ДОДАТОК А

Приклади програмних кодів

Фрагмент коду на Python, що реалізує модель, описану у підрозділі 3.2. Фрагмент включає побудову, навчання та демонстрацію результатів роботи моделі.

```
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.datasets import mnist
from tensorflow.keras.layers import Input, Dense, Reshape,
Flatten, Conv2D, Conv2DTranspose
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping
(x_train, _), (x_test, _) = mnist.load_data()

x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0
x_train = np.expand_dims(x_train, axis=-1)
x_test = np.expand_dims(x_test, axis=-1)
noise_factor = 0.5
x_train_noisy = x_train + noise_factor *
np.random.normal(loc=0.0, scale=1.0, size=x_train.shape)
x_test_noisy = x_test + noise_factor *
np.random.normal(loc=0.0, scale=1.0, size=x_test.shape)
x_train_noisy = np.clip(x_train_noisy, 0., 1.)
x_test_noisy = np.clip(x_test_noisy, 0., 1.)
input_shape = (28, 28, 1)
latent_dim = 128

# Encoder
inputs = Input(shape=input_shape)
x = Conv2D(32, kernel_size=3, strides=2,
activation='relu', padding='same')(inputs)
x = Conv2D(64, kernel_size=3, strides=2,
activation='relu', padding='same')(x)
x = Flatten()(x)
latent_repr = Dense(latent_dim)(x)

# Decoder
```

```
x = Dense(7 * 7 * 64)(latent_repr)
x = Reshape((7, 7, 64))(x)
x = Conv2DTranspose(32, kernel_size=3, strides=2,
activation='relu', padding='same')(x)
decoded = Conv2DTranspose(1, kernel_size=3, strides=2,
activation='sigmoid', padding='same')(x)

# Autoencoder model
autoencoder = Model(inputs, decoded)
autoencoder.compile(optimizer=Adam(lr=0.0002), loss='mse')
early_stopping = EarlyStopping(monitor='val_loss',
patience=5, restore_best_weights=True)
autoencoder.summary()
epochs = 10
batch_size = 128

history = autoencoder.fit(x_train_noisy, x_train,
validation_data=(x_test_noisy, x_test),
                          epochs=epochs,
batch_size=batch_size, callbacks=[early_stopping])
denoised_test_images = autoencoder.predict(x_test_noisy)

# Display original, noisy, and denoised images
n = 10
plt.figure(figsize=(20, 4))
for i in range(n):
    # Original images
    ax = plt.subplot(3, n, i + 1)
    plt.imshow(x_test[i].reshape(28, 28), cmap='gray')
    plt.title("Original")
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

    # Noisy images
    ax = plt.subplot(3, n, i + 1 + n)
    plt.imshow(x_test_noisy[i].reshape(28, 28),
cmap='gray')
    plt.title("Noisy")
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

    # Denoised images
    ax = plt.subplot(3, n, i + 1 + n + n)
```

```
    plt.imshow(denoised_test_images[i].reshape(28, 28),
cmap='gray')
    plt.title("Denoised")
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
plt.show()
```

