МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

МЕТОДИЧНІ ВКАЗІВКИ

та завдання до лабораторних робіт з курсу
«Глибинне навчання в задачах обробки даних»
для студентів денної та заочної форм навчання спеціальності
F4 – «Системний аналіз та наука про дані»

2025

Методичні вказівки та завдання до лабораторних робіт з курсу «Глибинне навчання в задачах обробки даних» для студентів денної та заочної форм навчання спеціальності F4 – «Системний аналіз та наука про дані» / Укл.: Д.В. Широкорад., А.В. Бакурова - Запоріжжя: НУ «Запорізька політехніка», 2025. - 28 с.

Укладачі**:** Д.В. Широкорад, доцент, к.ф.-м..н.,
А.В. Бакурова, проф., д.е.н.

Рецензент: А.Є. Рябенко, доцент, к.ф.-м.н.

Відповідальний
за випуск Е.В. Терещенко, доцент, к.ф.-м.н.

Затверджено на засіданні кафедри
“Системний аналіз та
обчислювальна математика»
Протокол № 9 від 23.01.2025

Рекомендовано до видання
НМК ФКНТ
Протокол № 6 від 13.02.2025

ЗМІСТ

ВСТУП

Глибинне навчання є одним із ключових напрямів сучасного машинного навчання, який дозволяє комп'ютерним системам самостійно виявляти складні закономірності в даних і виконувати завдання, що раніше вважалися виключно людськими. Завдяки здатності працювати з великою кількістю параметрів і шарів обробки інформації, глибинні нейронні мережі значно підвищили точність у таких завданнях, як класифікація зображень, розпізнавання мови, аналіз тексту та генерація нових даних.

Курс орієнтований на практичне освоєння методів і технік глибинного навчання для вирішення задач класифікації та генерації даних. Розглядаються основи архітектур глибинних нейронних мереж, таких як згорткові нейронні мережі (CNN) та рекурентні нейронні мережі (RNN), а також познайомимося з інноваційними моделями, такими як трансформери.

Виконання лабораторних робіт буде проводитися з використанням популярних бібліотек, таких як TensorFlow та Keras, які забезпечують зручний інтерфейс для роботи з великими нейронними мережами. Виконуючи практичні завдання, студенти зможуть створювати власні проекти та досліджувати різні стратегії навчання та оптимізації моделей.

Цей курс розрахований на студентів, які вже знайомі з основами програмування та лінійної алгебри, а також мають базові знання з машинного навчання.

1 ЛАБОРАТОРНА РОБОТА №1. ЛОГІСТИЧНА РЕГРЕСІЯ

1.1 Загальні теоретичні відомості

Логістична регресія — це один із базових методів класифікації в машинному навчанні, який використовується для розв'язання задач бінарної класифікації. Головна ідея полягає у використанні лінійної моделі, вихід якої трансформується у ймовірність за допомогою функції активації — сигмоїди.

Математичне обґрунтування

- Сигмоїдна функція:

$\sigma(z) = 1 / (1 + \exp(-z))$, де $z = w^T x + b$, w — ваговий вектор, x — вектор ознак, b — зміщення.

- Ймовірність класифікації:

$\hat{y} = \sigma(z)$, де $\hat{y} \in [0, 1]$.

- Функція втрат (логістична):

$L(y, \hat{y}) = - (1/m) \Sigma [y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})]$, де y — істинні мітки, $\hat{y}$ — передбачені ймовірності.

Градієнтний спуск

Для мінімізації функції втрат використовується ітеративний метод оптимізації — градієнтний спуск. Параметри оновлюються за формулами:

$w = w - \alpha \partial L/\partial w$

$b = b - \alpha \partial L/\partial b$,

де α — швидкість навчання.

Вплив швидкості навчання на якість моделі

Швидкість навчання (learning rate,) є важливим гіперпараметром, що визначає, наскільки великий крок робить алгоритм під час оновлення ваг. Від вибору швидкості навчання залежить:

Швидкість збіжності: Занадто мала швидкість уповільнює процес навчання, а занадто велика може призвести до нестабільності та розходження моделі.

Якість моделі: Невдало підібрана швидкість навчання може залишити модель у локальному мінімумі або не дозволити їй збігтися до оптимального значення.

Рекомендації:

Почати з базового значення (наприклад,) і поступово його змінювати.

Використовувати методи адаптивного підбору швидкості (наприклад, Adam, RMSProp).

