Лабараторна робота 16

app.py

import tensorflow as tf

from tensorflow.keras.layers import Dense, Reshape, Flatten, BatchNormalization, LeakyReLU, Conv2D, Conv2DTranspose

from tensorflow.keras.optimizers import Adam

import numpy as np

import matplotlib.pyplot as plt

(train\_images, \_), (\_, \_) = tf.keras.datasets.mnist.load\_data()

train\_images = (train\_images - 127.5) / 127.5

train\_images = np.expand\_dims(train\_images, axis=-1)

BUFFER\_SIZE = 60000

BATCH\_SIZE = 256

EPOCHS = 20

NOISE\_DIM = 100

train\_dataset = tf.data.Dataset.from\_tensor\_slices(train\_images).shuffle(BUFFER\_SIZE).batch(BATCH\_SIZE)

def build\_generator():

    model = tf.keras.Sequential([

        Dense(7 \* 7 \* 256, use\_bias=False, input\_shape=(NOISE\_DIM,)),

        BatchNormalization(),

        LeakyReLU(),

        Reshape((7, 7, 256)),

        Conv2DTranspose(128, (5, 5), strides=(1, 1), padding='same', use\_bias=False),

        BatchNormalization(),

        LeakyReLU(),

        Conv2DTranspose(64, (5, 5), strides=(2, 2), padding='same', use\_bias=False),

        BatchNormalization(),

        LeakyReLU(),

        Conv2DTranspose(1, (5, 5), strides=(2, 2), padding='same', use\_bias=False, activation='tanh')

    ])

    return model

def build\_discriminator():

    model = tf.keras.Sequential([

        Conv2D(64, (5, 5), strides=(2, 2), padding='same', input\_shape=[28, 28, 1]),

        LeakyReLU(),

        Conv2D(128, (5, 5), strides=(2, 2), padding='same'),

        LeakyReLU(),

        Flatten(),

        Dense(1, activation='linear')

    ])

    return model

generator = build\_generator()

discriminator = build\_discriminator()

mse\_loss = tf.keras.losses.MeanSquaredError()

generator\_optimizer = Adam(1e-4)

discriminator\_optimizer = Adam(1e-4)

def generator\_loss(fake\_output):

    return mse\_loss(tf.ones\_like(fake\_output), fake\_output)

def discriminator\_loss(real\_output, fake\_output):

    real\_loss = mse\_loss(tf.ones\_like(real\_output), real\_output)

    fake\_loss = mse\_loss(tf.zeros\_like(fake\_output), fake\_output)

    return real\_loss + fake\_loss

@tf.function

def train\_step(images):

    noise = tf.random.normal([BATCH\_SIZE, NOISE\_DIM])

    with tf.GradientTape() as gen\_tape, tf.GradientTape() as disc\_tape:

        generated\_images = generator(noise, training=True)

        real\_output = discriminator(images, training=True)

        fake\_output = discriminator(generated\_images, training=True)

        gen\_loss = generator\_loss(fake\_output)

        disc\_loss = discriminator\_loss(real\_output, fake\_output)

    gradients\_of\_generator = gen\_tape.gradient(gen\_loss, generator.trainable\_variables)

    gradients\_of\_discriminator = disc\_tape.gradient(disc\_loss, discriminator.trainable\_variables)

    generator\_optimizer.apply\_gradients(zip(gradients\_of\_generator, generator.trainable\_variables))

    discriminator\_optimizer.apply\_gradients(zip(gradients\_of\_discriminator, discriminator.trainable\_variables))

    return gen\_loss, disc\_loss

def train(dataset, epochs):

    gen\_losses, disc\_losses = [], []

    for epoch in range(epochs):

        epoch\_gen\_loss = 0

        epoch\_disc\_loss = 0

        for image\_batch in dataset:

            gen\_loss, disc\_loss = train\_step(image\_batch)

            epoch\_gen\_loss += gen\_loss

            epoch\_disc\_loss += disc\_loss

        gen\_losses.append(epoch\_gen\_loss / len(dataset))

        disc\_losses.append(epoch\_disc\_loss / len(dataset))

        print(f'Epoch {epoch+1}, Gen Loss: {gen\_losses[-1]}, Disc Loss: {disc\_losses[-1]}')

    return gen\_losses, disc\_losses

gen\_losses, disc\_losses = train(train\_dataset, EPOCHS)

plt.figure(figsize=(10, 5))

plt.plot(gen\_losses, label="Генератор")

plt.plot(disc\_losses, label="Дискримінатор")

plt.xlabel("Епоха")

plt.ylabel("Втрата")

plt.legend()

plt.show()

def generate\_images\_for\_date(date\_str):

    noise = tf.random.normal([len(date\_str), NOISE\_DIM])

    generated\_images = generator(noise, training=False)

    plt.figure(figsize=(15, 5))

    for i, char in enumerate(date\_str):

        plt.subplot(1, len(date\_str), i + 1)

