МІНІСТЕРСТВО ОСВІТИ І НАУКИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»



**Лабораторна робота №3**

**З дисципліни «Обробка зображень методами штучного інтелекту»**

***Виконав:*** *студент групи КН-408*

*Черещук Любомир*

***Викладач:***

Пелешко Д. Д.

Львів – 2022

**Тема роботи**: Класифікація зображень. Застосування нейромереж для пошуку подібних зображень.

**Мета роботи**: Набути практичних навиків у розв’язанні задачі пошуку подібних зображень на прикладі організації CNN класифікації.

**Теоретичні відомості.**

В основі класифікації (для пошуку подібних ) зображень пропонується використовувати Siamese networks. Ідея складається в тому щоб взяти випадково ініціалізовану мережу і застосувати її до зображень, щоб дізнатися наскільки вони схожі. Модель має значно полегшати виконання таких задач, як візуальний пошук по базі даних зображень, так як вона буде мати просту метрику подібності між 0 та 1 замість 2D масивів.

Генерація батчів

Ідея полягає в тому, щоб зробити батчі для навчання мережі для прискорення процесу навчання з мінімізацією втрат по якості. Для цього потрібно створити паралельні входи для зображень A і B, де виходом є відстань. Припускаємо, що якщо зображення знаходяться в одній групі, то їх схожість дорівнює 1, в іншому випадку - 0. Якщо випадковим чином вибрати усі зображення, то, швидше за все, отримаємо більшість зображень в різних групах.

**Хід роботи**

Варіант 10 (номер в списку групи – 32).

Побудувати CNN на основі ResNeXt-50 для класифікації зображень на основі датасету fashion-mnist. Зробити налаштування моделі для досягнення необхідної точності. На базі Siamese networks побудувати систему для пошуку подібних зображень в датасеті fashion-mnist. Візуалізувати отримані результати t-SNE.

**Код програми:**

# 10 variant Liubomyr Chereshchuk

import numpy as np

import tensorflow as tf

import skimage.transform

import matplotlib.pyplot as plt

from keras.datasets import fashion\_mnist

(x\_train, y\_train), (x\_test, y\_test) = fashion\_mnist.load\_data()

x\_train = np.array([skimage.transform.resize(image, (32, 32)) for image in np.expand\_dims(x\_train, -1)])[:10000]

x\_test = np.array([skimage.transform.resize(image, (32, 32)) for image in np.expand\_dims(x\_test, -1)])

y\_train = y\_train.astype('int')[:10000]

y\_test = y\_test.astype('int')

train\_groups = [x\_train[np.where(y\_train==i)[0]] for i in np.unique(y\_train)]

test\_groups = [x\_test[np.where(y\_test==i)[0]] for i in np.unique(y\_train)]

print('train groups:', [x.shape[0] for x in train\_groups])

print('test groups:', [x.shape[0] for x in test\_groups])

def gen\_random\_batch(in\_groups, batch\_halfsize = 8):

out\_img\_a, out\_img\_b, out\_score = [], [], []

all\_groups = list(range(len(in\_groups)))

for match\_group in [True, False]:

group\_idx = np.random.choice(all\_groups, size = batch\_halfsize)

out\_img\_a += [in\_groups[c\_idx][np.random.choice(range(in\_groups[c\_idx].shape[0]))] for c\_idx in group\_idx]

if match\_group:

b\_group\_idx = group\_idx

out\_score += [1] \* batch\_halfsize

else:

# anything but the same group

non\_group\_idx = [np.random.choice([i for i in all\_groups if i != c\_idx]) for c\_idx in group\_idx]

b\_group\_idx = non\_group\_idx

out\_score += [0]\*batch\_halfsize

out\_img\_b += [in\_groups[c\_idx][np.random.choice(range(in\_groups[c\_idx].shape[0]))] for c\_idx in b\_group\_idx]

return np.stack(out\_img\_a,0), np.stack(out\_img\_b,0), np.stack(out\_score,0)

pv\_a, pv\_b, pv\_sim = gen\_random\_batch(train\_groups, 3)

fig, m\_axs = plt.subplots(2, pv\_a.shape[0], figsize = (12, 6))

for c\_a, c\_b, c\_d, (ax1, ax2) in zip(pv\_a, pv\_b, pv\_sim, m\_axs.T):

ax1.imshow(c\_a[:,:,0])

ax1.set\_title('Image A')

ax1.axis('off')

ax2.imshow(c\_b[:,:,0])

ax2.set\_title('Image B\n Similarity: %3.0f%%' % (100\*c\_d))

ax2.axis('off')

!pip install git+https://github.com/qubvel/classification\_models.git

from classification\_models.keras import Classifiers

ResNeXt50, preprocess\_input = Classifiers.get('resnext50')

%%time

model = ResNeXt50(classes = 10, weights = None, input\_shape=(32, 32, 1))

