGAN-подібних архітектур [32], які дозволяють одночасну роботу з кількома доменами.

Метриками якості для автоенкодерів можуть слугувати [30]:

– структурна подібність (Structural Similarity Index, SSIM)

$$SSIM(x, y) = \frac{\left(2\mu_x \mu_y + c_1\right)\left(2\sigma_{xy} + c_2\right)}{\left(\mu_x^2 + \mu_y^2 + c_1\right)\left(\sigma_x^2 + \sigma_y^2 + c_2\right)},$$

де μ_x, μ_y – середні значення для зображень x та y відповідно; σ_x, σ_y – стандартні відхилення, σ_{xy} – коваріація між зображеннями, c_1, c_2 – константи стабілізації. Звідси, SSIM враховує яскравість, контраст та структуру зображення.

– $LPIPS$-метрика (Learned Perceptual Image Patch Similarity):

$$LPIPS = \sum \|\varphi_i(x) - \varphi_i(y)\|^2,$$

де φ_i – активації і-го шару попередньо навченої мережі.

– FID-метрика (Fréchet Inception Distance) – оцінює якість генерації:

$$FID = \|\mu_1 - \mu_2\|^2 + Tr\left(C_1 + C_2 - 2\sqrt{C_1 \cdot C_2}\right)$$

де μ_1, μ_2 – середні векторів ознак реальних та згенерованих зображень відповідно, C_1, C_2 – коваріаційні матриці, Tr – слід матриці. FID порівнює розподіли ознак реальних та синтетичних зображень.

1.4 Огляд існуючих програмних засобів для створення deepfake

Сучасна екосистема програмних засобів для створення deepfake-ів включає широкий спектр інструментів – від простих мобільних додатків до професійних платформ для дослідників. Розуміння можливостей та обмежень різних інструментів є критично важливим для вибору оптимального рішення для конкретних задач.

DeepFaceLab є одним з найпопулярніших та найпотужніших фреймворків для створення deepfake[16]. Розроблений командою дослідників, цей інструмент надає повний pipeline для створення високоякісних deepfake відео. DeepFaceLab підтримує різні архітектури нейронних мереж, включаючи SAEHD (Sparse AutoEncoder HD), Quick96, та інші. Фреймворк забезпечує детальний контроль над процесом тренування, включаючи налаштування параметрів мережі, вибір функцій втрат та стратегій аугментації даних. Однією з ключових переваг DeepFaceLab є його здатність працювати з відео високої роздільної здатності та

забезпечувати стабільні результати навіть при складних умовах освітлення та ракурсах.

FaceSwap представляє собою відкритий інструмент, який фокусується на простоті використання та модульності архітектури [17]. Програма надає графічний інтерфейс користувача, що робить її доступною для широкого кола користувачів. FaceSwap включає вбудовані інструменти для попередньої обробки даних, тренування моделей та постобробки результатів. Архітектура системи дозволяє легко розширювати функціональність через систему плагінів, що робить її привабливою для дослідників та розробників.

First Order Motion Model (FOMM) представляє інноваційний підхід до генерації deepfake через використання ключових точок та теплових карт [13]. Ця система може переносити рухи з одного відео на статичне зображення, створюючи реалістичні анімації без необхідності в явному моделюванні 3D структури обличчя. FOMM демонструє високу ефективність при роботі з різними типами об’єктів, включаючи людські обличчя, тварин та неживі об’єкти. Порівняння характеристик описаних програмних продуктів наведено в таблиці 1.1.

Ці професійні фреймворки (DeepFaceLab, FaceSwap) забезпечують найвищу якість результатів та максимальний контроль над процесом створення deepfake-контенту. Однак вони вимагають значних технічних знань, потужного апаратного забезпечення та тривалого часу для навчання моделей. Ці інструменти оптимальні для дослідницьких проєктів, створення високоякісного контенту та випадків, коли потрібен повний контроль над процесом генерації.

Незважаючи на значний прогрес, сучасні програмні засоби для створення deepfake стикаються з рядом технічних та етичних викликів. Технічні обмеження включають складнощі з обробкою відео високої роздільної здатності, необхідність у великих обсягах навчальних даних, чутливість до умов освітлення та ракурсів, а також проблеми з темпоральною консистентністю в відео. Апаратні вимоги залишаються високими для створення якісних результатів, що обмежує доступність технології для широкого кола користувачів.

Таблиця 1.1 – Порівняльна таблиця програмних засобів для створення deepfake-ів

Критерій	DeepFaceLab	FaceSwap	First Order Motion
Тип платформи	Desktop	Desktop	Research
Складність використання	Висока	Середня	Висока
Якість результату	Відмінна	Добра	Добра
Швидкість обробки	Повільна	Середня	Швидка
Вимоги до обладнання	Високі	Середні	Середні
Підтримка GPU	Так	Так	Так
Формати відео	MP4, AVI	MP4, AVI	MP4
Максимальна роздільність	4K+	1080p	1080p
Batch обробка	Так	Так	Ні
Відкритий код	Так	Так	Так
Вартість	Безкоштовно	Безкоштовно	Безкоштовно
Підтримка realtime	Ні	Ні	Ні
API доступ	Ні	Ні	Ні
Навчальні ресурси	Багато	Багато	Середньо
Спільнота	Велика	Велика	Середня

Етичні питання стосуються потенційного зловживання технологією для створення неправдивого контенту, порушення приватності та прав особистості. Правові аспекти включають необхідність у розробці відповідних регулятивних рамок та механізмів контролю за використанням технології. Розуміння цих викликів та обмежень є критично важливим для відповідального розвитку та використання технологій deepfake у майбутньому.

1.5 Постановка задачі на розробку

У межах кваліфікаційної роботи ставиться задача проєктування та реалізації програмного прототипу системи для створення deepfake-відео з використанням генеративного автоенкодера. Передбачається виконання низки дій. Обрані технологічні компоненти зібрані в таблиці 1.2. На етапі підготовки даних слід здійснити виокремлення кадрів з відео у форматі PNG, виявляти та

вирівнювати обличчя, будувати маски обличчя для фокусування моделі на релевантних зонах (очі, ніс, рот).

Навчання моделі проходитиме на основі архітектури генеративного автоенкодера з одиним енкодер та двома декодерами (для облич A та B відповідно). Енкодер стискає інформацію про обличчя в латентне представлення, яке декодери розкодовують в залежності від цільового стилю.

У якості основного фреймворку обрано PyTorch через простоту побудови власних моделей, підтримку GPU та активну спільноту. Функцією втрат визначено MSELoss (середньоквадратичну помилка) на маскованих ділянках обличчя. Орієнтовні гіперпараметри для моделі: кількість ітерацій – до 100 000, розмір пакету (batch size) – 8-16, швидкість навчання (learning rate) – 5e-6.

Таблиця 1.2 – Обрані інструменти та бібліотеки

Інструмент / Бібліотека	**Призначення**
PyTorch	Реалізація моделі автоенкодера, навчання та інференс
face_alignment	Виявлення ключових точок обличчя, афінне вирівнювання
BiSeNet (інтегрований модуль)	Сегментація обличчя, побудова маски
OpenCV	Обробка відео та зображень, зчитування та запис
tqdm	Візуалізація прогресу навчання
argparse	CLI-аргументи для управління скриптами
numpy / torchvision	Обробка масивів, трансформації, нормалізація
json_tricks	Збереження ключових точок у форматі JSON з підтримкою NumPy

Deepfake-відео має генеруватись після того, як модель проходить висновування (inference) на кожному кадрі відео. Застосовується інвертоване афінне перетворення, щоб згенероване обличчя замістило оригінальне з урахуванням позиції та об’єднання згенерованих кадрів у відео з допомогою OpenCV (cv2.VideoWriter) або ffmpeg. Оцінка результатів здійснюватиметься за

критеріями візуального аналізу стабільності обличчя між кадрами, порівняння якості при різних датасетах.

РОЗДІЛ 2
ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ ПРОГРАМНОЇ СИСТЕМИ

2.1 Специфікація вимог до програмного забезпечення

Функціональні можливості розробленої системи мають охоплювати широкий спектр операцій з медіаданими та машинним навчанням. Система має забезпечувати завантаження та обробку відеофайлів у найпоширеніших сучасних форматах, таких як .mp4 та .avi, що гарантує сумісність з більшістю існуючих відеоджерел. Розбиття вихідного відео на окремі статичні фрейми здійснюється з можливістю точного налаштування частоти кадрів, що дозволяє оптимізувати якість обробки залежно від специфіки конкретного проєкту та доступних обчислювальних ресурсів.

Особливо важливим компонентом системи є алгоритм автоматичного виявлення та локалізації облич на кожному кадрі відео. Цей процес включає не лише детекцію присутності обличчя, а й точне визначення його контурів та орієнтації у просторі. Після успішної ідентифікації система виконує прецизійне вирізання областей з обличчями, застосовуючи складні математичні перетворення для забезпечення оптимального позиціонування та масштабування.

Наступний етап обробки передбачає комплексну нормалізацію отриманих зображень облич, що включає стандартизацію розмірів, корекцію освітлення та колірної гами, а також створення спеціалізованих масок для точного визначення меж обличчя. Всі оброблені дані систематично організовуються та зберігаються у структурованому вигляді для подальшого використання на етапі машинного навчання.

Центральним елементом системи є процес навчання автоенкодерної моделі, який здійснюється одночасно на двох незалежних датасетах, що представляють різних осіб. Цей підхід дозволяє мережі вивчити унікальні характеристики кожного обличчя та навчитися виконувати високоякісну

трансформацію між ними. Після завершення навчального процесу система генерує нові кадри з виконаною заміною облич, зберігаючи при цьому природність міміки та вираз обличчя.

Фінальний етап роботи системи полягає в збірці всіх оброблених кадрів у цілісне відео з накладеним deepfake-ефектом. Управління всіма описаними етапами здійснюватиметься через зручний інтерфейс командного рядка, що забезпечує гнучкість налаштувань та можливість автоматизації процесу. Результати кожного етапу обробки систематично зберігаються у відповідних директоріях з логічною структурою, що полегшує навігацію та подальше використання даних.

Щодо нефункціональних характеристик системи, програмне забезпечення орієнтоване для роботи в середовищі операційної системи Windows 10 або більш сучасних версій. Вибір цієї платформи обумовлений її широкою розповсюдженістю серед користувачів та наявністю необхідних драйверів для роботи з графічними прискорювачами. Розробка програмного продукту ведеться з використанням мови програмування Python версії 3.10, яка забезпечує оптимальний баланс між продуктивністю виконання та зручністю розробки.