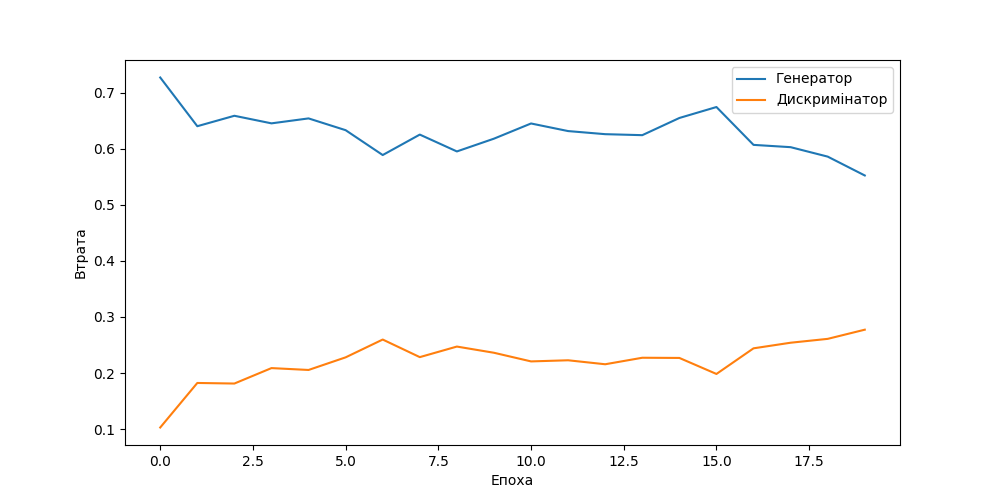
        plt.imshow(generated\_images[i, :, :, 0] \* 127.5 + 127.5, cmap='gray')

        plt.axis('off')

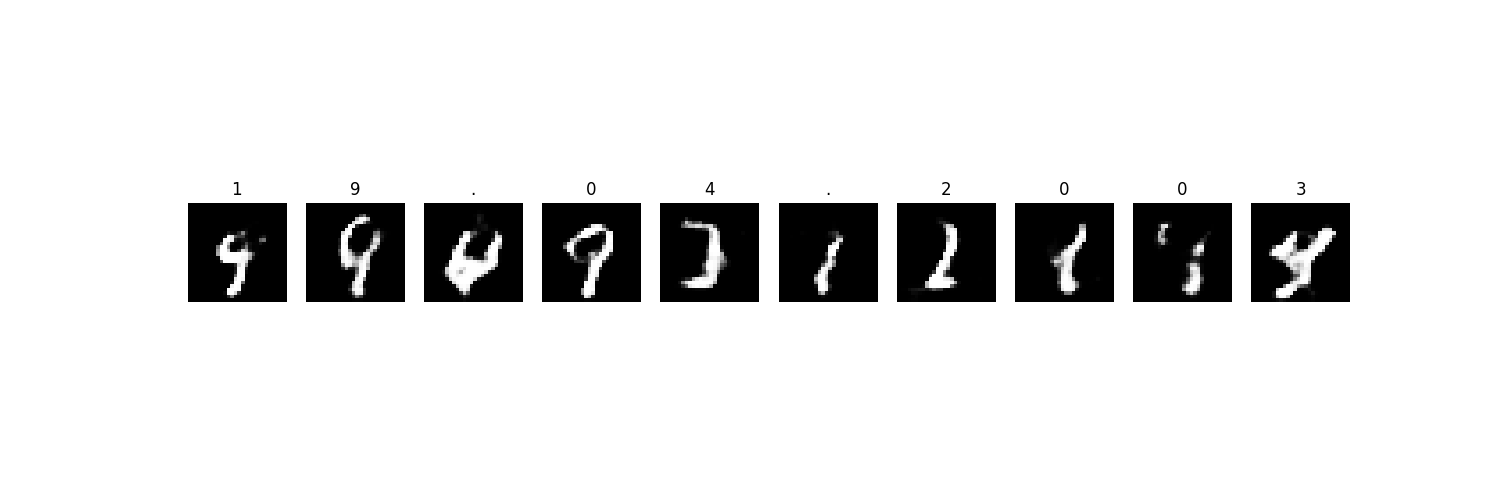
        plt.title(char)

    plt.show()

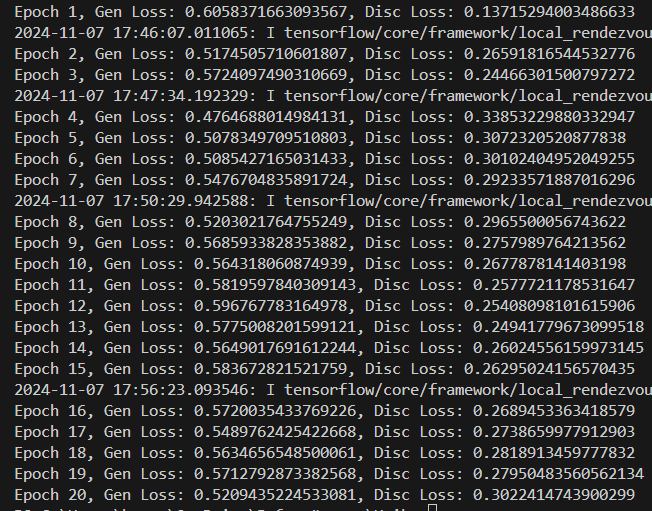
generate\_images\_for\_date("19.04.2003")



Form 1



Form 2



**Висновки**

LSGAN показала себе як стабільний і ефективний метод для генерації зображень. На відміну від традиційних GAN, цей підхід дозволяє уникнути проблеми зникнення градієнта, що сприяє кращому та більш стабільному навчанню генеративної моделі. Розроблена модель здатна генерувати правдоподібні зображення рукописних цифр, що робить її перспективною для подальшого використання в задачах генерації рукописних зображень.

Таким чином, проведене дослідження підтверджує доцільність використання методу LSGAN для завдань генерації зображень та дозволяє рекомендувати цей метод для розв’язання схожих задач у майбутньому