

ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КИЇВСЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ
ІМЕНІ ТАРАСА ШЕВЧЕНКА

Кафедра програмних систем і технологій

НІКОЛАСНКО А. Ю., ЮРЧУК І. А., ПИЛИПЕНКО А. І.

ПРИКЛАДНЕ ГЛИБОКЕ НАВЧАННЯ

Методичні вказівки до лабораторних занять
для здобувачів освітнього рівня «Магістр» зі спеціальності
121 – «Інженерія програмного забезпечення»

Київ – 2026

Рецензенти:

д-р техн. наук, професор **Жебка В. В.** – завідувач кафедри технологій цифрового розвитку Державного університету інформаційно-комунікаційних технологій

канд. техн. наук, доцент **Гладка М. В.** – доцент кафедри інформаційних систем та технологій Київського національного університету імені Тараса Шевченка

Рекомендовано до публікації в електронному вигляді кафедрою програмних систем і технологій Київського національного університету імені Тараса Шевченка, протокол № 14 від 01.04.2026 р.

Рекомендовано до публікації в електронному вигляді Вченою радою факультету інформаційних технологій Київського національного університету імені Тараса Шевченка, протокол №12 від 20.05.2026 р.

Прикладне глибоке навчання [Електронний ресурс]: методичні вказівки до лабораторних занять. Кафедра програмних систем і технологій / **А. Ю. Ніколаєнко, І. А. Юрчук, А. І. Пилипенко** – К.: КНУ імені Тараса Шевченка, 2026. – 41 с.

Методичні вказівки до лабораторних занять з дисципліни «Прикладне глибоке навчання» призначені для здобувачів другого курсу освітнього рівня «Магістр» денної форми навчання зі спеціальності 121 – «Інженерія програмного забезпечення» за освітньою програмою «Інженерія програмного забезпечення» галузі знань 12 – «Інформаційні технології».

Публікується в авторській редакції.

Електронна версія цього видання опублікована на сайті кафедри програмних систем і технологій за адресою: <https://pst.knu.ua/library> (дата публікації «02» квітня 2026 р.)

© Ніколаєнко А. Ю., Юрчук І. А., Пилипенко А. І., 2026 р.

ЗМІСТ

ВСТУП.....	4
Лабораторна робота №1 Основи роботи з даними для глибокого навчання	7
Теоретичні відомості.....	7
Завдання на лабораторну роботу	12
Зміст звіту	13
Питання для самоконтролю	13
Лабораторна робота №2 Навчання з учителем	15
Теоретичні відомості.....	15
Завдання на лабораторну роботу	18
Зміст звіту	19
Питання для самоконтролю	20
Лабораторна робота №3 Навчання без учителя	21
Теоретичні відомості.....	21
Завдання на лабораторну роботу	23
Зміст звіту	25
Питання для самоконтролю	26
Лабораторна робота №4 Генеративні моделі глибокого навчання	27
Теоретичні відомості.....	27
Завдання на лабораторну роботу	32
Зміст звіту	33
Питання для самоконтролю	34
Лабораторна робота №5 Оцінка моделей, метрики якості	35
Теоретичні відомості.....	35
Завдання на лабораторну роботу	37
Зміст звіту	38
Питання для самоконтролю	39
ЛІТЕРАТУРНІ ДЖЕРЕЛА	41

ВСТУП

Навчальна дисципліна «Прикладне глибоке навчання» спрямована на вивчення основних концепцій і технологій глибокого навчання, включно з принципами побудови та функціонування різних типів нейронних мереж. Набуваються практичні вміння з проєктування, тренування та оцінювання моделей глибокого навчання, а також їх ефективної реалізації в програмному коді.

Мета дисципліни – надати здобувачам освітнього рівня «Магістр» теоретичні знання та сформувати практичні навички із застосування методів і технологій глибокого навчання для розв’язання прикладних задач у галузі інженерії програмного забезпечення.

Завдання дисципліни полягає у формуванні у здобувачів освіти здатності створювати та обробляти набори даних, обґрунтовано вибирати архітектуру нейронної мережі для виконання задач комп’ютерного зору, розробляти моделі глибокого навчання із застосуванням сучасних інструментів і фреймворків, проєктувати та розробляти програмне забезпечення з використанням моделей глибокого навчання, оцінювати та забезпечувати ефективність моделей глибокого навчання, знаходити оптимальні шляхи вирішення практичних завдань та передбачувати їх наслідки в галузі інженерії програмного забезпечення.

Здобувач освіти має знати основні концепції та принципи програмування на мові Python; теоретичні основи лінійної алгебри, зокрема, матричних та векторних обчислень; базові принципи обробки великих масивів даних. Здобувач освіти має вміти застосовувати методи та інструменти високопродуктивних обчислень; здійснювати передобробку, візуалізацію та аналіз великих обсягів даних.

В результаті вивчення дисципліни здобувач освіти має опанувати основні поняття і технології глибокого навчання, методи підготовки великих наборів даних, принципи архітектурної побудови нейронних мереж різних типів,

стратегії навчання з учителем та без учителя, а також сучасні підходи до генерації контенту, та комплексного оцінювання якості моделей. Окрему увагу приділено методології комплексного оцінювання якості моделей, опанування якої дасть змогу здобувачу освіти об'єктивно аналізувати їхню точність, надійність та ефективність у розв'язанні прикладних задач.

Перша лабораторна робота присвячена підготовці даних як фундаменту для навчання ефективних моделей. У ході її виконання відпрацьовуються навички збору, очищення та анотування зображень, а також формування репрезентативних вибірок (train/val/test). Особлива увага приділяється методам аугментації за допомогою бібліотеки Pillow для створення великомасштабних датасетів на основі обмеженої кількості вихідних зразків.

Друга лабораторна робота присвячена парадигмі навчання з учителем на прикладі згорткових нейронних мереж (CNN). У ході її виконання відпрацьовуються навички проектування багат шарових архітектур, підбору функцій активації та оптимізаторів. Особлива увага приділяється дослідженню впливу методів регуляризації, таких як Dropout та Batch Normalization, на здатність моделі до узагальнення та швидкість її збіжності.

Третя лабораторна робота присвячена методам навчання без учителя, зокрема архітектурі симетричних автоенкодерів. У ході її виконання відпрацьовуються навички побудови компактних латентних представлень даних та дослідження структури прихованого простору за допомогою методів зниження розмірності (t-SNE/UMAP). Особлива увага приділяється використанню помилки реконструкції для вирішення задач кластеризації та виявлення аномалій.

Четверта лабораторна робота присвячена вивченню генеративних моделей, зокрема згорткових генеративно-змагальних мереж (DCGAN). У ході її виконання досліджується специфіка міні-максної гри між генератором і дискримінатором, відпрацьовуються методи стабілізації змагального навчання та оцінювання якості синтезованих зображень за допомогою візуального аналізу та кількісних метрик.

П'ята лабораторна робота присвячена поглибленому аналізу результатів та оцінюванню продуктивності моделей. У ході її виконання передбачено порівняння різних архітектур за комплексом метрик (Precision, Recall, F1-score), побудову ROC та PR-кривих, проведення порогового аналізу та калібрування ймовірностей. Особлива увага приділяється методу бутстреп для оцінювання статистичної значущості та стабільності отриманих результатів.

В результаті виконання лабораторних робіт здобувачі освіти мають набути практичних навичок формування та балансування репрезентативних наборів даних; самостійного проєктування та програмної реалізації нейромережевих архітектур у фреймворках TensorFlow/Keras або PyTorch; налаштування гіперпараметрів процесу навчання; візуалізації динаміки тренування; інтерпретації помилок класифікації та вибору оптимальних моделей для вирішення прикладних завдань інженерії програмного забезпечення.

Після проходження дисципліни здобувач освітнього рівня «Магістр» повинен вміти проєктувати моделі глибокого навчання з використанням сучасних фреймворків, проводити обчислювальні експерименти з їх тренування, аналізувати отримані результати, комплексно оцінювати якість і стабільність моделей та робити обґрунтовані висновки щодо їх застосування для розв'язання прикладних задач.

Лабораторна робота №1

Основи роботи з даними для глибокого навчання

Мета – отримання практичних навичок роботи з даними для глибокого навчання, зокрема: збору, попередньої обробки та аугментації зображень, а також ефективного формування великих наборів даних (понад 50 тис. зображень) з невеликої вихідної колекції.