Особлива увага приділена використанню бібліотеки PyTorch для реалізації алгоритмів машинного навчання та роботи з нейронними мережами. Система автоматично визначає наявність CUDA-сумісного графічного прискорювача та максимально використовує його обчислювальні можливості для значного прискорення процесу навчання та конвертації. Для всіх операцій з обробки зображень застосовується перевірена часом бібліотека OpenCV, яка забезпечує високу якість та швидкість обробки візуального контенту.

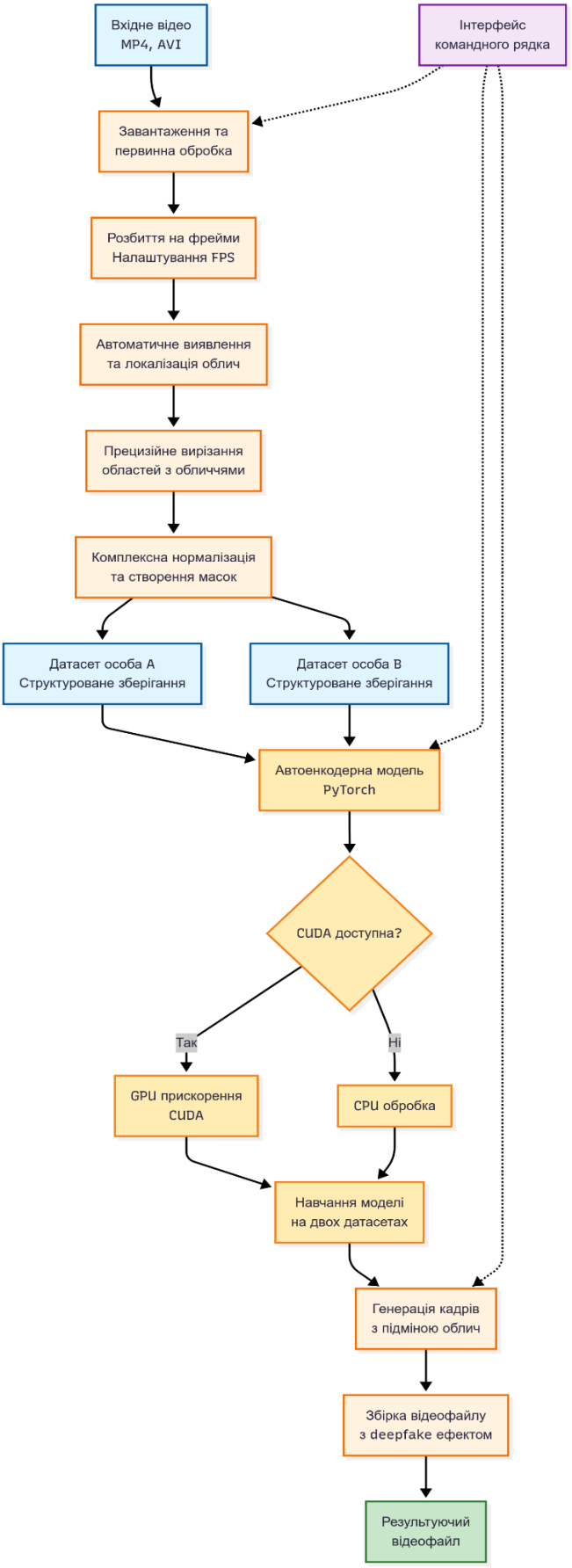


Рисунок 2.1 – Діаграма вимог до програмного забезпечення

2.2 Архітектура програмного забезпечення

Архітектурні рішення для подальшої програмної реалізації системи базуються на принципах модульності та розподілу відповідальності, що забезпечує високу гнучкість, масштабованість та можливість незалежного розвитку окремих компонентів. Кожен функціональний етап обробки даних реалізований у вигляді самостійного Python-модуля, що дозволяє легко вносити зміни, виправлення та покращення без ризику впливу на інші частини системи.

Взаємодія між різними модулями системи здійснюється через добре структуровану файлову систему, що забезпечує надійну передачу даних та можливість відстеження проміжних результатів обробки. Такий підхід також дозволяє легко інтегрувати додаткові інструменти аналізу та моніторингу процесу обробки (рис. 2.2). Фактично, отримано лінійну послідовність виконання компонентів.

Перший етап обробки реалізується модулем виявлення та обробки облич (face_detection.py), який виконує комплексну попередню обробку вхідних зображень. Цей модуль застосовує алгоритм SFD для детекції облич у зображеннях з налаштовуваним рівнем довіри, що за замовчуванням становить 0.9. Після виявлення облич система виконує їх вирівнювання на основі ключових точок обличчя для стандартизації положення та розміру. Особливо важливим є застосування моделі BiSeNet для створення масок обличчя, що виділяють релевантні області для подальшої обробки. Модуль також генерує JSON-файл з метаданими, що містить інформацію про матриці трансформації та координати ключових точок, необхідних для подальших етапів обробки.

Налаштування цього модуля включають параметр size для встановлення розміру вирівняних зображень облич, який за замовчуванням становить 256×256 пікселів. Параметр min_confidence визначає мінімальний рівень довіри для детекції облич, а min_size встановлює мінімальний відносний розмір обличчя в зображенні. Для оптимізації використання пам’яті передбачено параметр max_detection_size, який обмежує максимальний розмір зображення для обробки.

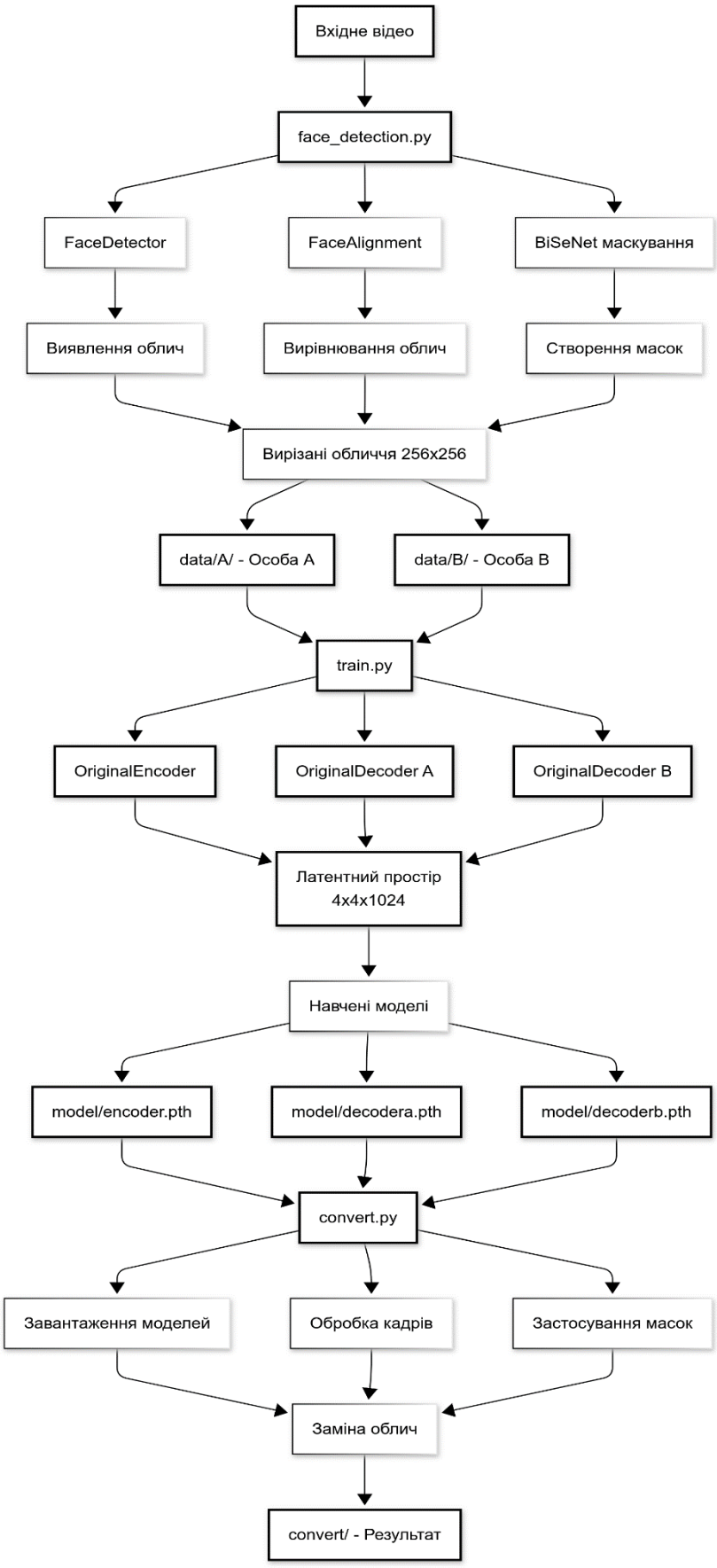


Рисунок 2.2 – Діаграма архітектури до програмного забезпечення

Центральним компонентом системи є модуль навчання (train.py), що реалізує процес навчання нейронної мережі на основі архітектури автоенкодера з двома декодерами. Спільний енкодер стискає зображення облич до латентного простору розмірності 4×4×1024, після чого два спеціалізованих декодери навчаються відтворювати обличчя конкретних осіб. Функція втрат базується на MSE з маскуванням для фокусування на релевантних областях обличчя, що значно покращує якість генерованих зображень.

Параметри навчання включають розмір пакета, який за замовчуванням становить 16 зображень, кількість ітерацій навчання з базовим значенням 100,000, швидкість навчання на рівні 1e-6 та частоту збереження моделі кожні 1000 ітерацій. Під час навчання система регулярно генерує прев'ю результатів, що дозволяє контролювати прогрес та якість навчання.

Фінальний етап реалізується модулем конвертації (convert.py), який виконує безпосередню заміну облич у цільових зображеннях. Модуль завантажує навчені параметри енкодера та обраного декодера, застосовує навчену модель до вирівняних облич, а потім виконує зворотну трансформацію для повернення оброблених облич до оригінального зображення з використанням збережених матриць трансформації. Компонування відбувається шляхом змішування оригінального та згенерованого зображень з використанням масок, що забезпечує природний та реалістичний результат.