Передобробка зображень

Передобробка зображень є ключовим етапом у підготовці даних для моделі машинного навчання. Основні техніки:

Конвертація у відтінки сірого: Зменшує розмірність даних, перетворюючи кольорове зображення (3 канали RGB) у монохромне (1 канал).

Нормалізація: Приведення значень пікселів до діапазону [0, 1] для зменшення впливу великих значень на навчання.

Зміна розміру: Масштабування всіх зображень до однакового розміру (наприклад, 64x64 пікселів).

Аугментація даних: Збільшення кількості зображень шляхом обертання, зміщення, зміни яскравості, застосування фільтрів тощо. Це допомагає уникнути перенавчання моделі.

Основні етапи реалізації логістичної регресії

1. Ініціалізація ваг w і b (наприклад, нулями або випадковими значеннями).

2. Обчислення передбачення за допомогою сигмоїди.

3. Обчислення функції втрат.

4. Оновлення параметрів за допомогою градієнтного спуску.

5. Оцінка точності.

Переваги логістичної регресії

- Проста реалізація.
- Інтерпретованість моделі (ваги показують вплив ознак).
- Стабільна робота з нормально розподіленими даними.

Обмеження логістичної регресії

- Непридатна для нелінійно розділених даних без попередньої трансформації ознак.
- Чутливість до мультиколінеарності серед ознак.
- Потребує масштабування ознак для коректної роботи.

1.2 Завдання до лабораторної роботи

1. Запустити скрипт та впевнитись в його роботі.
2. Перевірити якість класифікації на власних зображеннях.
3. Дослідити вплив швидкості навчання на якість класифікації зображень нейронною мережею.
4. Реалізувати ініціалізацію параметрів w, b іншими значеннями.

5. Додати додаткову передобробку зображень (наприклад, конвертація у відтінки сірого). Оцінка результатів роботи мережі з новими даними.

1.3 Вказівки до виконання лабораторної роботи

1. Запустити скрипт та впевнитись в його роботі.

Для цього за необхідністю потрібно створити безкоштовний обліковий запис. Допоміжні файли доступні в системі Moodle. Їх треба перетягнути на вкладку «Файли».

Для запуску скрипту вибрати в верхньому меню «Середовище виконання» - «Виконати всі». Також можливо послідовно запускати кожну комірку, натискаючі на кнопку зліва від неї.

2. Перевірити якість класифікації на власних зображеннях:

Завантажте власні зображення у форматі PNG або JPEG.

Використайте бібліотеку Pillow для обробки зображень:

```
from PIL import Image
import numpy as np

# Завантаження зображення
img = Image.open("path_to_image.jpg")
img_resized = img.resize((64, 64))  # Зміна розміру
img_array = np.array(img_resized) / 255.0  # Нормалізація
img_vector = img_array.flatten()  # Перетворення у вектор
```

3. Дослідити вплив швидкості навчання на якість класифікації зображень нейронною мережею:

Проведіть навчання моделі з різними значеннями швидкості навчання та порівняйте результати:

```
learning_rates = [0.01, 0.1, 0.5]
for lr in learning_rates:
```

```
        [запуск моделі]
        print(f"Швидкість навчання: {lr}, Точність:
{accuracy}")
```

4. Реалізувати ініціалізацію параметрів w, b іншими значеннями:

Додайте різні методи ініціалізації ваг і порівняйте їх вплив на навчання:

```
    def initialize_weights(shape, method="zeros"):
        if method == "zeros":
            return np.zeros(shape)
        elif method == "random":
            return np.random.randn(*shape) * 0.01
        elif method == "glorot":
            limit = np.sqrt(6 / sum(shape))
            return       np.random.uniform(-limit,
limit, shape)

    weights    =    initialize_weights((64,    64),
method="glorot")
```

5. Додати додаткову передобробку зображень (наприклад, конвертація у відтінки сірого):

Виконайте передобробку зображень і оцініть її вплив на модель:

```
    # Конвертація у відтінки сірого
    img_gray = img.convert("L")
    img_gray_resized = img_gray.resize((64, 64))
    img_gray_array = np.array(img_gray_resized) /
255.0
```

Реалізуйте аугментацію даних для покращення роботи моделі:

2 ЛАБОРАТОРНА РОБОТА №2. ОДНОШАРОВА НЕЙРОННА МЕРЕЖА

2.1 Загальні теоретичні відомості

Одношарова нейронна мережа (перцептрон) є найпростішою формою штучної нейронної мережі, що використовується для класифікації та розв'язання простих задач розпізнавання образів. Незважаючи на свою простоту, вона є важливою основою для розуміння більш складних багатошарових нейронних мереж.