Теоретичні відомості

У глибокому навчанні датасети зображень є фундаментальною основою для розроблення та тренування нейронних мереж, призначених у завданнях комп'ютерного зору. Якість та різноманітність цих наборів даних безпосередньо впливають на точність і надійність моделей, які вчать обробляти та аналізувати візуальну інформацію.

Датасети зображень застосовуються для широкого спектра завдань, серед яких:

- **Класифікація** – розподіл множини об'єктів на попередньо визначені групи (класи) за певними ознаками.
- **Детекція** – локалізація та ідентифікація об'єктів на зображенні з визначенням їхніх просторових меж.
- **Сегментація** – попиксельне маркування зображення для виділення окремих об'єктів або семантичних областей.
- **Генерація** – створення нових візуальних даних на основі навчених патернів.

Успішне застосування глибокого навчання в цих завданнях сприяло прогресу в багатьох критично важливих галузях, зокрема в безпілотному керуванні, медичній діагностиці, системах безпеки та робототехніці.

Якість моделей глибокого навчання значною мірою залежить від правильності підготовки датасету.

Основні етапи підготовки даних:

➤ **Збір.**

Джерелами можуть бути відкриті датасети (наприклад, [Kaggle](#), [ImageNet](#), [GoogleAPIs](#), [Roboflow](#)), різноманітні стокові ресурси (наприклад, [Unsplash](#), [Pixabay](#), [Freepik](#)) і власні колекції (наприклад, зроблені фото, скріншоти).

Важливо: набір має відображати різноманіття реальних умов (освітлення, масштаб, кут огляду).

➤ **Очищення.**

Процес передбачає видалення дублікатів, усунення пошкоджених зразків та стандартизацію форматів і розмірів (наприклад, PNG або JPEG, розмір $N \times M$ px).

Важливо: При зміні розміру зберігати співвідношення сторін або використовувати обрізання/відступи, щоб уникнути деформації.

➤ **Анотування (маркування).**

Для задач класифікації достатньо підписати, якому класу належить зображення.

У задачах детекції, окрім класу, потрібно визначити координати об'єкта. У задачах сегментації кожен піксель, що належить до об'єкта, позначається піксельною маскою.

➤ **Розподіл на вибірки (train/val/test).**

Традиційно дані ділять на три частини:

- тренувальна вибірка (train $\approx$ 70-80%) – для навчання моделі;
- валідаційна (validation $\approx$ 10-15%) – для підбору гіперпараметрів;
- тестова (test $\approx$ 10-15%) – для фінальної оцінки ефективності моделі.

➤ **Аугментація.**

Це штучне розширення датасету шляхом геометричних перетворень (повороти, зсуви, віддзеркалення), фотометричних модифікацій (корекція яскравості, контрасту), зашумлення та спеціалізованих технік (Cutout, MixUp, CutMix). Аугментація дає змогу зробити модель стійкою до варіацій у реальних даних.

➤ **Балансування класів.**

Методи вирівнювання включають штучне збільшення кількості прикладів для рідкісних класів (oversampling), зменшення кількості прикладів для

надлишкових класів (undersampling), синтетичну генерацію нових зразків для менш представлених класів та балансування на рівні навчання моделі (class weighting, focal loss). На практиці часто комбінують кілька підходів, щоб досягти більш стабільних результатів.

Зверніть увагу: аугментацію та будь-яке штучне балансування (oversampling) як правило застосовують тільки до тренувальної вибірки; валідаційна і тестова вибірки мають залишатися незалежними і репрезентативними для правдивої оцінки узагальнюючої здатності навченої моделі.

Бібліотека PIL

Python Imaging Library (Pillow, PIL) – це потужна бібліотека Python для роботи з растровими зображеннями. Вона надає можливості для відкриття, обробки, модифікації, збереження та аугментації зображень.

Модуль **Image** відповідає за створення та базову роботу з об'єктами зображень. Основні методи та атрибути:

- `Image.open(path)` – відкриває зображення з файлу за заданим шляхом `path` і повертає об'єкт `Image`. Підтримує JPEG, PNG, BMP, GIF та інші формати.
- `Image.fromarray(ndarray)` – створює об'єкт `Image` із масиву NumPy `ndarray`. Використовується для конвертації обчислених або змінених зображень у формат, придатний для збереження або відображення.
- `transpose(method)` – метод для перетворення зображення, наприклад, горизонтального або вертикального віддзеркалення:
 - `Image.FLIP_LEFT_RIGHT` – горизонтальне віддзеркалення зображення.
 - `Image.FLIP_TOP_BOTTOM` – вертикальне віддзеркалення зображення.
 - `rotate(angle, expand=True, fillcolor=None)` – обертання зображення на кут `angle` (в градусах). Параметр `expand=True` дозволяє змінити розмір полотна, щоб повністю вмістити повернуте зображення, `fillcolor` задає колір фону для порожніх областей.

Модуль **ImageEnhance** використовується для зміни властивостей зображення, таких як яскравість, контрастність, різкість та насиченість кольору. Основні класи та методи:

- `ImageEnhance.Brightness(image)` – клас для зміни яскравості зображення `image`. Метод класу `enhance(factor)` – якщо `factor = 1.0`, то залишає яскравість без змін; якщо `factor` менше `1.0` – затемнює; якщо більше `1.0` – робить яскравіше.
- `ImageEnhance.Contrast(image)` – клас для зміни контрастності зображення `image`. Метод класу `enhance(factor)` – якщо `factor = 1.0`, то залишає контрастність без змін; якщо `factor` менше `1.0` – зменшує контраст; якщо більше `1.0` – підсилює контраст.

Модуль **ImageFilter** надає набори фільтрів для обробки зображень, таких як розмиття, контуризація, згладжування. Наприклад, `ImageFilter.GaussianBlur(radius=2)` – фільтр гаусового розмиття з заданим радіусом `radius`. Використовується для згладжування або зменшення шуму на зображенні.

Приклад аугментації зображень

Розглянемо базові принципи аугментації даних з використанням бібліотеки PIL. Створіть маленький набір даних (3-4 папки по 4-5 невеликих зображень), який надалі будете використовувати для первинного налагодження коду. Шлях до цього набору збережіть у змінну `input_folder`. Шлях до директорії для збереження перетворених зображень запишіть у змінну `output_folder`. Виберемо для подальших перетворень вихідний розмір зображень `224×224` px.

```
import os
import random
import numpy as np
from PIL import Image, ImageEnhance, ImageFilter

input_folder = r"C:\Python\input_images"
output_folder = r"C:\Python\output_images"
valid_extensions = (".jpg", ".jpeg", ".png", ".bmp")
IMG_SIZE = (224, 224)
```

Набір зображень
Результат перетворень
Допустимі формати зображень
Розмір після аугментації

Створимо масив з методами аугментації, які включають горизонтальне віддзеркалення, обертання на випадковий кут, зміну яскравості та контрасту, додавання гаусівського шуму і розмиття:

```
def augment_image(img):
    augmentations = {
        'h_flip': img.transpose(Image.FLIP_LEFT_RIGHT),
        'rotate': img.rotate(random.uniform(-30, 30), expand=True,
                             fillcolor=(255, 255, 255)),
        'bright': ImageEnhance.Contrast(ImageEnhance.Brightness(img).enhance(
            random.uniform(0.5, 2.0))).enhance(random.uniform(0.5, 2.0)),
        'noise': Image.fromarray(np.clip(np.array(img) + np.random.normal(
            0, 25, np.array(img).shape), 0, 255).astype(np.uint8)),
        'jitter': Image.fromarray(np.array(img.convert('HSV').convert('RGB'))),
        'blur': img.filter(ImageFilter.GaussianBlur(random.uniform(0.5, 1)))
    }
    return augmentations
```

Застосуємо кожен з цих методів по черзі до всіх файлів у вхідній директорії, попередньо перетворивши їх до єдиного розміру (IMG_SIZE):

```
for root, dirs, files in os.walk(input_folder):
    for f in files:
        if f.lower().endswith(valid_extensions):
            try:
                img_path = os.path.join(root, f)
                rel_path = os.path.relpath(root, input_folder)
                out_dir = os.path.join(output_folder, rel_path) if rel_path != '.' else output_folder
                os.makedirs(out_dir, exist_ok=True)

                img = Image.open(img_path).convert('RGB')
                img = img.resize(IMG_SIZE)
                img.save(os.path.join(out_dir, f))

                # Аугментації
                for name, aug_img in augment_image(img).items():
                    name_parts = f.rsplit('.', 1)
                    name_aug = f"{name_parts[0]}_{name}.{name_parts[1]}"
                    aug_img.save(os.path.join(out_dir, name_aug))

            except Exception as e:
                print(f"Error {f}: {e}")
```

Кожне перетворене зображення зберігається в окремий файл, назва якого містить інформацію про застосований метод. Таким чином, для кожного оригінального зображення генерується шість додаткових варіантів (по одному для кожного типу аугментації), що призводить до семикратного збільшення обсягу початкової вибірки.