Архітектура нейронної мережі визначається в модулі models.py та складається з трьох основних компонентів. OriginalEncoder представляє багатошаровий згортковий енкодер з чотирма згортковими шарами з фільтрами розмірів 128, 256, 512 та 1024, використовуючи LeakyReLU активацію з параметром 0.1 та два повнозв'язних шари для перетворення у латентний простір (рис. 2.3). OriginalDecoder містить послідовність модулів збільшення з трьома модулями Upscale для поступового збільшення розміру, вихідним згортковим шаром для генерації RGB-зображення та сигмоїдальною активаційною функцією для нормалізації значень пікселів. Спеціалізований модуль Upscale поєднує згортковий прошарок для збільшення кількості каналів з

PixelShuffle для реорганізації пікселів та збільшення просторового розміру (додаток А).

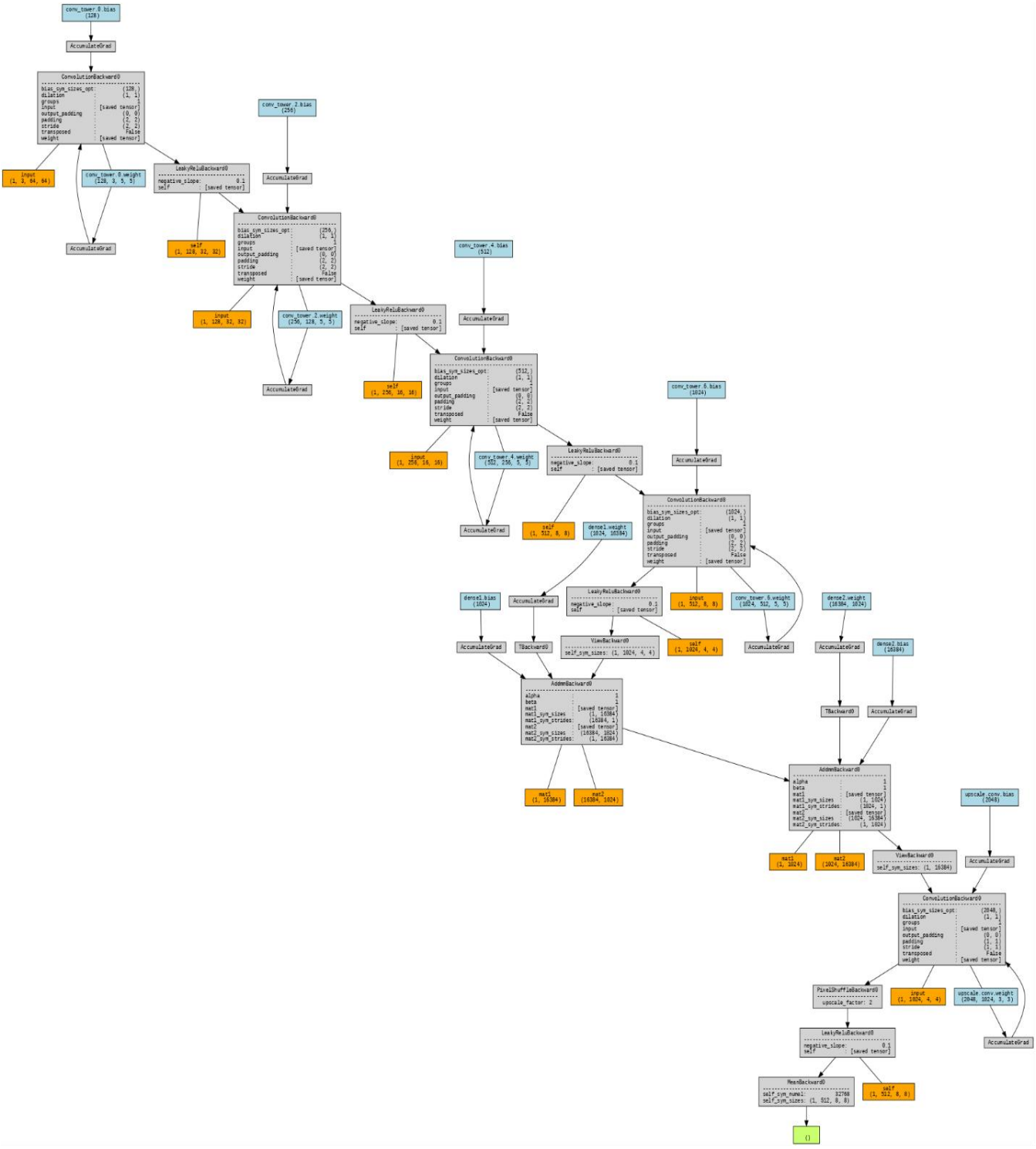


Рисунок 2.3 – Архітектура модуля енкодера

Організація файлової структури проєкту здійснена з урахуванням принципів логічного групування та зручності навігації. Центральна директорія

data містить спеціалізовані підпапки A та B, призначені для систематизованого зберігання фреймів, що представляють різних осіб у навчальному датасеті. Директорія model служить централізованим сховищем для збереження навчених параметрів енкодера та декодерів після завершення процесу навчання. Вихідна директорія convert призначена для зберігання результатів обробки у вигляді зображень після виконання заміни облич.

Операційна модель системи передбачає запуск кожної з основних фаз обробки, навчання та конвертації як незалежних команд з індивідуально налаштованими параметрами та аргументами, що забезпечує максимальну гнучкість використання та можливість тонкого налаштування процесу під конкретні потреби користувача.

2.3 Детальний опис програмного продукту

Модуль face_detection.py представляє собою комплексну систему для роботи з розпізнавання та обробки облич. Він відповідає за критично важливі операції автоматичного виявлення облич на кожному кадрі відео, використовуючи сучасні алгоритми комп'ютерного зору. Лістинг 2.1 демонструє базовий код запуску додатка.

Лістинг 2.1 – Основна функція виявлення облич

```
def main(opt):
    if not os.path.exists(opt.export_path):
        os.mkdir(opt.export_path)

    device = "cuda" if torch.cuda.is_available() and not opt.cpu else "cpu"

    face_detector = FaceDetector(device=device, verbose=False)
    face_aligner   =   FaceAlignment(LandmarksType.TWO_D,   device=device,
verbose=False)

    masker = BiSeNet(n_classes=19)
    if device == "cuda":
        masker.cuda()
    model_path = os.path.join(".", "binaries", "BiSeNet.pth")
    masker.load_state_dict(torch.load(model_path,
map_location=torch.device(device)))
    masker.eval()
    desired_segments = [1, 2, 3, 4, 5, 6, 10, 11, 12, 13]
```

Ця функція починає з перевірки, чи існує папка для експорту результатів. Якщо ні, то створює її. Далі визначається, чи доступна відеокарта (CUDA), і в залежності від цього вибирається пристрій для обчислень. Створюється детектор облич та система для вирівнювання облич. Ініціалізується сегментаційна модель BiSeNet, переміщується на GPU при потребі, завантажуються її ваги, і переводиться в режим оцінки. Також визначаються сегменти облич, які нас цікавлять (очі, шкіра, рот тощо).

Після успішної детекції модуль здійснює точне вирізання областей з обличчями, застосовуючи складні геометричні перетворення для забезпечення оптимального позиціонування. Оброблення виявлених облич та створення масок описано кодом у лістингу 2.2.

Лістинг 2.2 – Обробка виявлених облич та створення масок

```
for idx, face in enumerate(faces):
    top, left, bottom, right = (face[0:4] / adjustment).astype("int")
    top = max(0, top)
    left = max(0, left)
    bottom = min(height, bottom)
    right = min(width, right)

    confidence = face[4]
    if confidence < opt.min_confidence:
        continue

    face_height = bottom - top
    face_width = right - left
    face_size = face_height * face_width
    if face_size / (height * width) < opt.min_size:
        continue

    # Створення маски для обличчя
    mask_face = cv2.resize(aligned_face, (512, 512))
    mask_face = torch.tensor(mask_face, device=device).unsqueeze(0)
    mask_face = mask_face.permute(0, 3, 1, 2) / 255
    segments = masker(mask_face)[0]

    mask = torch.where(torch.sum(segments[:, desired_segments, :, :],
dim=1) > .7,
                       255, 0)[0]
    mask = mask.cpu().numpy()
```

У цьому циклі перебираються всі виявлені обличчя. Для кожного з них обчислюються координати, які обмежуються рамками зображення. Якщо рівень довіри до детекції або розмір обличчя надто малий, обробка пропускається. Після

цього вирівняне обличчя масштабується і передається до BiSeNet. На основі виходу сегментатора створюється бінарна маска, яка визначає, які ділянки обличчя слід залишити.

Процес нормалізації включає стандартизацію розмірів, корекцію перспективи та створення високоякісних масок для точного визначення контурів обличчя.

Навчальний модуль train.py реалізує складну систему машинного навчання, базовану на архітектурі автоенкодера з оригінальними компонентами OriginalEncoder та OriginalDecoder.

Лістинг 2.3 - Архітектура енкодера

```
class OriginalEncoder(nn.Module):
    def __init__(self):
        super().__init__()
        self.activation = nn.LeakyReLU(.1)
        self.conv_tower = nn.Sequential(nn.Conv2d(3, 128, 5, 2, padding=2),
                                        self.activation,
                                        nn.Conv2d(128, 256, 5, 2, padding=2),
                                        self.activation,
                                        nn.Conv2d(256, 512, 5, 2, padding=2),
                                        self.activation,
                                        nn.Conv2d(512, 1024, 5, 2, padding=2),
                                        self.activation)
        self.dense1 = nn.Linear(4 * 4 * 1024, 1024)
        self.dense2 = nn.Linear(1024, 4 * 4 * 1024)
        self.flatten = nn.Flatten()
        self.upscale = Upscale(512)

    def forward(self, x):
        batch_size = x.shape[0]
        x = self.conv_tower(x)
        x = self.flatten(x)
        x = self.dense1(x)
        x = self.dense2(x)
        x = torch.reshape(x, [batch_size, 1024, 4, 4])
        x = self.upscale(x)
        x = self.activation(x)
        return x
```

Цей клас описує енкодер, що складається з кількох згорткових прошарків, які поступово зменшують просторовий розмір і збільшують кількість каналів. Потім зображення перетворюється у вектор, стискається до 1024 нейронів і знову відновлюється до розміру, з яким працюватиме декодер. Наприкінці використовується Upscale() для першого кроку розгортання зображення назад.