Одношарова нейронна мережа складається з:

- Вхідного шару, що містить n нейронів (кількість ознак вхідних даних).
- Вихідного шару, що містить один або кілька нейронів (залежно від задачі).
- Вагових коефіцієнтів, які визначають внесок кожного вхідного сигналу у вихідний результат.
- Функції активації, що визначає реакцію нейрона на вхідні дані.

Математична модель нейрона описується рівнянням:

$y = f(\Sigma w_i x_i + b)$

де:

- x_i — вхідні сигнали,
- w_i — вагові коефіцієнти,
- b — зміщення (bias),
- f — функція активації,
- y — вихід нейрона.

Функції активації

Основні функції активації:

- Порігова функція (step function)
- Сигмоїдна функція (sigmoid)
- Гіперболічний тангенс (tanh)
- ReLU (Rectified Linear Unit)

Навчання одношарової нейронної мережі

Процес навчання полягає в оновленні вагових коефіцієнтів:

$w_i^{(t+1)} = w_i^{(t)} + \eta (y_{desired} - y) x_i$

де:

- η — коефіцієнт навчання,
- $y_{desired}$ — бажаний вихід,
- y — отриманий вихід.

Одношарові нейронні мережі використовуються у:
- Лінійній класифікації,
- Розпізнаванні простих образів,
- Фільтрації сигналів.

Основний недолік – неможливість розв’язувати задачі, що не є лінійно роздільними.

2.2 Завдання до лабораторної роботи

1. Дослідити якість класифікації точок за різних:

- розмірів прихованого шару;
- швидкості навчання;
- функції активації прихованого шару.

2. Застосувати мережу на інших датасетах.

2.3 Вказівки до виконання лабораторної роботи

Використати наданий код та допоміжні файли в системі Moodle для реалізації нейронної мережі.

Запустити модель та оцінити її продуктивність на вихідному датасеті.

Внести зміни у параметри моделі (кількість нейронів, швидкість навчання, функції активації) та проаналізувати вплив на точність класифікації.

Для завантаження датасету можливо використовувати наступний фрагмент коду.

```
    noisy_circles, noisy_moons, blobs,
gaussian_quantiles, no_structure =
load_extra_datasets()

    datasets = {"noisy_circles": noisy_circles,
                "noisy_moons": noisy_moons,
                "blobs": blobs,
                "gaussian_quantiles":
gaussian_quantiles}

    dataset = "gaussian_quantiles
```

Випробувати навчання нейронної мережі на кожному з датасетів.

Проаналізувати отримані результати та зробити висновки.

Представити результати у вигляді графіків та таблиць.

Описати вплив зміни параметрів на навчання мережі.

Підготувати висновки щодо роботи нейронної мережі з різними параметрами та датасетами..

3 ЛАБОРАТОРНА РОБОТА №3. БАГАТОШАРОВА НЕЙРОННА МЕРЕЖА

3.1 Теоретичні відомості

Багатошарові нейронні мережі (MLP, Multi-Layer Perceptron) є основним класом штучних нейронних мереж, що складаються з кількох шарів нейронів: вхідного, одного або декількох прихованих та вихідного шару. Такі мережі використовуються для моделювання складних залежностей між вхідними та вихідними даними.

Архітектура багатошарової нейронної мережі

Вхідний шар – приймає дані у вигляді числових векторів.

Приховані шари – виконують нелінійне перетворення вхідних даних, що дає можливість виявляти складні патерни.

Вихідний шар – генерує результат моделі у вигляді передбачень або класів.

Функції активації

У багатошарових мережах використовуються різні функції активації:

ReLU (Rectified Linear Unit) – широко застосовується через ефективність у глибоких мережах.

Sigmoid – використовується у вихідних шарах для бінарної класифікації.

Tanh – забезпечує значення між -1 та 1, що корисно в деяких задачах.

Зворотне поширення помилки та оптимізація

Процес навчання нейронної мережі базується на методі зворотного поширення помилки (Backpropagation), що дозволяє коригувати ваги шляхом градієнтного спуску або його варіантів (Adam, RMSprop тощо).