Для досягнення більшої варіативності даних доцільним є багаторазове використання методів з новими випадковими параметрами та випадкове комбінування декількох методів аугментацій для одного зображення. Такий

підхід забезпечить створення значно більшої кількості унікальних варіантів зображень.

Завдання на лабораторну роботу

1. Сформулювати початковий набір даних. Можна використати готовий датасет з відкритих джерел, зібрати з веб-ресурсів або створити власний набір зображень. **Вимоги:** набір даних має включати **мінімум 3 різні класи** з кількістю понад 1000 цифрових зображень у кожному класі.
2. Виявити та видалити пошкоджені файли.
3. Знайти та усунути дублікати (на основі хешування та візуальної подібності). Зверніть увагу: якщо один і той самий файл зустрічається у різних класах, то це помилка маркування (анотування), яка потребує ручної перевірки та виправлення.
4. Уніфікувати формат файлів (наприклад, PNG або JPEG).
5. Привести всі зображення до єдиного розміру (наприклад, 224×224 px або 256×256 px).
6. Розділити набір на train/validation/test у співвідношенні 80/10/10. Проаналізувати розподіл класів у кожній вибірці. Переконайтеся, що всі класи представлені в усіх трьох вибірках.
7. Застосувати методи аугментації до **тренувальної** вибірки для досягнення обсягу не менше 10 тис. зображень*. Реалізувати мінімум 6 різних методів аугментації.
Обов'язкові (4 методи): горизонтальне/вертикальне** віддзеркалення, поворот на випадковий кут, зміна яскравості/контрасту, кадрування (random crop) з наступним масштабуванням.
На вибір (2 і більше методів з переліку): гаусівський шум, зсув кольорів (color jittering), еластичні деформації, cutout/random erasing, blur ефекти.
8. Проаналізувати кількість зображень у кожному класі тренувальної вибірки до і після аугментації.

**Технічні рекомендації: спочатку налагодьте код на невеликому наборі (3-4 класи по 4-5 зображень); обов'язково візуально перевірте результати аугментації; при роботі з повним набором даних використовуйте багатопоточність для прискорення оброблення файлів.*

*** Примітка: Вертикальне віддзеркалення є менш універсальним і може бути неприродним для багатьох типів зображень, тому його слід застосовувати обережно, зважаючи на специфіку набору даних.*

Зміст звіту

1. Титульний аркуш;
2. Опис початкового набору даних: вкажіть джерело даних, загальну кількість зображень, кількість класів і кількість екземплярів у кожному з них;
3. Результати процесу очищення: кількість виявлених і видалених дублікатів та пошкоджених файлів, загальна кількість зображень та кількість екземплярів у кожному класі після очищення;
4. Статистика форматів файлів та розмірів зображень до і після уніфікації;
5. Блок-схема алгоритму аугментації датасету;
6. Код програми (може бути оформлений як додаток або прикріплений як окремий текстовий файл чи архів, якщо файлів багато);
7. Перелік використаних методів аугментації (перерахуйте та коротко опишіть використані методи);
8. Приклади аугментованих зображень (виберіть по одному зображенню з кожного класу, покажіть оригінальне зображення та його варіанти після застосування кожного з методів аугментації);
9. Статистика сформованого датасету (до і після аугментації);
10. Висновки.

Питання для самоконтролю

1. Опишіть основні етапи підготування набору даних для моделі глибокого навчання в задачах комп'ютерного зору.

2. Для чого потрібно проводити очищення даних (видалення дублікатів і пошкоджених файлів)?
3. У чому полягає основна відмінність між тренувальною, валідаційною та тестовою вибірками?
4. Чому співвідношення між тренувальною, валідаційною та тестовою вибірками не є фіксованим і може змінюватися залежно від загального обсягу набору даних?
5. Як збільшення обсягу датасету впливає на узагальнювальну здатність моделі?
6. Які наслідки для оцінки точності моделі можуть виникнути, якщо одне й те саме зображення потрапляє як у тренувальну, так і у валідаційну чи тестову вибірку?
7. Наведіть приклади основних геометричних методів аугментації та поясніть, як вони впливають на стійкість моделі до змін у просторовому розташуванні об'єктів.
8. Наведіть приклад, коли вертикальне віддзеркалення зображення може бути доречним, і коли воно може бути шкідливим для навчання моделі.
9. Наведіть приклади основних фотометричних методів аугментації та поясніть, як вони впливають на здатність моделі розпізнавати об'єкти за різних умов освітлення.
10. У чому полягає небезпека дисбалансу класів для навчання нейронної мережі?

Література: [1, С. 171–209], [2, С. 225–291], [5, С. 57–97], [10, С. 537–631].

Лабораторна робота №2

Навчання з учителем

Мета – отримання практичних навичок побудови та навчання згорткових нейронних мереж для класифікації зображень, дослідження впливу регуляризаційних технік на якість і стійкість моделі.

Теоретичні відомості

Згорткові нейронні мережі (Convolutional Neural Networks, CNN) ґрунтуються на принципі локальних зв'язків: кожен нейрон з'єднаний лише з обмеженою підмножиною вхідних даних.

Архітектура CNN складається з послідовності шарів, кожен з яких виконує певну функцію:

- **Згорткові шари** (Convolutional Layers) виявляють локальні ознаки на зображеннях (наприклад, контури, кути, текстури). Фільтри (ядра згортки) навчаються автоматично. Кожен фільтр створює карту ознак (feature map).
- **Функції активації** (Activation Layers) вносять нелінійність, надаючи мережі можливість розпізнавати складні залежності.
- **Шари пулінгу** (Pooling Layers, MaxPooling2D, AveragePooling2D) зменшують просторові розміри даних, роблять модель менш чутливою до зсувів і зменшують кількість параметрів.
- **Шари нормалізації** (Batch Normalization) стабілізують і прискорюють навчання, нормалізуючи активації у межах кожного міні-пакету. Зазвичай BN розташовують між згортковим шаром та функцією активації.
- **Dropout** є методом регуляризації, який випадково «вимикає» нейрони під час навчання, щоб зменшити перенавчання.
- **Операція сплющування** (Flattening) перетворює 2D-виходи згорткових шарів у 1D-вектор, який може бути переданий до повнозв'язних шарів.
- **Повнозв'язні шари** (Fully Connected Layers, Dense) виконують інтеграцію всіх ознак і формують підсумкове рішення (ймовірності належності до класів).

- **Global Average Pooling (GAP)** – використовується як альтернатива Flattening та повнозв'язним шарам вкінці CNN. Розміщується безпосередньо перед вихідним шаром. Його роль полягає у зниженні розмірності ознак до кількості класів.

- **Вихідний шар (Output Layer)** має стільки нейронів, скільки класів у задачі. Слугує для формування фінального прогнозу моделі. У задачах класифікації вихідні значення перетворюються функцією активації на кінцеві ймовірності належності зображень до кожного класу.

Функції Активації

- **ReLU (Rectified Linear Unit)** – найпоширеніша функція, застосовується після згорткових/повнозв'язних шарів. Вносить нелінійність і надає мережі можливість вивчати складні залежності.

- **Sigmoid** використовується у вихідному шарі для бінарної класифікації. Стискає значення в діапазон $[0,1]$, де вихід інтерпретується як ймовірність належності до позитивного класу.

- **Softmax** використовується у вихідному шарі для багатокласової класифікації. Перетворює вектор вихідних значень на ймовірнісний розподіл між класами. Сума ймовірностей усіх класів дорівнює 1. Це дає змогу моделі вибирати один клас із кількох можливих.