Лістинг 2.4 - Архітектура декодера

```
class OriginalDecoder(nn.Module):
    def __init__(self):
        super().__init__()
        self.activation = nn.LeakyReLU(.1)
        self.upscale_tower = nn.Sequential(Upscale(256),
                                           self.activation,
                                           Upscale(128),
                                           self.activation,
                                           Upscale(64),
                                           self.activation)
        self.output = nn.Conv2d(64, 3, 5, padding="same")
        self.activation = nn.LeakyReLU()

    def forward(self, x):
        x = self.upscale_tower(x)
        x = self.output(x)
        x = torch.sigmoid(x)
        return x
```

Зворотна частина моделі – декодер, який перетворює латентне представлення у фінальне зображення. Тут використовуються блоки Upscale, які збільшують розмір зображення на кожному кроці. Після кожного Upscale застосовується функція активації LeakyReLU. В останньому шарі використовується згортка для перетворення до 3 каналів (RGB), а сигмоїда нормалізує результат у діапазон [0,1].

Система здатна одночасно обробляти дані від різних осіб, вивчаючи унікальні характеристики кожного обличчя та формуючи внутрішнє представлення, що дозволяє виконувати високоякісну трансформацію між різними особами.

Процес навчання моделі програмно продемонстровано в лістингу 2.5. У цьому фрагменті коду реалізовано головний цикл навчання моделі, яка виконує підміну обличчя. Спочатку формується мініпакет з випадкових зображень особи A, які проходять базову підготовку: зміну розміру, завантаження масок і можливе горизонтальне віддзеркалення. Потім ці зображення перетворюються на тензори, нормалізуються, а маски бінаризуються. Після цього модель виконує прямий прохід через енкодер і декодер, обчислює функцію втрат з урахуванням маски, виконує зворотне поширення похибки та оновлює ваги мережі.

Лістинг 2.5 - Основний цикл навчання

```
for iteration in pbar:
    # Підготовка батчу для особи A
    images = random.sample(imagesa, opt.batchsize)
    for imgnum, imagefile in enumerate(images):
        image = cv2.imread(imagefile)
        image = cv2.resize(image, (64, 64))
        mask = cv2.imread(imagefile.replace("aligned", "mask"), 0)
        mask = cv2.resize(mask, (64, 64))
        if np.random.rand() > .5:
            image = cv2.flip(image, 1)
            mask = cv2.flip(mask, 1)

        img_tensor = torch.tensor(image[..., ::-1]/255).permute(2, 0, 1)
        mask_tensor = torch.where(torch.tensor(mask) > 200, 1, 0)
        if device == "cuda":
            img_tensora[imgnum] = img_tensor.cuda()
            mask_tensora[imgnum] = mask_tensor.cuda()

    # Навчання енкодера та декодера A
    encoder_optimizer.zero_grad()
    decodera_optimizer.zero_grad()

    outa = decodera(encoder(img_tensora))
    lossa = loss_function(outa * mask_tensora, img_tensora * mask_tensora)
    lossa.backward()
    encoder_optimizer.step()
    decodera_optimizer.step()
```

Конвертаційний модуль convert.py здійснює фінальну трансформацію оброблених даних, виконуючи складний процес заміни облич у відеокадрах після завершення етапу навчання. Цей код відповідає за початкову підготовку до процесу конвертації облич. Він визначає пристрій (графічний процесор або CPU), ініціалізує архітектуру енкодера та декодера, і завантажує ваги навчених моделей з файлів. Уривок коду, який описує процес конвертації облич, наведено в лістингу 2.6.

Лістинг 2.6 – Процес конвертації облич

```
def main(opt):
    device = "cuda" if torch.cuda.is_available() and not opt.cpu else "cpu"

    encoder = OriginalEncoder()
    decoder = OriginalDecoder()

    # Завантаження навчених моделей
    encoder.load_state_dict(
        torch.load(os.path.join(opt.model_path, "encoder.pth"),
                map_location=torch.device(device)).state_dict())
    if not opt.swap:
        decoder.load_state_dict(torch.load(os.path.join(opt.model_path,
"decoderb.pth"),
                map_location=torch.device(device)).state_dict())
    else:
        decoder.load_state_dict(torch.load(os.path.join(opt.model_path,
"decodera.pth"),
                map_location=torch.device(device)).state_dict())

    # Завантаження даних про вирівнювання
    with open(os.path.join(opt.extract_path, "face_alignments.json"), "r",
encoding="utf-8") as alignment_file:
        alignment_data = json_tricks.loads(alignment_file.read())
```

Залежно від параметра swap, вибирається, який декодер використовувати для створення обличчя. Також завантажується JSON-файл із метаданими про вирівнювання, що буде використовуватись далі для трансформацій. Подальший код наведено в лістингу 2.7.

На цьому етапі здійснюється процес заміни обличчя в кадрі. Спочатку виконується афінне перетворення вхідного зображення для стандартизації позиції обличчя, після чого зображення зменшується до розміру, який підходить для нейромережі. Далі модель генерує відповідне обличчя, яке потім масштабують назад до більшої роздільності. Результат змішується з оригінальним обличчям за допомогою маски, і врешті вставляється у вихідний кадр за допомогою зворотного перетворення.

Цей модуль використовує навчені моделі для генерації реалістичних зображень з підміненими обличчями, забезпечуючи при цьому збереження природності міміки та загального вигляду.

Лістинг 2.7 – Застосування трансформації та змішування

```
      for idx, face in enumerate(alignment_data[file]['faces']):
          aligned_face = cv2.warpAffine(image_rgb, face["warp_matrix"][:2], (256,
256))
          aligned_face_tensor          =          torch.tensor(aligned_face/255,
dtype=torch.float32).permute(2, 0, 1)
          aligned_face_tensor_small = torch.nn.functional.interpolate(
              aligned_face_tensor.unsqueeze(0),  size=(64,64),  mode='bilinear',
align_corners=False)

          with torch.no_grad():
              output_face_tensor = decoder(encoder(aligned_face_tensor_small))

          output_face_tensor                                                    =
torch.nn.functional.interpolate(output_face_tensor, size=(256,256))

          # Застосування маски для змішування
          mask_img          =          cv2.imread(os.path.join(opt.extract_path,
f"face_mask_{filename}_{idx}.png"), 0)
          mask_tensor = torch.where(torch.tensor(mask_img) > 200, 1, 0)

          output_face_tensor = (aligned_face_tensor.cpu() * (1 - mask_tensor)) +
(output_face_tensor.cpu() * mask_tensor)

          output_face                                                           =
(output_face_tensor[0].permute(1,2,0).numpy()*255).astype(np.uint8)
          output_image  =  cv2.warpAffine(output_face,  face["warp_matrix"][:2],
(height, width),
                                          output_image,
borderMode=cv2.BORDER_TRANSPARENT,
                                          flags=cv2.WARP_INVERSE_MAP)
```

Допоміжна підсистема включає спеціалізований модуль bisenet.py, який реалізує сучасну нейронну архітектуру BiSeNet, спеціально призначену для високоточної семантичної сегментації областей обличчя. Ця технологія дозволяє системі точно відрізняти різні частини обличчя від фону та інших об'єктів у кадрі. Модуль models.py містить детальну реалізацію архітектури автоенкодера, включаючи всі необхідні компоненти енкодера та декодерів з оптимізованими параметрами для досягнення найкращої якості результату. Програмна реалізація базової архітектури BiSeNet продемонстрована в лістингу 2.8.

Початкова фаза підготовки даних представляє собою комплексний технологічний процес, що розпочинається з декомпозиції вихідного відеоматеріалу на послідовність статичних фреймів. Для виконання цієї критично важливої операції система використовує потужні можливості професійної утиліти ffmpeg, яка забезпечує високоякісне розбиття відео з точним збереженням візуальних характеристик кожного кадру.

Лістинг 2.8 – Основна архітектура BiSeNet

```
class BiSeNet(nn.Module):
    def __init__(self, n_classes, *args, **kwargs):
        super(BiSeNet, self).__init__()
        self.cp = ContextPath()
        self.ffm = FeatureFusionModule(256, 256)
        self.conv_out = BiSeNetOutput(256, 256, n_classes)
        self.conv_out16 = BiSeNetOutput(128, 64, n_classes)
        self.conv_out32 = BiSeNetOutput(128, 64, n_classes)
        self.init_weight()

    def forward(self, x):
        H, W = x.size()[2:]
        feat_res8, feat_cp8, feat_cp16 = self.cp(x)
        feat_sp = feat_res8
        feat_fuse = self.ffm(feat_sp, feat_cp8)

        feat_out = self.conv_out(feat_fuse)
        feat_out16 = self.conv_out16(feat_cp8)
        feat_out32 = self.conv_out32(feat_cp16)

        feat_out = F.interpolate(feat_out, (H, W), mode='bilinear',
align_corners=True)
        feat_out16 = F.interpolate(feat_out16, (H, W), mode='bilinear',
align_corners=True)
        feat_out32 = F.interpolate(feat_out32, (H, W), mode='bilinear',
align_corners=True)
        return feat_out, feat_out16, feat_out32
```

Лістинг 2.9 демонструє основні команди, які використовуються на різних етапах роботи з системою. Перша команда виконує детекцію облич у відеокадрах, друга – запускає процес навчання моделі на двох датасетах, а третя – застосовує модель для конвертації облич у відео. Кожна команда використовує відповідні параметри, такі як шлях до даних, розмір зображення чи кількість ітерацій. Приклад запуску основних модулів системи наведено в лістингу 2.9.

Лістинг 2.9 - Приклад запуску основних модулів системи

```
Bash
# Виявлення та обробка облич
python face_detection.py “data/input_frames/” --size 256 --min_confidence 0.9

# Навчання моделі
python train.py “data/A/” “data/B/” --iterations 50000 --batchsize 16

# Конвертація зображень
python convert.py “data/input_frames/” --model-path “model/” --extract-path “data/face_images/”
```

Кожен отриманий фрейм підлягає ретельній обробці у спеціалізованому модулі face_detection.py, який представляє собою складну систему комп’ютерного зору. Цей модуль інтегрує передові технології, включаючи потужну бібліотеку face_alignment, призначену для високоточного визначення позиції та орієнтації облич у просторі.

Блок коду в лістингу 2.10 виконує точне вирівнювання обличчя за п’ятьма ключовими точками: очима, носом і кутиками рота. Спочатку виявляються landmarks на обличчі, після чого обчислюється афінне перетворення (з урахуванням масштабу, повороту і зміщення) за допомогою алгоритму Umeyama. Потім ця трансформація застосовується до зображення, щоб отримати вирівняте обличчя, з яким зручно працювати у подальших етапах.