Регуляризація та нормалізація

Для покращення навчання та уникнення перенавчання використовуються:

Dropout – випадкове вимикання частини нейронів під час навчання.

L1 та L2-регуляризація – допомагає уникати надмірного підлаштування моделі під навчальні дані.

Batch Normalization – покращує стабільність і швидкість навчання.

Практичне застосування

Багатошарові нейронні мережі широко використовуються для задач класифікації, регресії, обробки зображень та природної мови. Вони є базовою архітектурою для більш складних глибоких моделей, таких як згорткові та рекурентні нейронні мережі.

3.2 Завдання до лабораторної роботи

Використовуючи посилання на ноутбуки з реалізацією багатошарової нейронної мережі, розробити модель для розв'язку задачі бінарної классифікації.

Приклади задач та датасетів можна знайти за посиланням: https://machinelearningmastery.com/standard-machine-learning-datasets/, https://www.kaggle.com/

3.3 Вказівки до виконання лабораторної роботи

1. Завантаження та попередня обробка даних

Вибрати відповідний датасет для бінарної класифікації.

Завантажити та нормалізувати вхідні дані.

Розділити вибірку на навчальну та тестову частини.

```
    import numpy as np
    import matplotlib.pyplot as plt
    from       sklearn.model_selection       import
train_test_split
    from        sklearn.preprocessing        import
StandardScaler
    from tensorflow import keras
    from tensorflow.keras import layers

    # Завантаження та підготовка даних
    data = np.loadtxt("dataset.csv", delimiter=",")
    X, Y = data[:, :-1], data[:, -1]
    X_train,    X_test,    Y_train,    Y_test    =
train_test_split(X,        Y,        test_size=0.2,
random_state=42)
    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test)
```

Визначити архітектуру мережі (кількість шарів, кількість нейронів у кожному шарі).

Вибрати функції активації для кожного шару.

Реалізувати модель у Python, використовуючи скрипт та допоміжні файли з Moodle.

Провести навчання моделі на навчальному наборі даних.

Візуалізувати зміну функції втрат під час навчання.

Оцінка та візуалізація результатів

Оцінити точність моделі на тестових даних.

Візуалізувати межі класифікації, якщо можливо.

Проаналізувати отримані результати та зробити висновки.

4 ЛАБОРАТОРНА РОБОТА №4. МЕТОДИ РЕГУЛЯРИЗАЦІЇ ТА ОПТИМІЗАЦІЇ

4.1 Теоретичні відомості

Методи регуляризації та оптимізації є важливими інструментами в машинному навчанні та статистиці. Вони допомагають покращити узагальнюючу здатність моделей, запобігти перенавчанню та забезпечити ефективний процес пошуку оптимальних параметрів.

Регуляризація

Перенавчання (overfitting) виникає, коли модель добре пристосовується до навчальних даних, але погано узагальнює нові дані. Це може бути спричинене високою складністю моделі або недостатньою кількістю навчальних даних.

Основні методи регуляризації

1. L1-регуляризація (Lasso) – додає до функції втрат модуль вагових коефіцієнтів:

$L = \Sigma (y_i - \hat{y}_i)^2 + \lambda \Sigma |w_j|$

що сприяє розрідженню моделі (деякі коефіцієнти стають нульовими).

2. L2-регуляризація (Ridge) – додає до функції втрат квадрат вагових коефіцієнтів:

$L = \Sigma (y_i - \hat{y}_i)^2 + \lambda \Sigma w_j^2$

що зменшує їх значення та допомагає уникнути надмірного впливу окремих параметрів.

3. Dropout – випадкове вимкнення нейронів під час навчання нейронних мереж для зменшення залежності моделі від конкретних ознак.

4. Раннє зупинення (Early Stopping) – припинення навчання, коли помилка на валідаційних даних перестає зменшуватись.

Оптимізація

Градієнтний спуск

Градієнтний спуск – це один з основних методів мінімізації функції втрат. Його суть полягає в коригуванні параметрів у напрямку антиградієнта функції втрат:

$w := w - \alpha \nabla L(w)$

3.2. Варіанти градієнтного спуску

1. Звичайний градієнтний спуск (Batch Gradient Descent, BGD) – використовує весь навчальний набір для оновлення ваг.

2. Стохастичний градієнтний спуск (Stochastic Gradient Descent, SGD) – оновлює ваги після кожного прикладу, що може прискорити процес, але вносить шум у навчання.