Основні навчальні компоненти

Функція втрат (Loss function) вимірює різницю між передбаченнями моделі та правильними мітками. Для багатокласової класифікації зазвичай використовують *categorical crossentropy* (якщо мітки закодовані one-hot, тобто для кожної мітки правильний клас – 1, а всі інші класи – 0) або *sparse categorical crossentropy* (якщо мітки – цілі числа, які відповідають індексам класів). Вибір правильної функції втрат критично впливає на здатність моделі вчитися.

Оптимізатор – алгоритм, який змінює ваги моделі, щоб мінімізувати функцію втрат. Основні варіанти, які застосовуються на практиці:

- **SGD (Stochastic Gradient Descent)** – класичний метод, добре працює з налаштуванням learning rate.

- **Adam** – найчастіше використовується в перших експериментах, адаптує швидкість навчання для кожного параметра.

- **RMSprop** – стабільний вибір для роботи з зображеннями.

Гіперпараметри навчання – це параметри, які не навчаються автоматично, а задаються дослідником:

- **Batch size** – скільки зображень обробляється одночасно. Зазвичай обирають у діапазоні від 16 до 128.

- **Кількість епох (epochs)** – скільки разів модель пройде через увесь тренувальний набір.

- **Learning rate** – швидкість навчання; занадто велике значення призведе до нестабільності, занадто мале – до повільної збіжності.

- **Dropout rate** – частка вимкнених нейронів під час навчання.

- **Розташування Batch Normalization** – зазвичай додається після згорткового шару та перед функцією активації.

Метрики оцінювання. Для контролю навчання зазвичай відстежують *accuracy*. Для аналізу на тестовій вибірці додатково потрібні *precision*, *recall*, *F1-score* і *confusion matrix*.

Callbacks (зворотні виклики). У TensorFlow/Keras і PyTorch можна використовувати спеціальні інструменти:

- **EarlyStopping** – зупиняє навчання, якщо метрика не покращується.

- **ModelCheckpoint** – зберігає найкращу версію моделі під час навчання.

Порівняння фреймворків TensorFlow/Keras та PyTorch

TensorFlow/Keras має високорівневий API Keras, що забезпечує швидке прототипування та конструювання моделей як послідовного, так і функціонального стилю. Фреймворк є зручним для початківців завдяки вбудованим засобам візуалізації та моніторингу (об'єкт History, інтеграція з

TensorBoard). Підходить для навчальних проєктів та швидкої імплементації в продакшн (завдяки інструментам розгортання).

PyTorch орієнтований на дослідників і забезпечує більшу гнучкість у налаштуванні моделей. Фреймворк використовує імперативний синтаксис з динамічними обчислювальними графами (Eager Execution), що робить його логіку більш схожою на стандартний Python та спрощує відлагоджування. PyTorch надає більший контроль над процесом навчання через явну імплементацію навчального циклу (train/eval). Це особливо важливо для передових досліджень, де потрібна модифікація базових операцій.

Завдання на лабораторну роботу

1. Побудувати базову CNN-модель класифікації зображень за допомогою бібліотек та фреймворків глибокого навчання (наприклад, TensorFlow/Keras або PyTorch). Використати просту архітектуру на основі послідовного підходу з кількох згорткових та повнозв'язних шарів. Підібрати розмірність вихідного шару відповідно до кількості класів.
2. Налаштувати процес навчання: визначити функцію втрат (loss function), обрати оптимізатор, встановити основні гіперпараметри (batch size, epoch, learning rate).
3. Провести навчання моделі на тренувальній вибірці з контролем якості за валідаційною вибіркою. Використовувати датасет, сформований у ЛР №1 (train/validation/test).
4. Вивести проміжні результати (значення loss та accuracy) після кожної епохи.
5. Візуалізувати процес навчання у вигляді графіків: залежність accuracy від кількості епох, залежність loss від кількості епох.
6. Оцінити якість моделі на тестовій множині. Навести основні метрики (accuracy, precision, recall, F1-score) та матрицю плутанини (confusion matrix).
7. Подивитися приклади зображень, які модель класифікує неправильно (з матриці плутанини вибрати кілька випадків). Вивести на екран ці кілька

зображень разом із передбаченими ймовірностями класів. Спробувати визначити ймовірні причини виникнення помилок.

8. Модифікувати базову модель, додавши шар Dropout. Підібрати оптимальний коефіцієнт відсікання (перевірити кілька різних значень). Порівняти результати з базовою моделлю.
9. Модифікувати базову модель, додавши шар Batch Normalization (BN). Підібрати оптимальне розміщення BN в архітектурі. Порівняти результати з базовою моделлю.
10. Зберегти навчену модель (найкращу з версій) на диск та завантажити її для повторного використання. Переконайтеся, що після завантаження модель зберігає попередні параметри та відтворює ті самі результати на тестовій вибірці.
11. Зробити висновки: проаналізувати результати навчання різних версій моделі (Базова, з Dropout, з Batch Normalization), визначити найкращу модель на основі метрик тестової вибірки (F1-score та Accuracy), обґрунтувати вибір оптимального коефіцієнта Dropout та розміщення Batch Normalization.

**Технічні рекомендації: спочатку налагодьте код на невеликому наборі зображень; при роботі з повним набором даних використовуйте багатопоточність для прискорення оброблення файлів.*

Зміст звіту

1. Титульний аркуш;
2. Схема та опис архітектури базової моделі (кількість шарів, типи шарів, параметри);
3. Обґрунтування вибору функції втрат, оптимізатора та гіперпараметрів;
4. Графіки процесу навчання (accuracy та loss залежно від кількості епох);
5. Оцінка моделі на тестовій вибірці: accuracy, precision, recall, F1-score та матриця плутанини;
6. Приклади зображень, які модель класифікувала неправильно, з зазначенням

істинної мітки, передбачення, ймовірності класів та аналіз ймовірних причин помилок;

7. Результати порівняння моделі з Dropout та базової моделі;
8. Результати порівняння моделі з Batch Normalization та базової моделі;
9. Оцінка моделі після збереження та повторного завантаження на тестовій вибірці: accuracy, precision, recall, F1-score та матриця плутанини;
10. Висновки;
11. Код програми (прикріплений як окремий текстовий файл чи архів, якщо файлів багато).

Питання для самоконтролю

1. Які основні елементи має архітектура згорткової нейронної мережі для задачі класифікації зображень?
2. Поясніть, у чому полягає роль операції Pooling у згорткових мережах?
3. Як визначається кількість нейронів у вихідному шарі згорткової мережі для класифікації?
4. Які основні гіперпараметри впливають на навчання згорткової мережі та як вони задаються?
5. Що таке функція втрат (loss function) і яку роль вона відіграє у процесі навчання нейронних мереж?
6. Чим відрізняється функція активації softmax від sigmoid у задачі класифікації?
7. Яку проблему допомагає вирішувати Dropout у нейронних мережах?
8. Чи застосовується Dropout під час прогнозування нових даних уже навченою моделлю?
9. Яка роль Batch Normalization (BN) у нейронній мережі?
10. Яке практичне значення має збереження та повторне завантаження моделі у прикладних задачах?

Література: [3, Chapter 3, 8], [4, С. 238–279], [7, С. 17–40], [9, С. 189–196]

Лабораторна робота №3

Навчання без учителя

Мета – отримання практичних навичок побудови симетричних автоенкодерів, дослідження латентного простору і застосування моделі для задач реконструкції зображень, кластеризації та виявлення аномалій.

Теоретичні відомості

Навчання без учителя (Unsupervised Learning) – це підхід, у якому модель намагається знайти закономірності у необроблених, неанотованих даних. На відміну від навчання з учителем, де для кожного прикладу задана правильна відповідь, у навчанні без вчителя метою є виявлення прихованої структури, закономірностей або взаємозв'язків у даних. Основними завданнями є кластеризація, зниження розмірності, виявлення аномалій, побудова латентних представлень.

Одним із ключових інструментів глибокого навчання без учителя є **автоенкодери (Autoencoders)** – нейронні мережі, які навчаються відтворювати свої вхідні дані на виході. Метою є не ідеальне копіювання, а навчання інформативного та компактного представлення (латентного коду), що зберігає суттєві риси об'єкта.

Автоенкодер складається з двох основних частин:

- **Енкодер (Encoder)** – нейронна мережа, яка перетворює вхідне зображення у компактний вектор фіксованої довжини (латентне представлення). Енкодер поступово зменшує просторову розмірність, залишаючи лише найінформативніші ознаки.
- **Декодер (Decoder)** – нейронна мережа, яка намагається відновити вхідні дані з латентного простору до початкового розміру.