Система здатна працювати з обличчями у різних ракурсах та умовах освітлення, забезпечуючи стабільні результати навіть у складних сценаріях. Паралельно з процесом детекції облич система застосовує передову модель BiSeNet.pth для створення прецизійних масок сегментації.

Всі оброблені зображення підлягають стандартизації розмірів до уніфікованого формату 64x64 пікселі, що забезпечує оптимальний баланс між якістю деталізації та обчислювальною ефективністю процесу навчання. Цей розмір був обраний на основі експериментальних досліджень та забезпечує достатню деталізацію для збереження індивідуальних характеристик облич при мінімізації обчислювальних вимог.

Лістинг 2.10 – Визначення ключових точок обличчя

```
pythonlandmarks = face_aligner.get_landmarks_from_image(
    image_rgb,
    detected_faces=[face[0:4] / adjustment])

right_eye = np.mean(landmarks[0][36:42], axis=0)
left_eye = np.mean(landmarks[0][42:48], axis=0)
nose_tip = landmarks[0][30]
right_mouth = landmarks[0][48]
left_mouth = landmarks[0][54]

limited_landmarks = np.stack((right_eye, left_eye, nose_tip, right_mouth,
left_mouth))

# Вирівнювання обличчя за ключовими точками
mean_face = np.array([[0.25, 0.22],
                      [0.75, 0.22],
                      [0.50, 0.51],
                      [0.26, 0.78],
                      [0.74, 0.78]])

warp_matrix = umeyama(limited_landmarks,
                      mean_face * (opt.size * .3) + (opt.size * .35),
                      True)

aligned_face = cv2.warpAffine(aligned_face,
                              warp_matrix[:2],
                              (opt.size, opt.size))
```

Центральний етап навчання нейронної мережі реалізується через складну систему, втілену у скрипті train.py. Процес навчання здійснюється з використанням сучасного оптимізатора Adam, який забезпечує стабільну та ефективну конвергенцію моделі.

У лістингу 2.11 налаштовуються оптимізатори та функції втрат для процесу навчання. Оптимізатори типу Adam використовуються окремо для енкодера та декодерів, причому для енкодера встановлено зменшену швидкість навчання. Основною функцією втрат є MSELoss, але також використовується функція з абсолютною помилкою (L1) для додаткової оцінки якості. Таке поєднання забезпечує стабільність і точність під час тренування моделі.

Функція втрат MSELoss застосовується для мінімізації різниці між оригінальними та реконструйованими зображеннями, забезпечуючи високу якість відтворення деталей обличчя. Навчальні дані організовуються у пакети оптимального розміру, що дозволяє ефективно використовувати доступну пам’ять GPU та забезпечувати стабільність градієнтного спуску.

Лістинг 2.11 – Конфігурація оптимізаторів та функції втрат

```
      pythonencoder_optimizer      =      torch.optim.Adam(encoder.parameters(),
lr=opt.learning_rate/2)
      decodera_optimizer        =        torch.optim.Adam(decodera.parameters(),
lr=opt.learning_rate)
      decoderb_optimizer        =        torch.optim.Adam(decoderb.parameters(),
lr=opt.learning_rate)
      loss_function = MSELoss()

      def calcloss(original, out):
          return torch.abs(out-original).mean()
```

Система автоматично адаптується до доступних обчислювальних ресурсів, оптимально розподіляючи навантаження між центральним процесором та графічним прискорювачем. При наявності CUDA-сумісного графічного адаптера система автоматично переносить обчислення на нього, що може забезпечити прискорення у десятки разів порівняно з обчисленнями на CPU.

Етап постобробки, реалізований у модулі convert.py, представляє собою технологічно складний процес застосування навчених моделей для генерації фінальних результатів. Система використовує збережені параметри енкодера та декодерів для виконання трансформації облич у реальному часі. Особливістю цього етапу є застосування зворотних афінних перетворень, які були збережені під час початкового етапу вирізання облич.

Ці математичні перетворення дозволяють системі точно позиціонувати згенеровані обличчя на їх оригінальних позиціях у кадрі, забезпечуючи природність розміщення та збереження оригінальної перспективи. Система також застосовує складні алгоритми змішування країв для забезпечення плавного переходу між заміненим обличчям та навколишнім контекстом кадру.

Результати обробки кожного кадру систематично зберігаються у спеціально організованій директорії convert з логічною системою іменування файлів, що полегшує подальшу роботу з отриманими даними та забезпечує можливість якісного контролю результатів.

Завершальний етап формування фінального відеопродукту здійснюється через повторне використання потужних можливостей утиліти ffmpeg для об’єднання всіх оброблених кадрів у цілісний відеофайл. Система підтримує

гнучке налаштування параметрів кодування, включаючи частоту кадрів, якість стиснення та формат вихідного файлу, що дозволяє оптимізувати результат під конкретні вимоги проєкту.

Модульна архітектура системи забезпечує виняткову гнучкість розвитку та адаптації під змінні вимоги. Кожен компонент може бути незалежно модифікований, оптимізований або повністю замінений на альтернативну реалізацію без необхідності внесення змін до інших частин системи. Така архітектурна філософія забезпечує довготривалу супроводжуваність коду та можливість швидкої адаптації до нових технологічних рішень у галузі комп’ютерного зору та машинного навчання.

РОЗДІЛ 3
ТЕСТУВАННЯ ПРОГРАМНОЇ СИСТЕМИ

3.1 Мета та підхід до тестування

Основною метою тестування є перевірка коректності роботи всіх компонентів системи зі створення deepfake-відео на базі PyTorch. Тестування проводилося зосереджено на трьох основних етапах: правильність виявлення та обробки облич у вхідних зображеннях (рис. 3.1), перевірка якості роботи навченої автоенкодерної моделі, а також верифікація функціональності конвертації та генерації фінальних результатів.

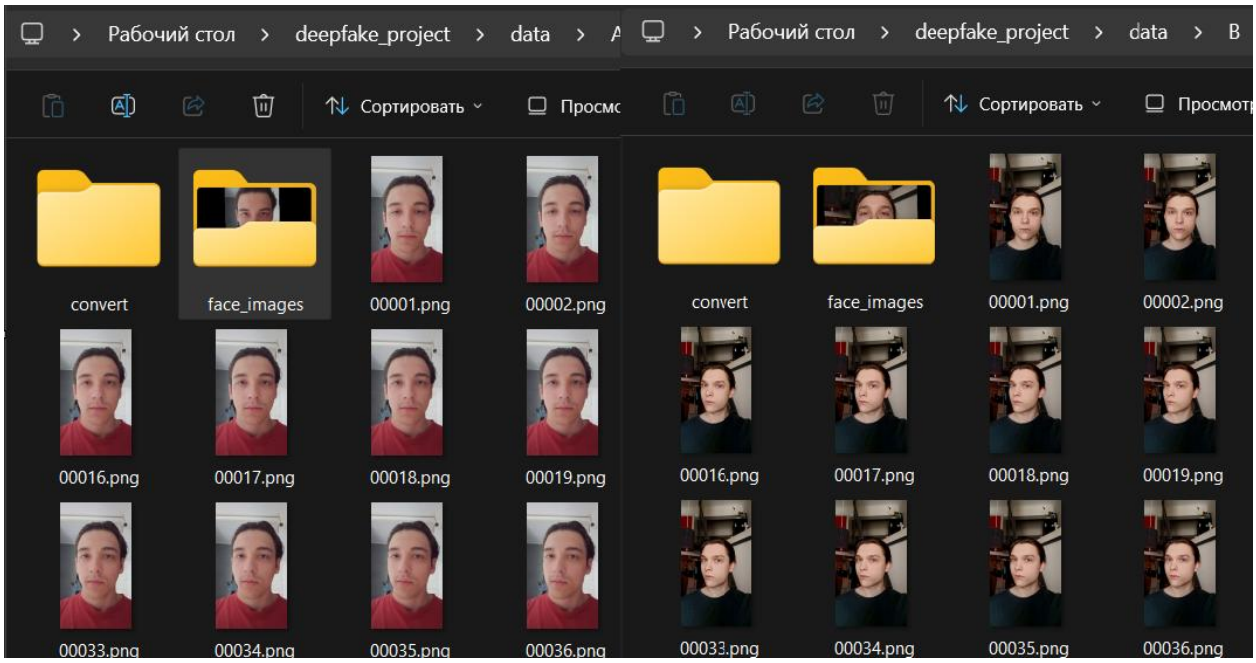


Рисунок 3.1 – Зібраний датасет з video.mp4

Було використано комплексний підхід до тестування, що включав ручне тестування CLI-інструментів (face_detection.py, train.py, convert.py), автоматизовану перевірку коректності обробки даних, а також візуальну оцінку результатів у вигляді згенерованих зображень.