3. Міні-пакетний градієнтний спуск (Mini-Batch Gradient Descent) – компроміс між BGD та SGD, оновлює ваги після обробки малих підмножин даних.

3.3. Адаптивні методи оптимізації

1. Momentum – використовує експоненційне згладжування градієнтів для прискорення руху в потрібному напрямку.

2. AdaGrad – змінює швидкість навчання для кожного параметра залежно від накопичених градієнтів.

3. RMSprop – модифікація AdaGrad, яка використовує ковзне середнє квадратів градієнтів.

4. Adam (Adaptive Moment Estimation) – комбінує ідеї Momentum та RMSprop, що робить його одним із найпопулярніших алгоритмів.

4. Висновки

Методи регуляризації та оптимізації відіграють важливу роль у побудові ефективних моделей машинного навчання. Регуляризація запобігає перенавчанню, а оптимізаційні алгоритми дозволяють ефективно знаходити оптимальні параметри. Вибір конкретного методу залежить від специфіки задачі та властивостей навчальних даних.

4.2 Завдання до лабораторної роботи

1. Реалізувати лінійну регресію з L1 або L2 регуляризацією, порівняти отримані результати.

2. Використати метод Dropout у нейронній мережі та оцінити його вплив на точність моделі.

3. Дослідити адаптивні методи оптимізації (мінімум 2 методи з Momentum, AdaGrad, RMSprop, Adam фбо інших) на конкретному датасеті та порівняти їх швидкість збіжності.

4. Виконати аналіз ефекту раннього зупинення на процес навчання моделі та її узагальнюючу здатність.

5. Для виконання роботи можна використовувати ноутбуки, зазначені у відповідних розділах Moodle (регуляризація, оптимізація), але необхідно самостійно реалізувати хоча б один метод регуляризації або оптимізації.

6. Дослідження можна проводити на одному з попередньо використаних датасетів або обрати новий.

4.3 Вказівки до виконання лабораторної роботи

Реалізувати моделі у Python, використовуючи скрипт та допоміжні файли з Moodle (регуляризація, оптимізація).

Провести навчання моделей на навчальному наборі даних.

Візуалізувати зміну функції втрат під час навчання.

Оцінити точність моделі на тестових даних.

Візуалізувати межі класифікації, якщо можливо.

Проаналізувати отримані результати та зробити висновки.

5 ЛАБОРАТОРНА РОБОТА №5 ЗГОРТОЧНІ НЕЙРОННІ МЕРЕЖІ

5.1 Теоретичні відомості

Згорточні нейронні мережі (Convolutional Neural Networks, CNN) — це тип штучних нейронних мереж, які ефективно обробляють зображення та інші дані із просторовою структурою. Вони використовують спеціальні шари для виділення локальних особливостей та зменшення кількості параметрів у моделі.

Основні компоненти CNN

Згорточний шар (Convolutional Layer)

Цей шар використовує фільтри (ядра), які проходять через зображення та виділяють важливі особливості. Формула згортки виглядає так:

$C(i, j) = \Sigma \Sigma I(m, n) * K(i-m, j-n)$

Функція активації (ReLU - Rectified Linear Unit)

Після згортки застосовується нелінійна функція активації: $f(x) = \max(0, x)$

Шар підбору (Pooling Layer)

Зменшує розмірність карти ознак, вибираючи максимальне або середнє значення в області.

Функція втрат

Для задач класифікації використовується крос-ентропійна функція:

$L = - \Sigma y \log(\hat{y})$

Архітектура типової CNN

Приклад типової архітектури нейронної мережі:

```
    from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D,
MaxPooling2D, Flatten, Dense
    model = Sequential([
        Conv2D(32, (3,3), activation='relu',
input_shape=(28,28,1)),
        MaxPooling2D((2,2)),
        Conv2D(64, (3,3), activation='relu'),
        MaxPooling2D((2,2)),
        Flatten(),
        Dense(128, activation='relu'),
        Dense(10, activation='softmax')
```

```
    ])
    model.compile(optimizer='adam',
loss='categorical_crossentropy',
metrics=['accuracy'])
    model.summary()
```

Дослідження ефективності CNN

Для покращення моделі можна експериментувати з такими параметрами:

- Розмір фільтрів (3×3, 5×5);
- Кількість згорткових шарів;
- Тип пулінгу (MaxPooling, AveragePooling);
- Аугментація даних;
- Оптимізація гіперпараметрів/