Архітектура автоенкодера зазвичай базується на принципі дзеркальної **симетрії**: структура декодера у зворотному порядку повторює структуру енкодера. Кожному шару, що зменшує розмірність, відповідає аналогічний шар

для її відновлення. Практично це спрощує проєктування та стабілізує навчання, оскільки забезпечує логічну узгодженість між процесами стиснення даних та їхньою подальшою реконструкцією.

У контексті комп'ютерного зору зазвичай використовують **згорткові автоенкодер** (Convolutional Autoencoders, CAE), які оперують не окремими пікселями, а локальними областями зображень. Енкодер складається зі стеку шарів Conv2D та MaxPooling (або Conv2D зі $\text{stride} > 1$). На кожному кроці він зменшує просторові розміри (ширину та висоту) і, як правило, збільшує глибину (кількість каналів/фільтрів). Це змушує мережу вивчати ієрархію ознак – від простих контурів до складних текстур. Для відновлення даних у декодері застосовують два основні підходи:

- **UpSampling + Conv2D**: зображення збільшується простою інтерполяцією, а згортка лише «уточнює» контент. Цей метод є простим, стабільним і не створює артефактів. Але UpSampling2D не навчається масштабуванню, оскільки не має параметрів.
- **Conv2DTranspose** (Транспонована згортка): модель сама визначає оптимальний спосіб розгортання простору. Хоча цей підхід гнучкіший, він вимагає ретельного підбору параметрів `stride` та `kernel_size`, щоб уникнути появи візуальних артефактів «шахової дошки» (checkerboard patterns).

Латентний простір – це внутрішнє представлення даних, у якому об'єкти з подібними характеристиками розташовані близько один до одного. Аналізуючи структуру латентного простору, можна виявляти приховані кластери, що відповідають схожим типам зображень, навіть без міток.

Латентний простір має значно меншу розмірність, ніж вхід. Ця структурна перешкода («пляшкове горло», bottleneck) змушує модель відкидати надлишкову інформацію та шум, зберігаючи лише найсуттєвіші закономірності, необхідні для відновлення. Розмірність латентного простору суттєво впливає на якість реконструкції:

- надто мала розмірність призводить до втрати важливої інформації та нездатності моделі відновити складні деталі (underfitting);

- надто велика – до перенавчання і втрати узагальнювальної здатності (overfitting), оскільки відбувається тривіальне копіювання входу на вихід.

Помилка реконструкції (reconstruction error) обчислюється через функцію втрат (найчастіше середньоквадратичну помилку – MSE) і використовується для оновлення ваг мережі методом градієнтного спуску.

Щоб проаналізувати латентні вектори, їх часто зменшують до двох вимірів за допомогою методів **t-SNE** (t-distributed Stochastic Neighbor Embedding) або **UMAP** (Uniform Manifold Approximation and Projection). Це дає змогу побачити, як об'єкти групуються в просторі ознак, та оцінити семантичну близькість кластерів.

Після навчання енкодер можна використовувати як механізм витягання ознак (feature extractor). Отримані вектори подаються на вхід алгоритмів кластеризації, щоб згрупувати зображення за схожими ознаками. У цьому разі модель автоматично виявляє логічні групи без участі користувача.

Якщо автоенкодер навчений на «нормальних» зображеннях, то для нових або аномальних прикладів він не зможе точно відтворити вхід, а отже — помилка реконструкції буде суттєво вищою. Цю властивість використовують для виявлення аномалій, визначаючи поріг помилки, за яким приклади вважаються «аномальними».

Автоенкодери широко застосовуються у задачах комп'ютерного зору для стиснення та відновлення зображень, видалення шуму, виявлення дефектів і аномалій у даних, а також для попереднього навчання глибоких моделей без використання міток.

Завдання на лабораторну роботу

1. Реалізувати архітектуру симетричного автоенкодера, у якій декодер є дзеркальним відображенням енкодера.
2. Використати для навчання моделі датасет, сформований у ЛР №1, але без використання міток класів.

3. Вибрати функцію втрат, яка підходить для задачі реконструкції зображень. Обґрунтувати вибір. З її допомогою проаналізувати якість відновлення. Порівняти оригінали та реконструкції візуально для декількох екземплярів.
4. Дослідити вплив розмірності латентного простору на якість відновлення. Врахуйте, що за принципом «вузького місця» (bottleneck) розмір латентного вектора має бути значно меншим, ніж розмірність вхідних даних. Для кожної розмірності візуалізувати по 5 випадкових прикладів реконструкції з тестової вибірки. Визначити розмір латентного простору, за якого візуальні відмінності між оригінальними зображеннями та їхніми реконструкціями мінімальні при максимальному стисненні інформації.
5. Побудувати графік залежності середньої помилки реконструкції на тестовій вибірці від розміру латентного вектора. Проаналізувати графік та визначити розмірність, яка є «точкою перегину» – після якої якість реконструкції перестає суттєво покращуватися.
6. Використати t-SNE або UMAP для проєкції латентних векторів у 2D. Візуалізувати кластери, що утворюються в латентному просторі.
7. Виконати кластеризацію векторів ознак, отриманих після енкодера. Вибрати кількість кластерів рівною кількості класів у наборі даних. Візуалізувати по 5 представників кожного кластера. Візуально оцінити семантичну схожість зразків у межах кожної групи.
8. Використати помилку реконструкції автоенкодера як критерій для виявлення аномалій (нових, не бачених раніше класів об'єктів). Для цього використати зображення класів, не представлених у навчальній вибірці, наприклад з іншого датасету. Визначити поріг аномалії на основі 95-го перцентиліа помилок реконструкції на тестовій вибірці. Продемонструвати приклади реконструкцій зображень, які класифіковано як «аномальні», порівняно з нормальними зображеннями.
9. Зробити висновки щодо:
 - Залежності якості реконструкції зображень від розмірності латентного простору.

- Здатності автоенкодера вивчати семантично значущі ознаки для задач кластеризації без використання міток.
- Ефективності використання помилки реконструкції як критерію для виявлення аномалій.

**Технічні рекомендації: спочатку налагодьте код на невеликому наборі зображень; при роботі з повним набором даних використовуйте багатопоточність для прискорення оброблення файлів; для прискорення експериментів зберігайте на диск навчену модель енкодера.*

Зміст звіту

1. Титульний аркуш.
2. Схема та опис архітектури автоенкодера (кількість шарів енкодера та декодера із зазначенням їхніх параметрів).
3. Опис гіперпараметрів навчання та обґрунтування вибору функції втрат.
4. Приклади оригінальних зображень та їхні реконструкції, відновлені автоенкодером.
5. Графік залежності середньої помилки реконструкції на тестовій вибірці від розміру латентного вектора. Візуальні приклади реконструкцій для кожної досліджуваної розмірності (по 5 прикладів, оригінал та відновлення поруч). Коментар щодо оптимальної розмірності латентного простору (за візуальною оцінкою та за «точкою перегину»).
6. Візуалізація латентного простору: 2D-діаграма розсіювання, отримана за допомогою t-SNE або UMAP, з аналізом утворених кластерів.
7. Результати кластеризації в латентному просторі: візуалізація представників кожного кластера (по 5 зображень) з візуальною оцінкою та коментарем щодо семантичної схожості зразків у межах кластерів.
8. Визначення порогу аномалії (95-й перцентиль). Приклади зображень із «нормальних» класів та їхні помилки реконструкції. Приклади «аномальних» зображень, їхні невдалі реконструкції та значення помилок, які перевищують поріг.

9. Висновки.
10. Код програми (прикріплений як окремий текстовий файл чи архів, якщо файлів багато).

Питання для самоконтролю

1. У чому полягає основна відмінність між навчанням з учителем і без учителя?
2. Які дві основні частини має архітектура автоенкодера та яка функція кожної з них?
3. Яку роль у декодері виконують транспоновані згорткові шари і чим їхня функція відрізняється від згорткових шарів в енкодері?
4. Що таке латентний простір і яку роль він відіграє у автоенкодерах?
5. Як функція втрат впливає на процес навчання автоенкодера?
6. Як зміна розмірності латентного простору впливає на якість реконструкції зображень? Поясніть компроміс між стисненням даних та втратою інформації.
7. З якою метою застосовують алгоритми t-SNE або UMAP для візуалізації векторів, отриманих з енкодера?
8. Як вибирається поріг аномалії при використанні помилки реконструкції і що він означає?
9. Як можна інтерпретувати помилку реконструкції для виявлення аномалій?
10. Чому для задачі кластеризації часто ефективніше використовувати вектори з латентного простору, а не вихідні зображення?

