**ПРАКТИЧНА РОБОТА №5**

**Розпізнавання образів**

Мета: Закріпити основні поняття використання нейронных мереж для розпізнавання зображень.

Завдання:

Розглянути задачу класифікації рукописних цифр з використанням набору даних MNIST. Це популярний набір даних, який містить зображення рукописних цифр (від 0 до 9). Створити просту модель нейронної мережі для розпізнавання цих цифр.

**Хід виконання.**

1. Імпортуємо необхідні бібліотеки

import tensorflow as tf

from tensorflow.keras import layers, models

from tensorflow.keras.datasets import mnist

import matplotlib.pyplot as plt

Якщо вам потрібно вручну завантажити набір даних MNIST для інших цілей або використання, ви можете зробити це з офіційного вебсайту:

<https://www.nist.gov/itl/products-and-services/emnist-dataset>

На цій сторінці ви знайдете різні формати даних, включаючи оригінальні зображення та відповідні мітки, які можна завантажити у вигляді zip-файлів

Але можна зробити це вбудованими засобами бібліотеки

1. Завантажимо набір даних MNIST.

(train\_images, train\_labels), (test\_images, test\_labels) = mnist.load\_data()

1. Виконаємо попередню обробку даних

**Перетворення формату**: використовуємо reshape, щоб змінити форму зображень. Спочатку зображення мають форму (60000, 28, 28), де 60000 — це кількість зображень, а 28 x 28 — розмір кожного зображення. Додаємо четвертий вимір (канал) для сумісності з моделями, які очікують вхідні дані у формі (кількість зображень, висота, ширина, канали).

**Нормалізація**: Ділимо значення пікселів на 255, щоб нормалізувати їх до діапазону [0, 1]. Це важливо для швидшого та ефективнішого навчання нейронної мережі.

train\_images = train\_images.reshape((60000, 28, 28, 1)).astype('float32') / 255

test\_images = test\_images.reshape((10000, 28, 28, 1)).astype('float32') / 255

1. Побудуємо модель

У наступному коді створюється **нейронна мережа**, яка вміє розпізнавати зображення, наприклад, цифри від 0 до 9.

**Модель Sequential.** Це означає, що всі шари (етапи обробки) додаються один за одним, по черзі. Як конвеєр — одне за іншим.

**Конволюційний шар:**

Ці шари шукають важливі ознаки на зображенні, наприклад, краї, лінії, кути.

Conv2D(32, (3, 3), activation='relu'):

32 фільтри (такі собі маленькі "сканери"),кожен має розмір 3x3 пікселі,

ReLU допомагає моделі краще навчатися (він прибирає негативні значення).

**MaxPooling2D:**

Цей шар зменшує розмір зображення, залишаючи тільки найважливішу інформацію. Умовно кажучи, "стискає" дані без втрати суті.

**Другий конволюційний блок**

Такий самий, як перший, але з 64 фільтрами — він шукає вже складніші ознаки.

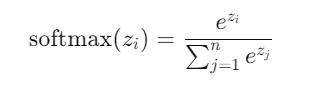
**Flatten:**

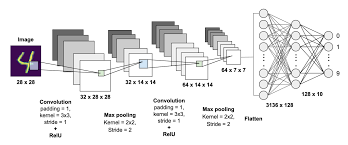
Цей шар перетворює всі дані в один довгий список (вектор), щоб їх можна було передати на останній етап.

**Повнозв'язні шари:**

Dense(64, activation='relu'): Повнозв'язний шар з 64 нейронами та функцією активації ReLU.

Dense(10, activation='softmax'): Вихідний шар, який має 10 нейронів (по одному для кожної цифри від 0 до 9) і використовує функцію активації softmax для ймовірнісного прогнозування класів.





model = models.Sequential([

layers.Conv2D(32, (3, 3), activation='relu'),

layers.MaxPooling2D((2, 2)),

layers.Conv2D(64, (3, 3), activation='relu'),

layers.MaxPooling2D((2, 2)),

layers.Flatten(),

layers.Dense(64, activation='relu'),

layers.Dense(10, activation='softmax')

])

1. Виконаємо компіляцію моделі:

**Параметри компіляції**

**Оптимізатор:**

optimizer='adam': Adam (Adaptive Moment Estimation) — це популярний оптимізатор, який адаптує швидкість навчання для кожного параметра на основі середніх значень градієнтів та їх квадратів. Adam часто працює краще за стандартні методи оптимізації, такі як SGD (Stochastic Gradient Descent), і є хорошим вибором для багатьох задач.