5.2 Завдання до лабораторної роботи

1. **Класифікація рукописних цифр (MNIST)**
   - Завантажте та підготуйте набір даних MNIST.
   - Побудуйте базову CNN-модель для класифікації цифр.
   - Візуалізуйте передбачені результати на тестових зображеннях.
2. **Дослідження впливу гіперпараметрів**
   - Змініть кількість згорткових шарів (2, 3 або більше) та оцініть зміни в точності.
   - Випробуйте різні розміри фільтрів (3×3, 5×5).
   - Порівняйте різні типи пулінгу (MaxPooling vs. AveragePooling).
3. **Аугментація даних**
   - Використовуйте ImageDataGenerator для генерації нових варіацій зображень.
   - Переконайтесь, що модель працює краще на розширених даних.
4. **Створення нейромережі для власних зображень**
   - Використайте власні чорно-білі зображення (мінімум 5 класів).
   - Налаштуйте та навчіть модель для їх класифікації.
   - Оцініть результати роботи мережі на ваших даних.

_

5.3 Вказівки до виконання лабораторної роботи

1. **Завантаження та підготовка даних**

```
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical
(X_train, y_train), (X_test, y_test) = mnist.load_data()
X_train = X_train.reshape((-1, 28, 28, 1)) / 255.0
X_test = X_test.reshape((-1, 28, 28, 1)) / 255.0
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)
```

2. **Побудова базової моделі CNN**

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

model = Sequential([
    Conv2D(32, (3,3), activation='relu', input_shape=(28,28,1)),
    MaxPooling2D((2,2)),
    Conv2D(64, (3,3), activation='relu'),
    MaxPooling2D((2,2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dense(10, activation='softmax')
])

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model.summary()
```

3. **Навчання та оцінка моделі**

```
model.fit(X_train, y_train, epochs=10, validation_data=(X_test, y_test))
loss, accuracy = model.evaluate(X_test, y_test)
print(f"Точність моделі: {accuracy:.4f}")
```

4. **Аугментація даних для покращення результатів**

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
datagen = ImageDataGenerator(rotation_range=10, width_shift_range=0.1, height_shift_range=0.1, zoom_range=0.1)
datagen.fit(X_train)
```

5. **Навчання моделі на розширених даних**

```
model.fit(datagen.flow(X_train, y_train, batch_size=32), epochs=10, validation_data=(X_test, y_test))
```

6. **Візуалізація передбачень**

```
import matplotlib.pyplot as plt
import numpy as np
predictions = model.predict(X_test[:5])
for i in range(5):
    plt.imshow(X_test[i].reshape(28, 28), cmap='gray')
    plt.title(f"Передбачено: {np.argmax(predictions[i])}")
plt.show()
```

7. **Навчання моделі на власних зображеннях**

Зберіть власний набір зображень у форматі 28x28 пікселів.

Використовуйте ImageDataGenerator.flow_from_directory() для завантаження даних.

Навчіть модель і оцініть точність передбачень.

6 ЛАБОРАТОРНА РОБОТА №6 ВИКОРИСТАННЯ ПЛАТФОРМИ HUGGINFACE

6.1 Теоретичні відомості

1. Моделі комп'ютерного зору (CV) у Hugging Face

Hugging Face надає інструменти для роботи з сучасними архітектурами CV:

- **Vision Transformer (ViT)**

Перетворює зображення на послідовність патчів (фрагментів), обробляє їх механізмами уваги. Ефективний для класифікації, порівняно з CNN.

- **DETR (Detection Transformer)**

Використовує трансформери для детекції об'єктів. Заміняє традиційні методи з якорними рамками на пряме прогнозування координат.

- **Swin Transformer**

Архітектура з ієрархічною структурою, що дозволяє обробляти зображення різних масштабів. Використовується для сегментації та класифікації.

- **CLIP (Contrastive Language–Image Pretraining)**

Модель, яка зв'язує текст та зображення. Використовується для пошуку зображень за текстовим запитом або генерації описів.

2. Попередня обробка зображень

- **Feature Extractors** автоматизують:
  - **Нормалізацію пікселів** (масштабування значень до діапазону [0, 1] або [-1, 1]).
  - **Розбиття на патчі** (для ViT).
  - **Зміну розміру** (наприклад, до 224x224 для сумісності з моделлю).
  - **Аугментацію** (обертання, віддзеркалення).