Література: [1, С. 186–197], [3, Chapter 9], [6, С. 499–523], [8, С. 589–595].

Лабораторна робота №4

Генеративні моделі глибокого навчання

Мета – отримання практичних навичок побудови згорткових генеративно-змагальних мереж, дослідження процесу змагального навчання і застосування моделі для синтезу нових зображень.

Теоретичні відомості

Архітектура DCGAN

У класичній реалізації згорткової генеративно-змагальної мережі (**Deep Convolutional GAN, DCGAN**) обидві моделі побудовані на згорткових шарах:

➤ **Генератор (G):**

- Використовує транспоновані згортки (ConvTranspose2d) для послідовного збільшення просторової розмірності з латентного вектора до зображення;
- Функція активації: ReLU у прихованих шарах, Tanh на виході;
- Batch Normalization після кожного шару, крім першого та останнього.
- Вхід – латентний вектор $\mathbf{z}$, вихід – зображення розміром $H \times W \times C$.

➤ **Дискримінатор (D):**

- Побудований на класичних згорткових шарах, які зменшують розмірність зображення до скаляра (ймовірності);
- Функція активації: LeakyReLU у прихованих шарах та Sigmoid на виході;
- Batch Normalization після кожного згорткового шару, крім першого (вхідного) і останнього (класифікаційного);
- Вхід – цифрове зображення розміром $H \times W \times C$, а вихід – ймовірність того, що зображення реальне (від 0 до 1).

Попередня обробка даних

Для коректного навчання моделі необхідно узгодити діапазон вхідних даних з виходом Генератора. Оскільки Генератор використовує активацію Tanh, його вихід знаходиться в діапазоні $[-1, 1]$. Відповідно, всі зображення

навчального набору мають бути **нормалізовані** до діапазону $[-1, 1]$ перед подачею до Дискримінатора.

Такий підхід забезпечує стабільніше навчання та сприяє формуванню гладких градієнтів, що особливо важливо у роботі з реалістичними зображеннями.

Функція втрат

Теоретична міні-максна гра на практиці реалізується за допомогою Бінарної Перехресної Ентропії (Binary Cross Entropy, **BCE**), оскільки оптимізатори (як Adam) завжди виконують мінімізацію функції втрат (Loss). Оптимізація відбувається у два етапи, і кожна мережа має власну функцію втрат, яку вона мінімізує.

➤ Функція втрат Дискримінатора:

$$\mathcal{L}_D = -\log D(x) - \log(1 - D(G(\mathbf{z}))), \quad (1)$$

де x – реальне зображення з датасету, $\mathbf{z}$ – випадковий латентний вектор, $G(\mathbf{z})$ – згенероване зображення, $D(x)$ – передбачена ймовірність того, що x реальне, $D(G(\mathbf{z}))$ – передбачена ймовірність того, що $G(\mathbf{z})$ реальне.

Завдання Дискримінатора – навчитися правильно класифікувати реальні та згенеровані зразки. Його функція втрат складається з суми двох BCE-втрат:

- Перший доданок $(-\log D(x))$ – це BCE-втрати для реальних зображень x , де цільова мітка – 1 (справжній).
- Другий доданок $(-\log(1 - D(G(\mathbf{z}))))$ – це BCE-втрати для згенерованих зображень $G(\mathbf{z})$, де цільова мітка – 0 (підробка).

Дискримінатор мінімізує $\mathcal{L}_D$, що еквівалентно правильній класифікації обох типів зразків.

➤ Функція втрат Генератора:

$$\mathcal{L}_G = -\log D(G(\mathbf{z})). \quad (2)$$

Генератор намагається мінімізувати $\mathcal{L}_G$, тобто максимізувати ймовірність того, що Дискримінатор класифікує згенеровані зображення як реальні.

Слід зазначити, що це так звана «не-насичувана» (non-saturating) функція втрат. Хоча теоретична гра вимагала б від Генератора мінімізувати $\log(1 - D(G(\mathbf{z})))$, ця функція має зникаючі градієнти на початку навчання, коли Дискримінатор легко розпізнає підробки, $D(G(\mathbf{z})) \approx 0$. Тому на практиці Генератор мінімізує $(-\log D(G(\mathbf{z})))$, що еквівалентно навчанню синтезувати зразки, які Дискримінатор класифікує з міткою 1 (справжній).

Процес навчання

Під час навчання оновлення ваг здійснюється по чергово (співвідношення 1:1):

- Спочатку тренується Дискримінатор, який бачить як реальні, так і згенеровані зразки.
- Потім тренується Генератор, який намагається покращити свої результати так, щоб Дискримінатор почав помилятися.

Цей процес повторюється на кожній ітерації (batch) протягом усіх епох навчання.

Гіперпараметри навчання

На якість генерації сильно впливають такі гіперпараметри:

- розмір латентного вектора (зазвичай, 64-128 елементів),
- швидкість навчання (зазвичай 0.0001-0.0002),
- тип оптимізатора (зазвичай Adam з $\beta_1 = 0.5$, $\beta_2 = 0.999$),
- параметр α у функції LeakyReLU (зазвичай 0.01–0.2).

Оцінювання якості генерації

Якість синтезованих зображень можна оцінити двома способами:

- Візуально – через порівняння реальних і згенерованих прикладів.
- Кількісно – за допомогою метрик:
- Inception Score (IS) – вимірює різноманітність і чіткість зображень;

- Fréchet Inception Distance (**FID**) – оцінює відстань між розподілами ознак реальних і синтетичних зображень, вилучених із попередньо натренованої мережі.

Проблеми навчання

➤ **Дисбаланс** між G та D.

Важливо дотримуватись балансу між Генератором і Дискримінатором. Якщо один з них тренується швидше, модель стає нестабільною.

Якщо D навчається надто швидко, то втрати G не зменшуються або зростають, градієнти для G зникають (D завжди впевнено класифікує), згенеровані зображення не покращуються.

Якщо G навчається надто швидко, то втрати D прямують до нуля, D завжди помиляється, втрачається навчальний сигнал, можлива генерація нереалістичних артефактів.

Ознаки балансу: обидві втрати коливаються в певному діапазоні, але не прямують до 0 чи ∞ .

➤ **«Колапс мод»** (Mode Collapse).

Це типова проблема, коли Генератор починає створювати майже ідентичні зразки, втрачаючи різноманітність генерацій. Це можна помітити візуально – всі згенеровані зображення виглядають схоже, а на графіку t-SNE/UMAP згенеровані точки формують щільний кластер.

Особливості застосування Batch Normalization

Основна мета застосування Batch Normalization (BN) у генеративно-змагальних мережах полягає у стабілізації процесу навчання. BN нормалізує активації на виході кожного проміжного шару до середнього ≈ 0 та стандартного відхилення ≈ 1 , з додатковими навчальними параметрами γ (масштаб) та β (зсув), які дають змогу мережі адаптувати ступінь нормалізації.

Ефекти BN у контексті нестабільної міні-максної гри GAN:

- Сприяє стабільнішому поширенню градієнтів (запобігає їх «згасанню» або «вибуху») і робить навчання менш чутливим до ініціалізації ваг;
- Опосередковано підвищує різноманітність генерацій, зменшуючи ризик «колапсу мод» (збіжності Генератора до обмеженого набору зразків).

Проте BN повністю не усуває проблему «колапсу мод» – для цього застосовують додаткові техніки, такі як minibatch discrimination, unrolled GAN тощо.