Фрагмент коду на Python, що реалізує модель, описану у підрозділі 3.3. Фрагмент включає побудову, навчання та демонстрацію результатів роботи моделі

```
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.datasets import mnist
from tensorflow.keras.layers import Input, Dense, Reshape,
Flatten, Conv2D, Conv2DTranspose
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping
(x_train, _), (x_test, _) = mnist.load_data()

x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0
x_train = np.expand_dims(x_train, axis=-1)
x_test = np.expand_dims(x_test, axis=-1)
noise_factor = 0.5
x_train_noisy = x_train + noise_factor *
np.random.normal(loc=0.0, scale=1.0, size=x_train.shape)
x_test_noisy = x_test + noise_factor *
np.random.normal(loc=0.0, scale=1.0, size=x_test.shape)
x_train_noisy = np.clip(x_train_noisy, 0., 1.)
x_test_noisy = np.clip(x_test_noisy, 0., 1.)
import tensorflow

input_data = tensorflow.keras.layers.Input(shape=(28, 28,
1))

encoder = tensorflow.keras.layers.Conv2D(64, (5,5),
activation='relu')(input_data)

encoder =
tensorflow.keras.layers.MaxPooling2D((2,2))(encoder)
```

```
encoder = tensorflow.keras.layers.Conv2D(128, (3,3),
activation='relu')(encoder)

encoder =
tensorflow.keras.layers.MaxPooling2D((2,2))(encoder)

encoder = tensorflow.keras.layers.Conv2D(256, (3,3),
activation='relu')(encoder)

encoder =
tensorflow.keras.layers.MaxPooling2D((2,2))(encoder)

decoder = tensorflow.keras.layers.Conv2DTranspose(256,
(3,3), activation='relu')(encoder)

decoder = tensorflow.keras.layers.Conv2DTranspose(128,
(3,3), activation='relu')(decoder)

decoder =
tensorflow.keras.layers.UpSampling2D((2,2))(decoder)

decoder = tensorflow.keras.layers.Conv2DTranspose(64,
(3,3), activation='relu')(decoder)

decoder =
tensorflow.keras.layers.UpSampling2D((2,2))(decoder)

decoded = tensorflow.keras.layers.Conv2DTranspose(1,
(5,5), activation='relu')(decoder)

autoencoder =
tensorflow.keras.models.Model(inputs=input_data,
outputs=decoded)
autoencoder.compile(loss='mse', optimizer='adam')
autoencoder.summary()

autoencoder.fit(x_train_noisy, x_train, epochs=10,
batch_size=64, validation_data=(x_test_noisy, x_test))
denoised_test_images = autoencoder.predict(x_test_noisy)

# Display original, noisy, and denoised images
n = 10
plt.figure(figsize=(20, 4))
```

```
for i in range(n):
    # Original images
    ax = plt.subplot(3, n, i + 1)
    plt.imshow(x_test[i].reshape(28, 28), cmap='gray')
    plt.title("Original")
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

    # Noisy images
    ax = plt.subplot(3, n, i + 1 + n)
    plt.imshow(x_test_noisy[i].reshape(28, 28),
cmap='gray')
    plt.title("Noisy")
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

    # Denoised images
    ax = plt.subplot(3, n, i + 1 + n + n)
    plt.imshow(denoised_test_images[i].reshape(28, 28),
cmap='gray')
    plt.title("Denoised")
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
plt.show()
```