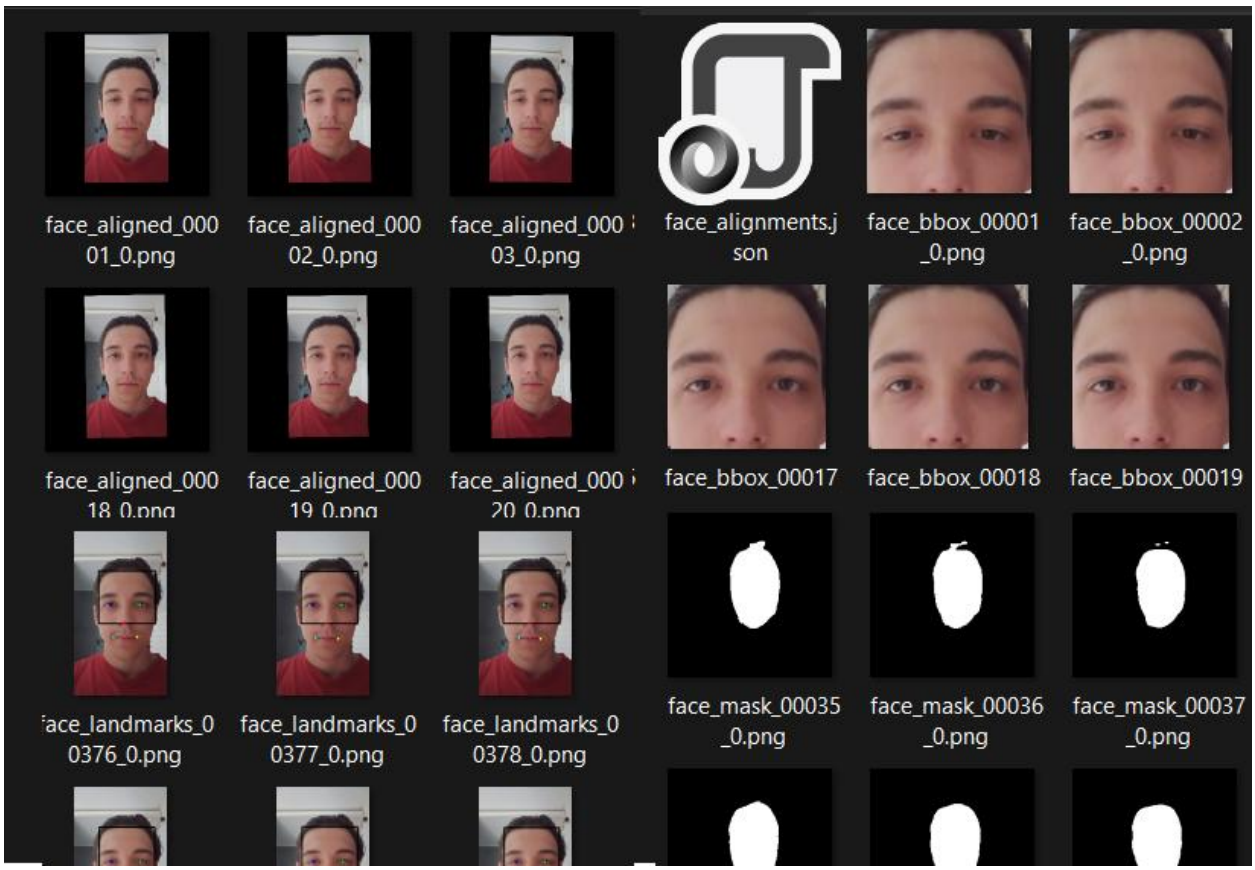


Рисунок 3.2 – Вміст директорії face_images після виконання скрипта face_detection.py

Особлива увага приділялася перевірці правильності роботи детектора облич на основі SFD (S3FD Face Detector), точності вирівнювання облич за допомогою landmark-точок, а також якості генерації масок для коректного накладення результатів.

3.2 Результати тестування моделі

Модель була навчена протягом 100,000 ітерацій з використанням архітектури автоенкодера, що складається зі спільного енкодера та двох окремих декодерів для кожного типу облич. Якість результатів перевірялась шляхом візуального порівняння відтворених фреймів зі справжніми, а також аналізу значень функції втрат під час навчання.

Модель демонструвала здатність до відтворення основних рис обличчя з помітними артефактами при великих поворотах голови або зміні освітлення. Функція втрат стабілізувалась після приблизно 50,000 ітерацій, досягнувши значень близько 0.01-0.02 для обох декодерів. Було помічено, що якість результатів значною мірою залежить від кількості та різноманітності фреймів у навчальному датасеті.

Рисунок 3.3 – Огляд prewiev.png після тренування

У процесі тестування були також перевірені: узгодженість розмірів вхідних тензорів (64×64 пікселі), здатність енкодера створювати осмислені латентні представлення розміром 512×8×8, коректність роботи декодерів при реконструкції зображень, а також стабільність навчання при використанні оптимізатора Adam з різними швидкостями навчання для енкодера та декодерів. Модель показала задовільну працездатність навіть при обмежених обчислювальних ресурсах, успішно навчаючись на GPU.

3.3 Тестування кінцевого результату

Після завершення навчання моделі було відтестовано повний пайплайн конвертації з використанням скрипта convert.py (рис. 3.4), що дозволяє

генерувати нові зображення з підміненим обличчям на основі збережених параметрів вирівнювання з етапу детекції. Процес конвертації включав завантаження натренованих моделей, читання даних про вирівнювання облич з JSON-файлу та застосування зворотного афінного перетворення для інтеграції згенерованого обличчя в оригінальне зображення.

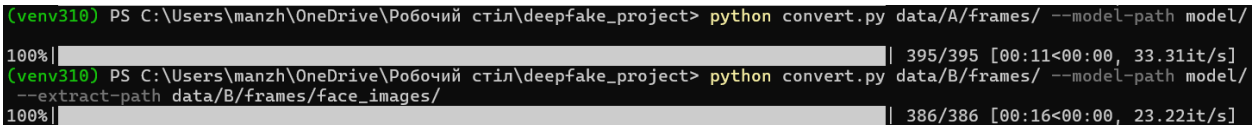


Рисунок 3.4 – Виконання convert.py без swap

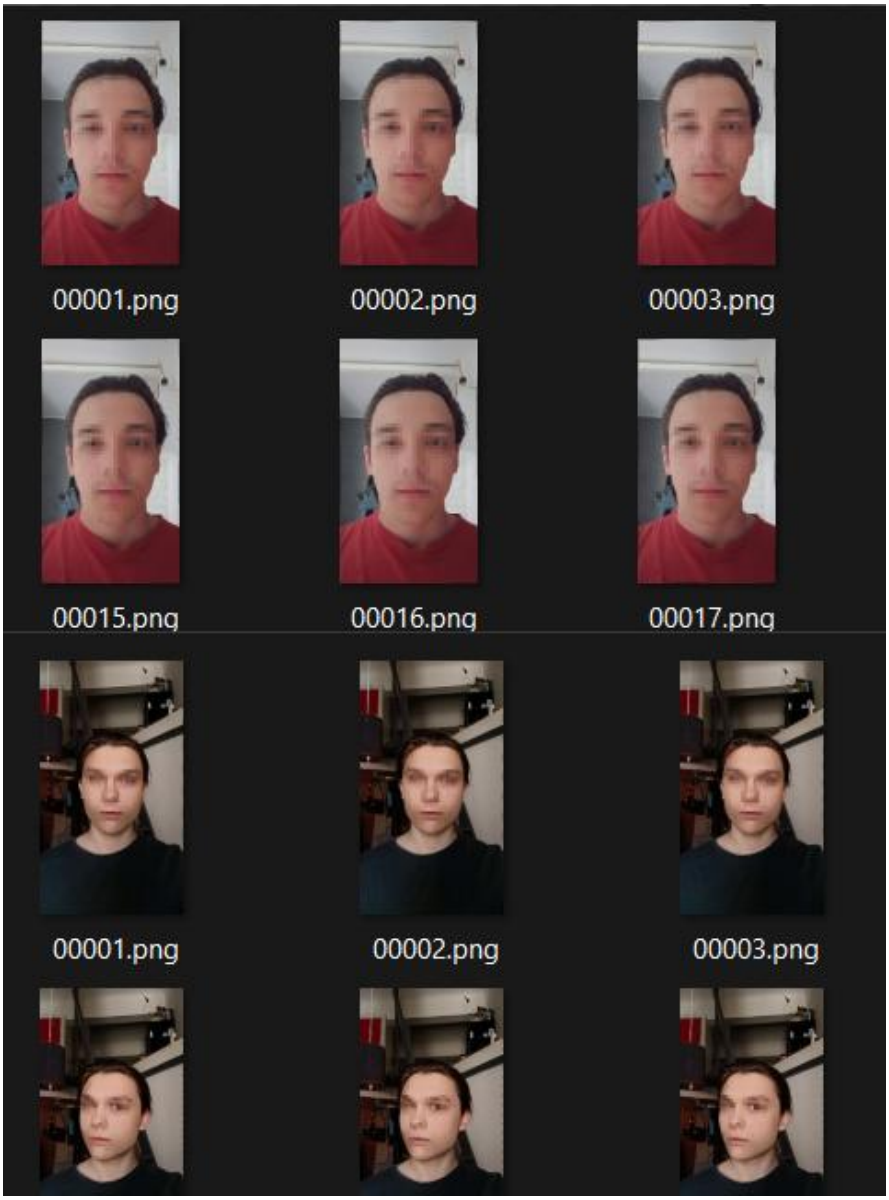


Рисунок 3.5 – Результат виконання convert.py без swap

Перевірка функціональності конвертації включала: збереження відповідності оригінальній позі та освітленню завдяки використанню масок, створених за допомогою мережі BiSeNet; відповідність отриманих результатів очікуваному розміру та формату; коректність застосування зворотного афінного перетворення для природного вписування згенерованого обличчя в оригінальний контекст зображення. Система маскування показала високу ефективність у виділенні областей обличчя що забезпечило плавні переходи між згенерованими та оригінальними частинами зображення.