**Функція втрат:**

loss='sparse\_categorical\_crossentropy': Ця функція втрат використовується для задач класифікації з кількома класами, коли мітки класів представлені цілими числами (наприклад, 0, 1, 2, ..., 9 для MNIST). Вона обчислює крос-ентропію між передбаченими й істинними мітками. Це хороший вибір, коли ви маєте справу з великим числом класів.

**Метрики:**

metrics=['accuracy']: Ви вибрали точність (accuracy) як метрику для оцінювання продуктивності моделі під час навчання та тестування. Точність показує, яка частка передбачень моделі є правильними.

model.compile(optimizer='adam',  
 loss='sparse\_categorical\_crossentropy',  
 metrics=['accuracy'])

1. Виконаємо навчання моделі

model.fit(train\_images, train\_labels, epochs=5, batch\_size=64, validation\_split=0.1)

1. Виконаємо оцінку моделі

test\_loss, test\_acc = model.evaluate(test\_images, test\_labels)

print(f'Точність на тестових даних: {test\_acc:.4f}')

**Результати**: Ваша точність буде значенням від 0 до 1, де 1 означає 100% точність. Чим ближче точність до 1, тим краще модель справляється з класифікацією.

1. Виконаємо розпізнавання для навченої моделі:

predictions = model.predict(test\_images)

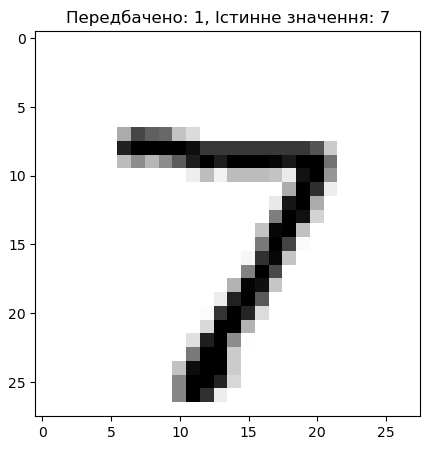
1. Візуалізуємо розпізнавання першого зображення з набору:

plt.figure(figsize=(5,5))

plt.imshow(test\_images[0].reshape(28, 28), cmap=plt.cm.binary)

plt.title(f'Передбачено: {predictions[0].argmax()}, Істинне значення: {test\_labels[0]}')

plt.show()



**Завдання для самостійного виконання:**

1. Спробуйте розпізнати інший символ з набору
2. Створи своє власне зображення цифри та спробуй його розпізнати.

Щоб створити своє зображення та розпізнати його на моделі, навченій на MNIST, вам потрібно буде виконати кілька кроків. Ось докладний посібник:

Намалюйте цифру: Використай paint та намалюйте цифру від 0 до 9. Намагайтеся зробити її схожою на цифри в наборі даних MNIST (тонкі лінії, центроване положення). Збережі зображення з розширенням PNG.

Далі для обробки зображень знадобиться інсталювати бібліотеку opencv-python

Завантаж зображення у блокнот. Моделі MNIST навчаються на чорно-білих зображеннях у градаціях сірого. Перетворіть ваше кольорове зображення у відтінки сірого.

import cv2

img = cv2.imread("d:/my.png", cv2.IMREAD\_GRAYSCALE)

Зміна розміру: Усі зображення в наборі даних MNIST мають розмір 28x28 пікселів. Змініть розмір вашого зображення до 28x28 пікселів.

img\_resized = cv2.resize(img\_inverted, (28, 28), interpolation=cv2.INTER\_LINEAR)

Центрування (необов'язково, але рекомендовано): Переконайтеся, що ваша цифра знаходиться приблизно в центрі зображення 28x28.

Нормалізація (важливо): Нормалізуйте значення пікселів у діапазоні від 0 до 1. У MNIST значення пікселів знаходяться в діапазоні від 0 (чорний) до 255 (білий), тому вам потрібно розділити всі значення пікселів на 255.

img\_normalized = img\_resized.astype('float32') / 255.0

Описуємо масив numpy для обробки:

1 (перший вимір): Це розмір пакету (batch size). У контексті обробки одного окремого зображення для передбачення, ми зазвичай обробляємо його як пакет, що містить одне зображення. Тому перший вимір дорівнює 1.

28 (другий вимір): Це висота зображення в пікселях. Набір даних MNIST складається із зображень розміром 28x28 пікселів.

28 (третій вимір): Це ширина зображення в пікселях. Знову ж таки, для MNIST це 28 пікселів.

1 (четвертий вимір): Це кількість каналів кольору. Зображення MNIST є у відтінках сірого, тому вони мають лише один колірний канал. Для кольорових зображень (наприклад, RGB) цей вимір був би 3.

img\_reshaped = img\_normalized.reshape(1, 28, 28, 1)

Тепер можна подати цей набір для прогнозування в навчену модель