Фрагмент коду на Python, що реалізує модель, описану у підрозділі 3.4. Фрагмент включає побудову, навчання та демонстрацію результатів роботи моделі.

```
import numpy as np
import matplotlib.pyplot as plt

from tensorflow.keras.datasets import mnist

(trainX, trainy), (testX, testy) = mnist.load_data()

print('Training data shapes: X=%s, y=%s' % (trainX.shape,
trainy.shape))
print('Testing data shapes: X=%s, y=%s' % (testX.shape,
testy.shape))

for j in range(5):
```

```
    i = np.random.randint(0, 10000)
    plt.subplot(550 + 1 + j)
    plt.imshow(trainX[i], cmap='gray')
    plt.title(trainy[i])
plt.show()
# Adding Noise to the dataset
def guassian_noise(image):
    r,c= image.shape
    mean = 0
    var = 0.1
    sigma = var**0.5
    gaussian = np.random.normal(mean,sigma,(r,c))
    gaussian = gaussian.reshape(r,c)
    noisy = image + gaussian
    return noisy

def salt_and_pepper_noise(image):
    ratio = 0.9
    amount = 0.1
    noisy = np.copy(image)

    salt_count = np.ceil(amount * image.size * ratio)
    coords = [np.random.randint(0, i - 1, int(salt_count))
for i in image.shape]
    noisy[coords] = 1

    pepper_count = np.ceil(amount* image.size * (1. -
ratio))
    coords = [np.random.randint(0, i - 1,
int(pepper_count)) for i in image.shape]
    noisy[coords] = 0
    return noisy

def poisson_noise(image):
    vals = len(np.unique(image))
    vals = 2 ** np.ceil(np.log2(vals))
    noisy = np.random.poisson(image * vals) / float(vals)
    return noisy

def speckle_noise(image):
    r,c = image.shape
    speckle = np.random.randn(r,c)
    speckle = speckle.reshape(r,c)
    noisy = image + image * speckle
```

```
    return noisy

def add_noise(image):
    p = np.random.random()
    if p <= 0.25:
        print("Guassian")
        noisy = guassian_noise(image)
    elif p <= 0.5:
        print("SnP")
        noisy = salt_and_pepper_noise(image)
    elif p <= 0.75:
        print("Poison")
        noisy = poisson_noise(image)
    else:
        print("speckle")
        noisy = speckle_noise(image)
    return noisy
print ("Corrupted Example Samples")
for j in range(9):
    i = np.random.randint(0, 10000)
    plt.subplot(330 + 1 + j)
    noisy = add_noise(trainX[i]/255)
    plt.imshow(noisy, cmap='gray')
plt.show()
train_clean = [image/255 for image in trainX]
test_clean = [image/255 for image in testX]

train_noisy = [add_noise(image/255) for image in trainX]
test_noisy = [add_noise(image/255) for image in testX]

train_clean = np.reshape(train_clean, (60000, 28, 28, 1))
test_clean = np.reshape(test_clean, (10000, 28, 28, 1))

train_noisy = np.reshape(train_noisy, (60000, 28, 28, 1))
test_noisy = np.reshape(test_noisy, (10000, 28, 28, 1))

print (train_clean.shape, train_noisy.shape,
test_clean.shape, test_noisy.shape)
import tensorflow

input_data = tensorflow.keras.layers.Input(shape=(28, 28,
1))
```

```
encoder = tensorflow.keras.layers.Conv2D(64, (5,5),
activation='relu')(input_data)
encoder =
tensorflow.keras.layers.MaxPooling2D((2,2))(encoder)

encoder = tensorflow.keras.layers.Conv2D(128, (3,3),
activation='relu')(encoder)

encoder =
tensorflow.keras.layers.MaxPooling2D((2,2))(encoder)

encoder = tensorflow.keras.layers.Conv2D(256, (3,3),
activation='relu')(encoder)

encoder =
tensorflow.keras.layers.MaxPooling2D((2,2))(encoder)

decoder = tensorflow.keras.layers.Conv2DTranspose(256,
(3,3), activation='relu')(encoder)

decoder = tensorflow.keras.layers.Conv2DTranspose(128,
(3,3), activation='relu')(decoder)

decoder =
tensorflow.keras.layers.UpSampling2D((2,2))(decoder)

decoder = tensorflow.keras.layers.Conv2DTranspose(64,
(3,3), activation='relu')(decoder)

decoder =
tensorflow.keras.layers.UpSampling2D((2,2))(decoder)

decoded = tensorflow.keras.layers.Conv2DTranspose(1,
(5,5), activation='relu')(decoder)

autoencoder =
tensorflow.keras.models.Model(inputs=input_data,
outputs=decoded)
autoencoder.compile(loss='mse', optimizer='adam')
autoencoder.summary()

autoencoder.fit(train_noisy, train_clean, epochs=10,
batch_size=64, validation_data=(test_noisy, test_clean))
offset=92
```

```
print("Noisy test images")
for i in range(9):
    plt.subplot(330 + 1 + i)
    plt.imshow(test_noisy[i+offset,:,:, -1], cmap='gray')
plt.show()

# Reconstructed Images
print ("Cleaned Version(Denoising Autoencoder):) ")
for i in range(9):
    plt.subplot(330 + 1 + i)
    output =
autoencoder.predict(np.array([test_noisy[i+offset]]))
    op_image = np.reshape(output[0]*255, (28, 28))
    plt.imshow(op_image, cmap='gray')
plt.show()
```

ДОДАТОК Б

Відомість кваліфікаційної роботи

Позначення	Найменування	Дод. відомості
	Текстові документи	
1.	Пояснювальна записка	54 с.
	Інші документи	
2.	Презентаційні матеріали	__ плакатів

.	.	Прізвище та ініціали.	Підп.	Дата			
Розробив		Марченко М.Є.			Дослідження використання автокодувальників для знешумлення зображень	Шифр групи	Код напр./спец.
Перевірив		Попов С.В.				СШІм-22-2	122
Н.контр.		Малєєва І.А.				ХНУРЕ кафедра ШІ	
Затв.		Філатов В.О.					