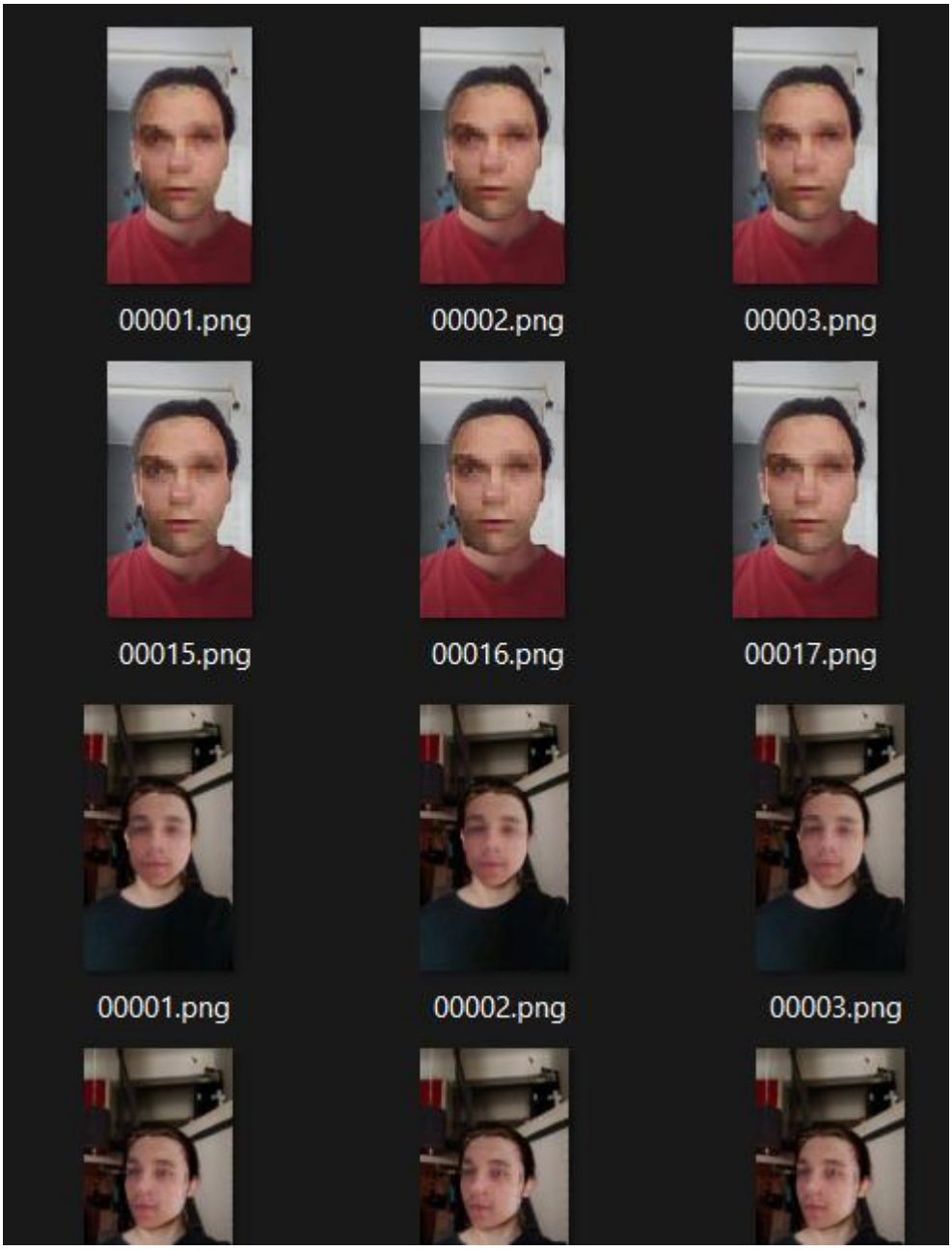


Рисунок 3.6 – Результат виконання convert.py зі swap

Тестування включало перевірку стабільності результатів при різних умовах освітлення та ракурсах, а також сумісності з різними форматами вхідних зображень. Основними проблемами, виявленими на цьому етапі, стали сильна втрата деталей при масштабуванні з 64×64 до повного розміру зображення, а також необхідність ретельного налаштування порогових значень для масок для досягнення оптимального балансу між природністю результату та збереженням згенерованих рис обличчя.

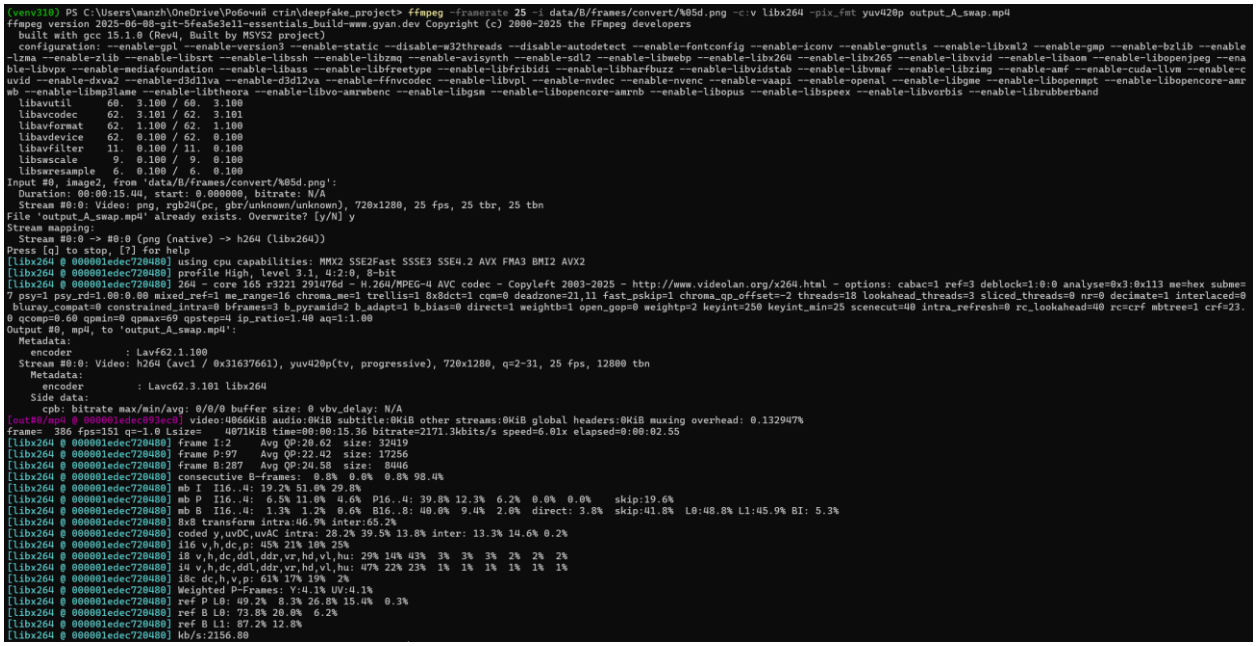


Рисунок 3.7 – Процес збирання конвертованих фреймів зі swap A->B в відео

З отриманого результату видно, що накладене обличчя виглядає пікселізованим і нечітким (рис. 3.8). Основна причина цього полягає в використанні моделі, що працює з дуже низькою роздільною здатністю. Автоенкодер, який було реалізовано, навчався на фреймах розміром 64×64 пікселі. Це надто мала роздільна здатність, щоб зберігати чіткість і дрібні деталі обличчя, особливо при збільшенні зображення до повнорозмірного кадру. Після реконструкції обличчя масштабується назад до розміру відео, але візуальна якість залишається обмеженою. Саме тому з’являються розмиті контури, піксельні переходи та штучний вигляд.

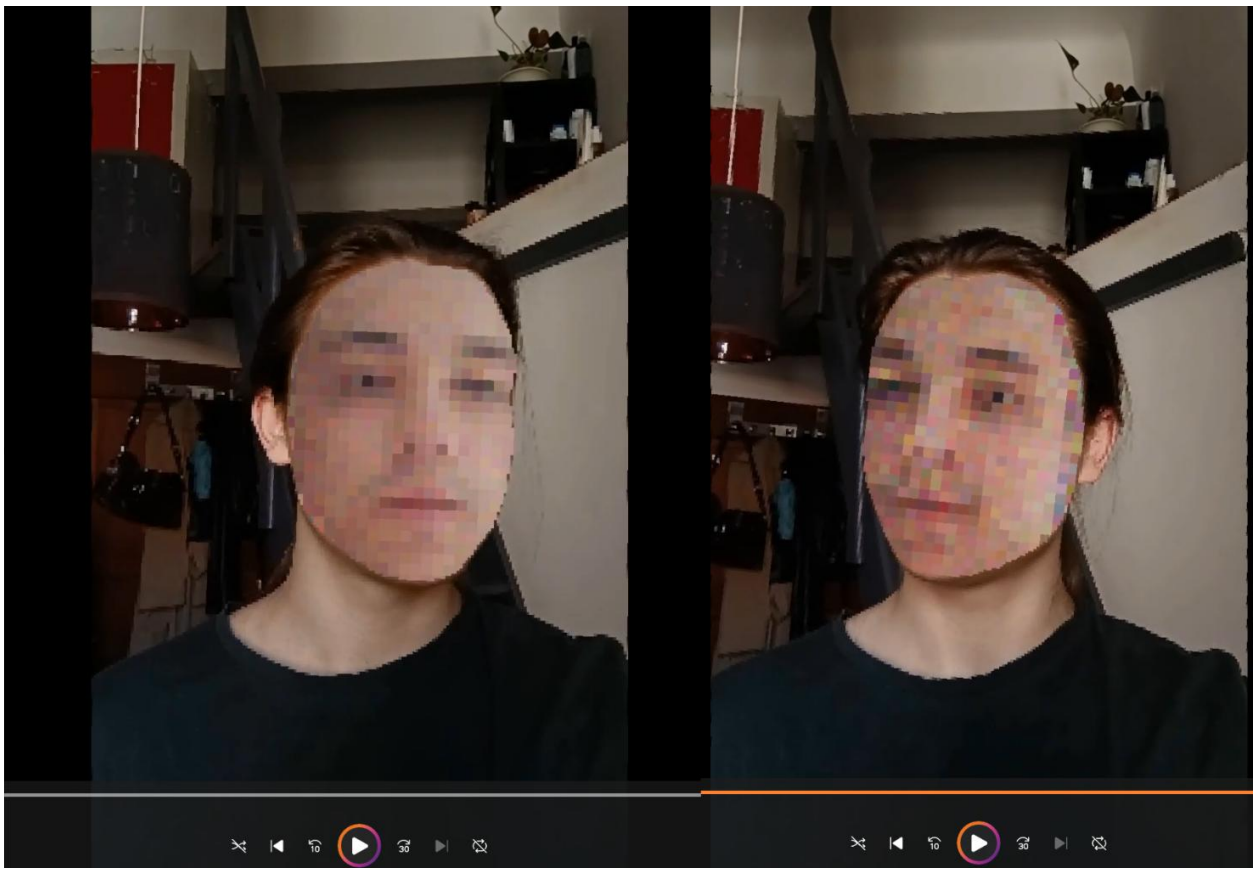


Рисунок 3.8 – Фрейми з зібраного відео output_A_swap.mp4

Проблема не в самій моделі, а в її архітектурних обмеженнях. У такому вигляді вона демонструє працездатність, тобто дійсно навчається, розпізнає і замінює обличчя. Але якість результату – це компроміс між продуктивністю й візуальною правдоподібністю. Щоб покращити якість, можна модифікувати модель під більший розмір (наприклад, 128×128 або 256×256), застосовувати алгоритми постобробки на кшталт Real-ESRGAN для покращення згенерованих зображень, або підбирати більш складні архітектури. Також допоможе більший і різноманітніший датасет з кадрами при різному освітленні, ракурсах і виразах обличчя. У поточному стані система має очевидні обмеження, які, все ж, дають простір для вдосконалення й розвитку в майбутньому.

ВИСНОВКИ

У ході виконання дипломної роботи було детально досліджено архітектуру систем глибинного навчання, які лежать в основі deepfake-технології. На основі вивченої теоретичної бази було реалізовано практичний проєкт з побудови власної системи для створення відео з підміною облич, використовуючи архітектуру автоенкодера. Було здійснено повний цикл розробки: від збору та підготовки вхідних даних до тренування моделі, створення інструментів постобробки і генерації фінального результату.

Особливу увагу було приділено вибору інструментів і фреймворків. У реалізації було використано мову програмування Python, фреймворк PyTorch, а також бібліотеки для обробки зображень та відео, такі як OpenCV. Було протестовано основні компоненти системи, зокрема модулі вирізання облич, сегментації, тренування і генерації, а також перевірено працездатність всього pipeline у різних умовах.