У класичній архітектурі DCGAN чітко визначено шари, де Batch Normalization **не слід** використовувати:

- У вхідному шарі Генератора (проекція латентного вектора). Застосування BN може обмежити здатність Генератора повноцінно використовувати весь діапазон латентного простору. Перший шар має самостійно навчитися перетворювати стандартний нормальний розподіл у потрібний розподіл ознак.
- У вихідному шарі Генератора (з активацією Tanh). Batch Normalization змінює масштаб вихідних активацій, порушуючи фіксований діапазон $[-1, 1]$, який забезпечує узгодженість між згенерованими та реальними нормалізованими зображеннями.
- У вхідному шарі Дискримінатора. Вхідні зображення (як реальні, так і згенеровані) вже нормалізовані до діапазону $[-1, 1]$. Додаткова нормалізація є зайвою і може спотворити вхідні сигнали, що призведе до втрати важливої інформації про структуру зображення.
- У вихідному шарі Дискримінатора (з активацією Sigmoid). Вихід цього шару – єдиний скаляр, який відображає ймовірність того, що зображення є реальним. Застосування BN на цьому рівні спотворює індивідуальні передбачення в межах батчу та порушує інтерпретацію виходу як ймовірності. Крім того, функція Sigmoid вже обмежує значення у діапазоні $[0, 1]$, тому додаткова нормалізація не має сенсу.

Завдання на лабораторну роботу

1. Реалізувати архітектуру згорткової генеративно-змагальної мережі. Використати згорткові та транспоновані згорткові шари для побудови Генератора і Дискримінатора. Додати Batch Normalization після кожного шару (крім першого та останнього). В Генераторі використати як функцію активації ReLU у прихованих шарах та Tanh у вихідному шарі. В Дискримінаторі використати LeakyReLU-активацію у прихованих шарах та Sigmoid активація у вихідному шарі.
2. Використати для навчання моделі датасет, сформований у ЛР №1, без міток класів. Нормалізувати зображення до діапазону $[-1; 1]$.
3. Провести початкове навчання моделі, по чергово оновлюючи ваги Генератора та Дискримінатора, і простежити динаміку зміни функції втрат для обох компонентів. Використати як функцію втрат Binary Cross-Entropy Loss. Логувати значення втрат для обох компонентів після кожної епохи. Візуалізувати динаміку зміни функції втрат на графіку.
4. Візуалізувати згенеровані зображення після певної кількості епох (наприклад, кожні 2-5 епох). Порівняти їхню якість та різноманітність у процесі навчання.
5. Дослідити вплив гіперпараметрів навчання (розмір латентного вектора, швидкість навчання, тип оптимізатора, варіації функції активації LeakyReLU з різними α) на якість синтезованих зображень.
6. Візуально оцінити різноманітність згенерованих зображень. Обчислити Inception Score (IS) або Fréchet Inception Distance (FID). Порівняти розподіл згенерованих та реальних зображень (наприклад, через t-SNE або UMAP).
7. Проаналізувати стабільність навчання моделі. Визначити, чи спостерігається «колапс моди» (mode collapse), коли всі зразки схожі та мала різноманітність генерацій.
8. Зробити висновки щодо стабільності процесу навчання згорткових генеративно-змагальних мереж, реалістичності та різноманітності синтезованих зображень.

**Технічні рекомендації: спочатку налагодьте код на невеликому наборі зображень; при роботі з повним набором даних використовуйте багатопоточність для прискорення оброблення файлів; для швидкого проведення експериментів збережіть на диск навчену модель Генератора.*

Зміст звіту

1. Титульний аркуш.
2. Структурна схема та опис архітектури згорткової генеративно-змагальної мережі (вказати типи шарів, розміри тензорів, функції активації та позиції шарів нормалізації);
3. Опис гіперпараметрів навчання (розмір латентного вектора, швидкість навчання, тип оптимізатора, функція втрат, розмір пакета та кількість епох навчання);
4. Графіки функцій втрат Генератора та Дискримінатора у процесі навчання (по епохах). Короткий коментар про поведінку втрат (збіжність чи розбіжність, баланс між компонентами та виявлені проблеми);
5. Приклади згенерованих зображень після різної кількості епох (наприклад, кожні 2-5 епох). Короткий коментар про зміни якості та різноманітності зображень у ході навчання;
6. Порівняльна таблиця (з IS або FID) для демонстрації впливу на якість синтезованих зображень різних гіперпараметрів навчання (розміру латентного вектора, швидкості навчання, типу оптимізатора, параметра α у функції LeakyReLU). Кілька різноманітних прикладів згенерованих зображень;
7. Порівняльна візуалізація реальних і згенерованих зображень у 2D-просторі (t-SNE або UMAP) з коротким аналізом розподілу точок;
8. Візуальні приклади проявів нестабільності навчання, які демонструють ознаки «колапсу моди» (якщо спостерігався) чи переваги одного з компонентів (домінування Генератора або Дискримінатора);
9. Висновки щодо стабільності навчання моделі, впливу гіперпараметрів на якість зображень, реалістичності та різноманітності синтезованих зразків.

10. Код програми (прикріплений як окремий текстовий файл чи архів, якщо файлів багато).

Питання для самоконтролю

1. У чому полягає принцип змагального навчання в генеративно-змагальних мережах?
2. Які ролі виконують Генератор і Дискримінатор у структурі GAN?
3. Для чого застосовується Batch Normalization у генераторі та дискримінаторі, і чому її не використовують у першому та останньому шарах?
4. Як формулюється функція втрат у класичній генеративно-змагальній мережі?
5. У чому різниця між Binary Cross-Entropy Loss для дискримінатора і для генератора?
6. Що таке «колапс мод» (Mode Collapse) і які його основні ознаки?
7. Чому нормалізація зображень до діапазону $[-1, 1]$ чи до $[0, 1]$ є важливою вимогою для Генератора?
8. Як можна визначити, що генератор і дискримінатор знаходяться у «балансі» під час навчання?
9. Яким чином t-SNE або UMAP допомагають візуалізувати та оцінити розподіл згенерованих даних?
10. Як інтерпретуються значення Inception Score та Fréchet Inception Distance у контексті оцінки якості згенерованих зображень?

Література: [4, С. 364–411], [6, С. 651–716], [8, С. 596–636].

Лабораторна робота №5

Оцінка моделей, метрики якості

Мета – отримання практичних навичок порівняння кількох моделей глибокого навчання за комплексом метрик класифікації, ROC/PR-кривими, пороговим аналізом, калібруванням ймовірностей та бутстреп-оцінюванням невизначеності.

Теоретичні відомості

Для оцінювання моделей глибокого навчання у задачах класифікації зазвичай використовуються інструменти з бібліотек **Scikit-learn**, **NumPy**, **Matplotlib/Seaborn** та методи з фреймворків **TensorFlow/Keras** або **PyTorch**.

Для отримання ймовірнісних передбачень моделі у TensorFlow/Keras застосовується метод `model.predict()`, який повертає масив softmax-ймовірностей:

```
probs = model.predict(X_test)
```

У PyTorch аналогічно використовують:

```
model.eval()
with torch.no_grad():
    outputs = model(X_test)
    probs = torch.softmax(outputs, dim=1)  # .numpy() або .cpu().numpy()
```

Ймовірності потрібні для побудови ROC/PR-кривих, аналізу порогів та калібрування.

Бібліотека **Scikit-learn** надає готові інструменти для формування матриці плутанини та базових метрик якості класифікації:

```
confusion_matrix(y_true, y_pred)
classification_report(...)
precision_recall_fscore_support(...)
```

Узагальнені macro-average, micro-average та weighted-average метрики обчислюються за допомогою параметра `average` у функції `precision_recall_fscore_support`. Наприклад, для macro-average:

```
precision_recall_fscore_support(..., average='macro').
```

Для багатокласових задач ROC/PR-криві будують для одного вибраного класу. Це підхід «one-vs-rest» – один вибраний клас проти всіх інших класів разом. Спочатку формують бінарні мітки:

```
y_binary = (y_true == target_class).astype(int)
```

Потім беруть відповідний стовпець softmax-імовірностей для цього класу:

```
p = probs[:, target_class]
```

У бібліотеці **Scikit-learn** доступні функції для обчислення площ під ROC-та PR-кривими:

```
fpr, tpr, roc_thr = roc_curve(y_binary, p)
roc_auc = auc(fpr, tpr)      # або roc_auc_score(y_binary, p)

precision, recall, pr_thr = precision_recall_curve(y_binary, p)
pr_auc = auc(recall, precision)  # площа під PR-кривою
ap = average_precision_score(y_binary, p)  # Average Precision
```

Average Precision – інтегральна оцінка PR-кривої, яка є кращим показником для незбалансованих класів, ніж проста площа під кривою, оскільки враховує зміну точності при кожному підвищенні порогового значення.

ROC-криві є більш інформативними для збалансованих даних, PR-криві – для задач із суттєво незбалансованими класами.