3. Fine-tuning у CV

- **Перенос навчання**: Використання попередньо навченої моделі (на великих датасетах, як ImageNet) та адаптація її під конкретну задачу (наприклад, класифікацію медичних зображень).
- **Переваги**:
  - Економить час та обчислювальні ресурси.
  - Досягає високої точності на малих датасетах.

4. Hugging Face Datasets для CV

- Популярні датасети:
  - **CIFAR-10/100**: 60 тис. зображень у 10/100 класах.
  - **ImageNet**: 1.2 млн зображень у 1000 класах.
  - **COCO**: 330 тис. зображень для детекції, сегментації, підписів.

6.2 Завдання до лабораторної роботи

1: Робота з готовими пайплайнами

Мета: Використати pipeline для базових задач.
Кроки:

Виконайте класифікацію зображення за допомогою моделі ViT.

Використайте пайплайн для генерації підписів до зображень (image captioning).

2: Детекція об’єктів з DETR

Завантажте зображення з камери або інтернету.

Використайте модель facebook/detr-resnet-50 для детекції.

Візуалізуйте результати за допомогою matplotlib.

3: Сегментація зображень

Використайте модель facebook/mask2former-swin-small-coco-instance.

Застосуйте її до зображення з людиною або автомобілем.

Збережіть маску сегментації у вигляді PNG-файлу.

4: Fine-tuning ViT для класифікації

Завантажте датасет cifar10 через datasets.

Підготуйте дані: нормалізація, розбиття на тренувальний/тестовий набори.

Налаштуйте TrainingArguments та запустіть тренування.

5: Створення додатку з Gradio **(необов’язково)**

Створіть функцію для обробки зображень.

Використайте gr.Interface для побудови UI.

Запустіть додаток локально або опублікуйте на Spaces.

6: Робота з CLIP

Завантажте модель openai/clip-vit-base-patch32.

Порівняйте текстовий запит ("a red car") з набором зображень.

Виведіть зображення з найвищою схожістю.

6.3 Вказівки до виконання лабораторної роботи

Завдання 1: Класифікація зображень

Встановіть бібліотеки transformers та pillow.

```
from transformers import pipeline
classifier = pipeline("image-classification", model="google/vit-base-patch16-224")
result = classifier("шлях_до_зображення.jpg")
print("Топ-3 класи:", result[:3])
```

Збережіть результати у текстовому файлі або виведіть у консоль.

Примітка: Для зображень з інтернету використовуйте URL замість шляху.

Завдання 2: Детекція об'єктів

Встановіть torch та matplotlib для візуалізації.

```
from transformers import AutoImageProcessor, AutoModelForObjectDetection
image_processor = AutoImageProcessor.from_pretrained("facebook/detr-resnet-50")
model = AutoModelForObjectDetection.from_pretrained("facebook/detr-resnet-50")
inputs = image_processor(images=image, return_tensors="pt")
outputs = model(**inputs)
```

Візуалізуйте результати за допомогою matplotlib, використовуючи координати з outputs.

Порада: Використовуйте документацію DETR для формату виводу.

Завдання 3: Сегментація зображень

Завантажте модель та обробіть зображення:

```
from transformers import AutoImageProcessor, AutoModelForUniversalSegmentation
processor = AutoImageProcessor.from_pretrained("facebook/mask2former-swin-small-coco-instance")
```

```
    model = AutoModelForUniversalSegmentation.from_pretrained("facebook/mask2former-swin-small-coco-instance")
    inputs = processor(images=image, return_tensors="pt")
    outputs = model(**inputs)
```

Збережіть маску сегментації у форматі PNG:

```
    import numpy as np
    from PIL import Image
    mask = np.argmax(outputs.class_queries_logits, axis=1)
    Image.fromarray(mask.astype(np.uint8)).save("mask.png")
```

Завдання 4: Fine-tuning ViT

Завантажте датасет CIFAR-10:

```
    from datasets import load_dataset
    dataset = load_dataset("cifar10")
```

Підготуйте дані за допомогою ViTImageProcessor:

```
    from transformers import ViTImageProcessor
    processor = ViTImageProcessor.from_pretrained("google/vit-base-patch16-224")
```

Навчіть модель:

from transformers import TrainingArguments, Trainer

training_args = TrainingArguments(output_dir="./results", num_train_epochs=3, per_device_train_batch_size=16)

trainer = Trainer(model=model, args=training_args, train_dataset=dataset["train"], eval_dataset=dataset["test"])

trainer.train()

Важливо: Використовуйте GPU (наприклад, у Google Colab).