У результаті роботи вдалося досягти стабільного функціонування системи та отримати перші deepfake-відео. Разом із тим, під час тестування були виявлені обмеження, пов’язані з низькою роздільною здатністю латентного простору (64×64), що впливає на деталізацію кінцевого результату.

Отримані результати підтверджують можливість створення функціональної deepfake-системи на базі PyTorch, придатної для подальшої модифікації, досліджень або експериментального застосування в освітніх і наукових цілях. Надалі перспективними напрямами розвитку можуть стати перехід до архітектур з вищою роздільною здатністю.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

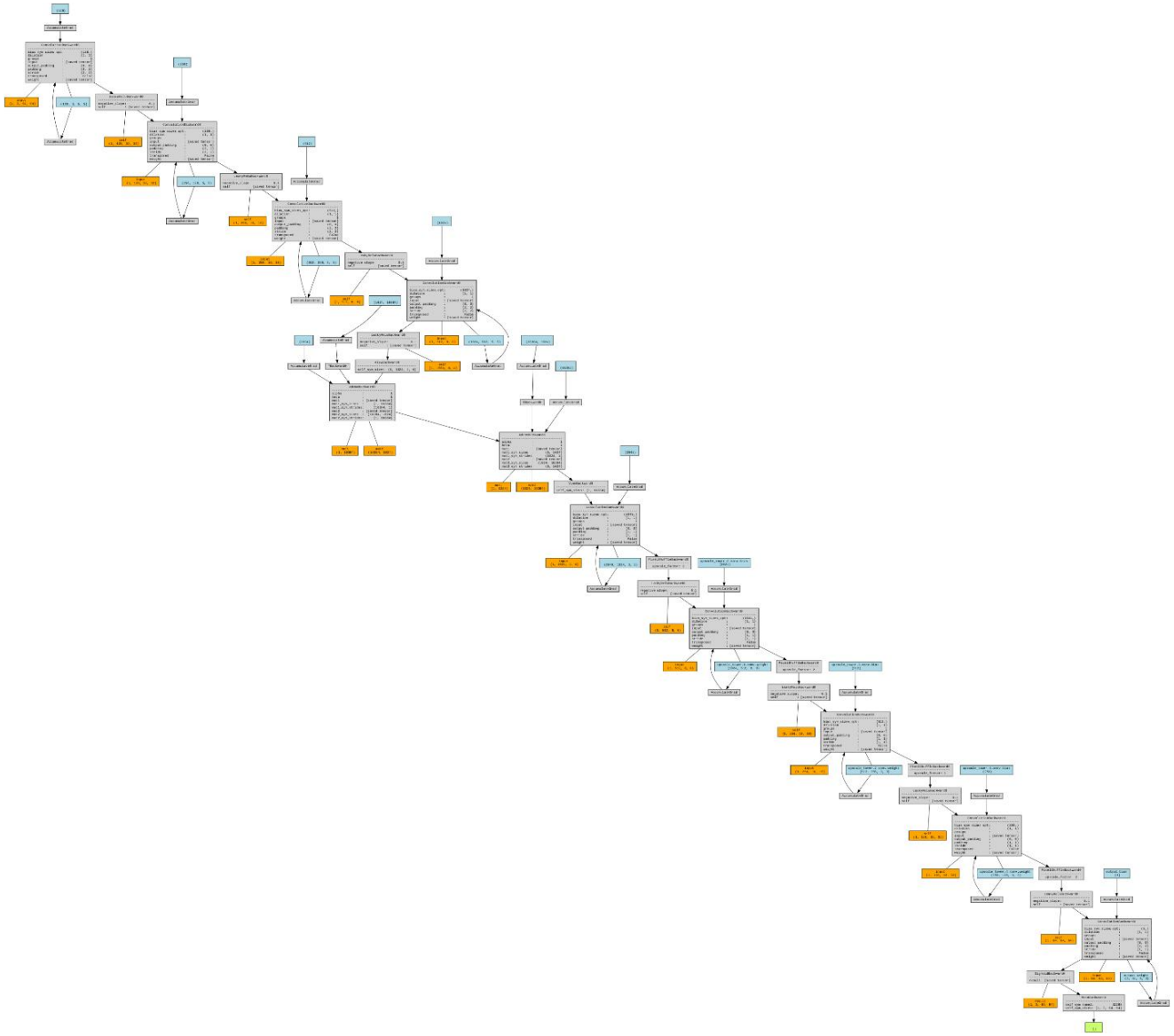
1. Lyon, B., & Tora, M. (2023). Exploring Deepfakes: Deploy powerful AI techniques for face replacement and more with this comprehensive guide. Birmingham: Packt Publishing.
2. Agiliway. (2025). Як TensorFlow і PyTorch розвивають глибоке навчання. URL: https://www.agiliway.com/uk-ua/yak-tensorflow-i-pytorch-rozvyvayut-glyboke-navchannya/
3. Khalid, S., et al. (2024). A comprehensive survey on GAN-based deepfake generation and detection techniques. Applied Sciences, 15(2), 923. URL: https://www.mdpi.com/2076-3417/15/2/923
4. Tolosana, R., Vera-Rodriguez, R., Fierrez, J., Morales, A., & Ortega-Garcia, J. (2020). Deepfakes and beyond: A survey of face manipulation and fake detection. Information Fusion, 64, 131–148. DOI: 10.1016/j.inffus.2020.07.007.
5. Abbas, F., & Taeihagh, A. (2024). Unmasking deepfakes: A systematic review of deepfake detection and generation techniques using artificial intelligence.Expert Systems With Applications. https://doi.org/10.1016/j.eswa.2024.124260
6. Walorska, A. M. (2020). *Deepfakes & Disinformation* (V. Oliinyk, Trans.). Friedrich Naumann Foundation for Freedom. URL: https://www.aup.com.ua/uploads/DEEPFAKES_FNF_AUP_2020.pdf
7. Gao, S.-H., Cheng, M.-M., et al. (2021). *Res2Net:* A New Multi-scale Backbone Architecture // IEEE Transactions on Pattern Analysis and Machine Intelligence. URL: https://arxiv.org/abs/1904.01169
8. Jaursch, J. (2019). Regulatorische Reaktionen auf Desinformation. Stiftung Neue Verantwortung. URL: https://www.interface-eu.org/storage/archive/files/regulatorische_reaktionen_auf_desinformation.pdf
9. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., ... & Bengio, Y. (2014). Generative adversarial nets. Advances in neural information processing systems, 27.

10. Cloudfresh. (2024). AWS vs Azure vs Google Cloud: Вибір платформи для ШІ. URL: https://cloudfresh.com/ua/cloud-blog/aws-vs-azure-vs-google-cloud/.
11. Makhzani, A., Shlens, J., Jaitly, N., Goodfellow, I., & Frey, B. (2016). Adversarial Autoencoders // URL: https://arxiv.org/abs/1511.05644Z
12. Zhu, J. Y., Park, T., Isola, P., & Efros, A. A. (2017). Unpaired image-to-image translation using cycle-consistent adversarial networks. Proceedings of the IEEE international conference on computer vision.
13. Siarohin, A., et al. (2019). First Order Motion Model for Image Animation. URL: https://github.com/AliaksandrSiarohin/first-order-model
14. OpenCV Team. (2021). OpenCV: Open Source Computer Vision Library. URL: https://opencv.org/
15. Dlib C++ Library. (2021). King, D. E. Dlib-ml: A machine learning toolkit. Journal of Machine Learning Research, 10.
16. DeepFaceLab. (2021). DeepFaceLab: The leading software for creating deepfakes. URL: https://github.com/iperov/DeepFaceLab
17. FaceSwap. (2021). FaceSwap: The purpose of this software is to democratize deepfakes. URL: https://github.com/deepfakes/faceswap
18. Kingma, D. P., & Welling, M. (2014). Auto-Encoding Variational Bayes. URL: https://arxiv.org/abs/1312.6114
19. Unite.AI. (2024). Топ інструменти MLOps: Weights & Biases, Comet, Neptune тощо. – URL: https://www.unite.ai/uk/top-mlops-tools-guide-weights-biass-comet-%D1%82%D0%BE%D1%89%D0%BE/.
20. Alanazi, S., & Asif, S. (2024). Exploring deepfake technology: creation, consequences and countermeasures // Human-Intelligent Systems Integration. URL: https://link.springer.com/article/10.1007/s42454-024-00054-8
21. ikropyvnytskyi.com. Технологія Deepfake: як створюються фейкові відео та як їх розпізнати. URL: https://ikropyvnytskyi.com/uk/articles-14565-tehnologiya-deepfake-yak-stvoryuyutsya-fejkovi-video-ta-yak-yih-rozpiznaty.

22. He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. Proceedings of the IEEE conference on computer vision and pattern recognition.
23. Tarnawski, M., et al. (2021). Deep learning architectures for image translation tasks. Journal of Real-Time Image Processing, 18, 97–115. URL: https://link.springer.com/article/10.1007/s11554-021-01166-z
24. Kaur J. Generative Adversarial Networks (GANs) for Image Synthesis. URL: https://www.xenonstack.com/blog/gans-for-image-synthesis (дата звернення: 02.06.2025).
25. NVIDIA Developer. (2024). CUDA Toolkit Documentation. URL: https://developer.nvidia.com/cuda-toolkit Intel. (2023)
26. OpenVINO Toolkit Overview. URL: https://www.intel.com/content/www/us/en/developer/tools/openvino-toolkit/overview.html
27. Nirkin, Y., Keller, Y., & Hassner, T. (2019). FSGAN: Subject agnostic face swapping and reenactment. Proceedings of the IEEE/CVF international conference on computer vision.
28. Zakharov, E., Shysheya, A., Burkov, E., & Lempitsky, V. (2019). Few-Shot Adversarial Learning of Realistic Neural Talking Head Models // URL: https://arxiv.org/abs/1905.08233
29. GilkaHub. (2024). AI Tools – Штучний інтелект для бізнесу, маркетингу та дизайну // GilkaHub. – URL: https://www.gilkahub.com/ai
30. Dang, A. T. (2021). How to Perform a Facial Landmark Detection with Keras. URL: https://pub.towardsai.net/how-to-perform-a-facial-landmark-detection-with-keras-eb6b4e7a2722

ДОДАТКИ

Додаток А – Архітектура модуля декодера



Додаток Б – Програмна реалізація

Репозиторій з програмною реалізацією, виконаною в межах кваліфікаційної роботи, розташовано за адресою: https://github.com/DeniXRX/deepfake_project