Для дослідження порогів зазвичай формують набір порогів у діапазоні [0,1] та для кожного значення з набору обчислюють F1-score:

```
thresholds = np.linspace(0.0, 1.0, 101)
f1_scores = []
for t in thresholds:
    y_pred_thr = (p >= t).astype(int)
    f1 = f1_score(y_binary, y_pred_thr)
    f1_scores.append(f1)
```

Для оцінки відповідності ймовірностей реальним частотам подій використовують:

```
brier_score_loss(y_binary, p)
calibration_curve(y_binary, p, n_bins=10)
```

Діаграма надійності будується на основі масивів фактичних частот і середніх прогнозованих ймовірностей, отриманих із `calibration_curve(...)`, де порівнюються середні прогнозовані ймовірності та фактичні частоти позитивного класу.

Бутстреп-підхід реалізується за допомогою вибірки з поверненням:

```
idx = np.random.choice(len(y_true), size=len(y_true), replace=True)
metric_boot = f1_score(y_binary[idx], y_pred_binary[idx])
```

де `y_pred_binary` – попередньо обчислений масив бінарних передбачень моделі при вибраному пороговому значенні; під час бутстреп-оцінювання цей масив не змінюється.

Після повторення цієї процедури (не менше 1000 разів) утворюється емпіричний розподіл метрики, з якого обчислюють процентильні довірчі інтервали.

Для візуалізації результатів аналізу (ROC/PR-криві, F1-криві, Reliability Diagram, матриця плутанини, бутстреп-розподіли) застосовуються стандартні інструменти з бібліотек Matplotlib та Seaborn.

Завдання на лабораторну роботу

1. Завантажити три навчені моделі з ЛР №2 (базову, з Batch Normalization та з Dropout). Для однакової тестової вибірки отримати передбачення у вигляді ймовірностей для кожного класу (softmax outputs) та фінальні передбачені класи. Переконалися, що всі подальші обчислення виконуються на одній і тій самій тестовій вибірці.
2. Для кожної з трьох моделей обчислити матрицю плутанини та метрики точності класифікації для багатокласової задачі (Precision, Recall, F1-score для кожного класу), а також macro-average, micro-average та weighted-average значення. Результати представити у вигляді зведеної таблиці для порівняння. Проаналізувати, які класи і в якій моделі розпізнаються найкраще, а де спостерігаються найбільші помилки.
3. Вибрати один з класів, який розпізнається найгірше, як «позитивний» і для кожної моделі побудувати ROC-криву та PR-криву в підході «one-vs-rest»

(один вибраний клас проти всіх інших класів разом), обчислити AUC-ROC та AUC-PR і порівняти здатність моделей відокремлювати цей клас від усіх інших.

4. Для вибраного «позитивного» класу дослідити залежність F1-score від порогу прийняття рішення для кожної моделі, визначити поріг, що максимізує F1-score, порівняти з результатами при стандартному порозі 0.5 та проаналізувати, як змінюється баланс між хибнопозитивними та хибнонегативними помилками.
5. Оцінити калібрування ймовірностей для кожної моделі за допомогою Brier score та діаграми надійності (Reliability Diagram з 10 інтервалами) для вибраного «позитивного» класу і порівняти, яка з моделей надає найбільш наближені до реальної частоти подій ймовірнісні прогнози.
6. Використати метод бутстреп (Bootstrap, мінімум 1000 ітерацій) для оцінки невизначеності однієї узагальненої метрики (наприклад, F1-score для вибраного класу) окремо для кожної моделі. Побудувати емпіричний розподіл значень, визначити 95% довірчі інтервали та проаналізувати стабільність якості моделей.
7. На основі отриманих метрик, кривих, порогів, оцінки калібрування та довірчих інтервалів сформулювати узагальнений висновок про те, яка з трьох моделей є найкращою для цієї задачі та з яким порогом прийняття рішення доцільно її застосовувати.

**Рекомендації: використати датасет, сформований у ЛР №1, і три навчені CNN-моделі з ЛР №2 (базову, з Batch Normalization та з Dropout). Якщо збережених моделей немає, заново навчити моделі згідно з ЛР №2 з тими самими гіперпараметрами та архітектурами. Для відтворюваності результатів зафіксувати значення seed генератора випадкових чисел.*

Зміст звіту

1. Титульний аркуш.
2. Короткий опис використаних моделей (з ЛР №2) та тестової вибірки.

3. Матриці плутанини та зведена таблиця метрик якості класифікації для трьох моделей. Аналіз найкращих і найгірших класів для кожної моделі.
4. ROC-криві та PR-криві для вибраного «позитивного» класу, значення AUC-ROC та AUC-PR для трьох моделей і порівняння здатності моделей відокремлювати вибраний клас від інших.
5. Графіки залежності F1-score від порогу для трьох моделей, значення оптимального порогу для кожної моделі та коротке порівняння отриманих результатів із порогом 0.5.
6. Діаграма надійності (Reliability Diagram) та значення Brier score для трьох моделей із порівнянням якості калібрування їхніх ймовірнісних прогнозів.
7. Результати бутстреп-оцінювання вибраної метрики: емпіричний розподіл значень та 95% довірчі інтервали для кожної моделі з короткою інтерпретацією стабільності отриманих оцінок.
8. Узагальнений висновок щодо порівняння трьох моделей з урахуванням отриманих метрик, кривих, калібрування, оптимальних порогів і довірчих інтервалів, а також рекомендації щодо вибору моделі для практичного використання.
9. Код програми (прикріплений як окремий текстовий файл чи архів, якщо файлів багато).

Питання для самоконтролю

1. Яка різниця між передбаченими класами та ймовірнісними передбаченнями моделі?
2. Для чого застосовується матриця плутанини та яку інформацію вона дає змогу проаналізувати?
3. Чим відрізняються macro-average, micro-average та weighted-average метрики у багатокласовій класифікації?
4. Що означає підхід «one-vs-rest» при побудові ROC/PR-кривих у багатокласовій задачі і як він застосовується до ймовірностей одного класу?

5. Яка різниця між ROC-кривими та PR-кривими? У яких випадках PR-криві є більш інформативними, ніж ROC-криві?
6. Що вимірює AUC-ROC та як його значення пов'язане зі здатністю моделі відокремлювати класи?
7. Як зміна порогу прийняття рішення впливає на Precision, Recall, F1-score та співвідношення хибнопозитивних і хибнонегативних помилок?
8. Що означає «оптимальний поріг» для вибраного класу і як його визначити?
9. Що вимірює Brier score і яку інформацію надає діаграма надійності (Reliability Diagram) щодо каліброваності ймовірностей?
10. У чому полягає принцип бутстреп-оцінювання і для яких задач його варто використовувати?
11. Яку інформацію дають довірчі інтервали для метрик якості і як вони допомагають оцінити стабільність моделі?

Література: [6, С. 629–650], [8, С. 171–204], [9, С. 333–386].

ЛІТЕРАТУРНІ ДЖЕРЕЛА

1. Bishop C. M., Bishop H. Deep Learning: Foundations and Concepts. Springer, 2024. – 669 p.
2. Bishop C. M., Nasrabadi N. M. Pattern recognition and machine learning. – New York: springer, 2006, Vol. 4, No. 4. – 738 p.
3. Burkov A. The hundred-page machine learning book. Quebec City, Canada: Andriy Burkov, 2019. – 160 p.
4. Chollet F. Deep Learning with Python, Second Edition, 2021. – 504 p.
5. Fleuret F. The little book of deep learning. University of Geneva, 2024. – 185 p.
6. Goodfellow I., Bengio Y., Courville A., Bengio Y. Deep learning. – Cambridge: MIT press, 2016, Vol. 1, No. 2. – 800 p.
7. Prince S. J. Understanding deep learning. MIT press, 2023. – 544 p.
8. Raschka S., Liu Y. H., Mirjalili V. Machine Learning with PyTorch and Scikit-Learn: Develop machine learning and deep learning models with Python. Packt Publishing Ltd, 2022. – 774 p.
9. Roberts D. A., Yaida S., Hanin B. The principles of deep learning theory. Cambridge, MA, USA: Cambridge University Press, 2022. – 471 p.
10. Zhang A., Lipton Z. C., Li M., Smola A. J. Dive into deep learning. – Cambridge University Press. 2023. – 1185 p.