Завдання 5: Додаток з Gradio (необов'язково)

Створіть файл app.py з кодом:

```
    import gradio as gr
    from transformers import pipeline
    classifier = pipeline("image-classification", model="google/vit-base-patch16-224")
```

```
    def classify(image):
        results = classifier(image)
        return {item["label"]: item["score"] for
item in results}
    gr.Interface(fn=classify, inputs="image",
outputs="label").launch()
```

Запустіть додаток локально:

python app.py

Для деплою на Hugging Face Spaces додайте requirements.txt зі списком залежностей.

Завдання 6: Пошук зображень з CLIP

Завантажте модель:

```
    from transformers import CLIPProcessor,
CLIPModel
    model =
CLIPModel.from_pretrained("openai/clip-vit-base-
patch32")
    processor =
CLIPProcessor.from_pretrained("openai/clip-vit-
base-patch32")
```

Обчисліть схожість тексту та зображення:

```
    inputs = processor(text=["текст_запиту"],
images=image, return_tensors="pt", padding=True)
    outputs = model(**inputs)
    probs =
outputs.logits_per_image.softmax(dim=1)
```

Виведіть зображення з найвищою ймовірністю.

Загальні рекомендації

Для **завдань 2-4** обов'язково використовуйте GPU (налаштуйте середовище у Google Colab).

Зберігайте проміжні результати (наприклад, ваги моделей) для подальшого аналізу.

Додавайте коментарі до коду та скріншоти результатів у звіт.

ЛІТЕРАТУРА

1. Bishop C. M. Pattern Recognition and Machine Learning / C. M. Bishop. – New York : Springer, 2006. – 738 p.

2. Goodfellow I., Bengio Y., Courville A. Deep Learning / I. Goodfellow, Y. Bengio, A. Courville. – Cambridge, MA : MIT Press, 2016. – 800 p.

3. Субботін, С. О.Нейронні мережі : теорія та практика: навч. посіб. / С. О. Субботін. – Житомир : Вид. О. О. Євенок, 2020. – 184 с.

4. Руденко, О. Г. Штучні нейронні мережі / О. Г. Руденко, Є. В. Бодянський. – Харків : Компанія СМІТ, 2006. – 404 с.

5. Олійник, А. О. Інтелектуальний аналіз даних : навч. посіб. / А. О. Олійник, С. О. Субботін, О. О. Олійник. – Запоріжжя : ЗНТУ, 2011. – 271 с.

6. Rumelhart D. E., Hinton G. E., Williams R. J. Learning representations by back-propagating errors // Nature. – 1986. – Vol. 323, no. 6088, pp. 533–536.

7. A. Géron, Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, Third edition. Sebastopol, CA: O'Reilly Media, Inc., 2023. ISBN: 978-1-098-12597-4.

8. F. Chollet, Deep Learning with Python, Second edition. Shelter Island, NY, USA: Manning Publications Co., 2021. ISBN: 978-1-61729-686-4.

9. Hugging Face Documentation [Електронний ресурс] / Hugging Face Documentation. – Available at: https://huggingface.co/docs (accessed: 01.02.2025).

10. Haykin S. Neural Networks and Learning Machines / S. Haykin. – Upper Saddle River, NJ : Pearson, 2008. – 624 p.

11. LeCun Y., Bengio Y., Hinton G. Deep learning // Nature. – 2015. – Vol. 521, no. 7553, pp. 436–444.

12. K. P. Murphy, Probabilistic Machine Learning: Advanced Topics. Cambridge, MA, USA: The MIT Press, 2023. ISBN: 978-0-262-04843-9. Available: https://probml.github.io/pml-book/book2.html.

13. Литвин В. В., Пелещак Р. М., Висоцька В. А. Глибинне навчання, 2021.Prometheus. «Машинне навчання» [Електронний

ресурс] / Prometheus. – Режим доступу: https://prometheus.org.ua/ (дата звернення: 01.02.2025).

14. Булгакова О. С., Зосімов В. В., Поздєєв В. О. Методи та системи штучного інтелекту. Теорія та практика. Навчальний посібник. – Олді плюс, 2020, - 356 с.