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

history = model.fit(x\_train, y\_train, epochs = 4, batch\_size = 16, validation\_split = 0.2, verbose = 1)

from keras.layers import concatenate

img\_a\_in = tf.keras.layers.Input(shape = x\_train.shape[1:], name = 'ImageA\_Input')

img\_b\_in = tf.keras.layers.Input(shape = x\_train.shape[1:], name = 'ImageB\_Input')

img\_a\_feat = model(img\_a\_in)

img\_b\_feat = model(img\_b\_in)

features = concatenate([img\_a\_feat, img\_b\_feat], name = 'features')

features = tf.keras.layers.Dense(16, activation='relu')(features)

features = tf.keras.layers.BatchNormalization()(features)

features = tf.keras.layers.Activation('relu')(features)

features = tf.keras.layers.Dense(4, activation='relu')(features)

features = tf.keras.layers.BatchNormalization()(features)

features = tf.keras.layers.Activation('relu')(features)

features = tf.keras.layers.Dense(1, activation='sigmoid')(features)

siamese\_model = tf.keras.models.Model(inputs = [img\_a\_feat, img\_b\_feat], outputs = [features], name = 'Siamese\_model')

siamese\_model.compile(optimizer = 'adam', loss = 'binary\_crossentropy', metrics = ['mae'])

siamese\_model.summary()

def show\_model\_output(nb\_examples = 3):

pv\_a, pv\_b, pv\_sim = gen\_random\_batch(test\_groups, nb\_examples)

pred\_sim = siamese\_model.predict([model.predict(pv\_a), model.predict(pv\_b)])

fig, m\_axs = plt.subplots(2, pv\_a.shape[0], figsize = (12, 6))

for c\_a, c\_b, c\_d, p\_d, (ax1, ax2) in zip(pv\_a, pv\_b, pv\_sim, pred\_sim, m\_axs.T):

ax1.imshow(c\_a[:,:,0])

ax1.set\_title(f'Image A\n Actual: { 100 \* c\_d }')

ax1.axis('off')

ax2.imshow(c\_b[:,:,0])

ax2.set\_title(f'Image B\n Predicted: {int( 100 \* p\_d[0] )}')

ax2.axis('off')

return fig

# a completely untrained model

\_ = show\_model\_output()

def siam\_gen(in\_groups, batch\_size = 32):

while True:

pv\_a, pv\_b, pv\_sim = gen\_random\_batch(in\_groups, batch\_size//2)

yield [model.predict(pv\_a), model.predict(pv\_b)], pv\_sim

valid\_a, valid\_b, valid\_sim = gen\_random\_batch(test\_groups, 1024)

loss\_history = siamese\_model.fit(siam\_gen(train\_groups),

steps\_per\_epoch = 500,

validation\_data=([model.predict(valid\_a), model.predict(valid\_b)],

valid\_sim),

epochs = 2,

verbose = True)

\_ = show\_model\_output()

%%time

from sklearn.manifold import TSNE

x\_test\_features = model.predict(x\_test, verbose = True, batch\_size=128)

tsne\_obj = TSNE(n\_components=2,

init='pca',

random\_state=101,

method='barnes\_hut',

n\_iter=500,

verbose=1)

tsne\_features = tsne\_obj.fit\_transform(x\_test\_features)

obj\_categories = [

'T-shirt/top','Trouser','Pullover','Dress',

'Coat','Sandal','Shirt','Sneaker','Bag','Ankle boot'

]

colors = plt.cm.rainbow(np.linspace(0, 1, 10))

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

for c\_group, (c\_color, c\_label) in enumerate(zip(colors, obj\_categories)):

plt.scatter(tsne\_features[np.where(y\_test == c\_group), 0],

tsne\_features[np.where(y\_test == c\_group), 1],

marker='o',

color=c\_color,

linewidth=1,

alpha=0.8,

label=c\_label)

plt.xlabel('Dimension 1')

plt.ylabel('Dimension 2')

plt.title('t-SNE on Testing Samples')

plt.legend(loc='best')

plt.savefig('clothes-dist.png')

plt.show(block=False)

Результат роботи програми:

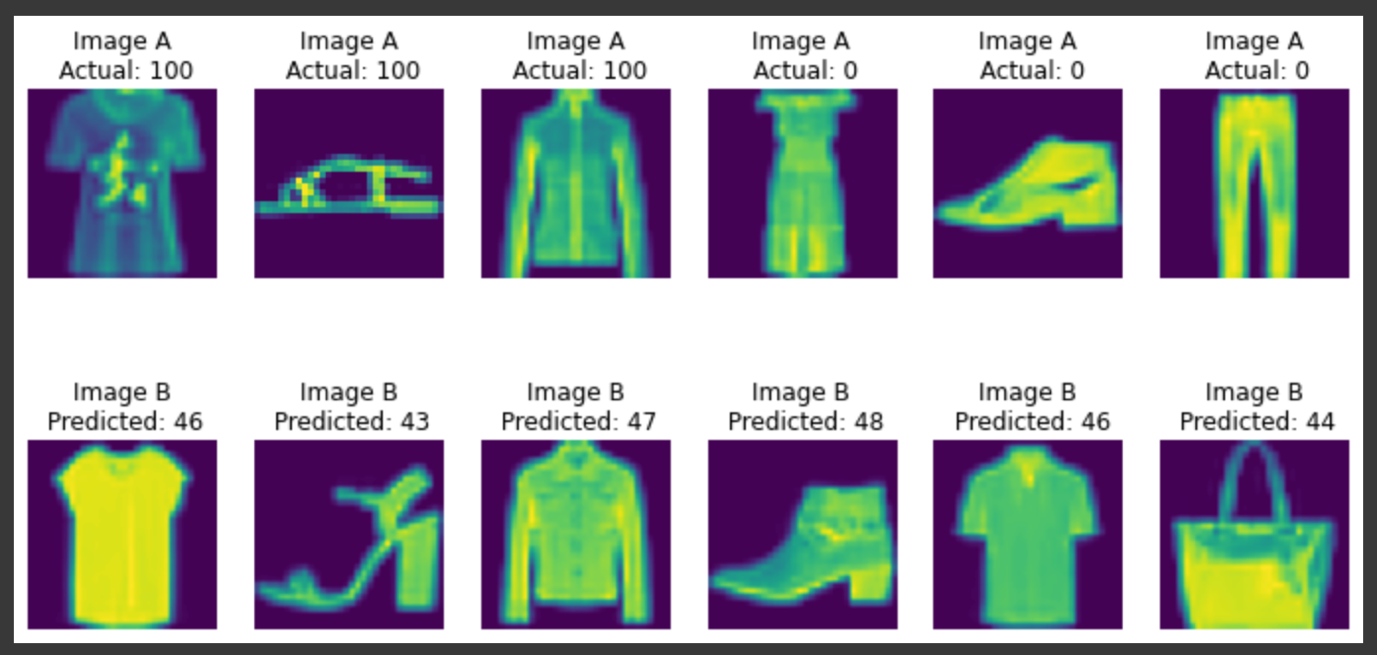


Рис. 1 Результат роботи власної моделі.

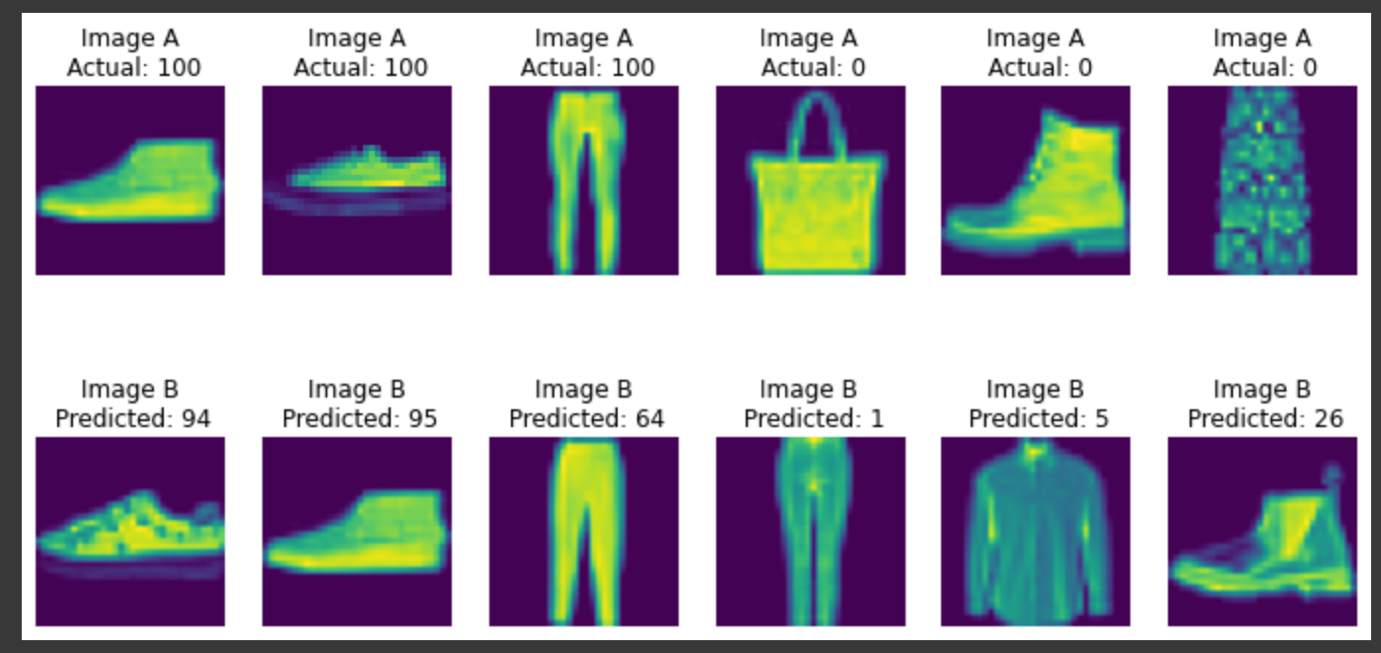


Рис. 2 Результат роботи siam gen.

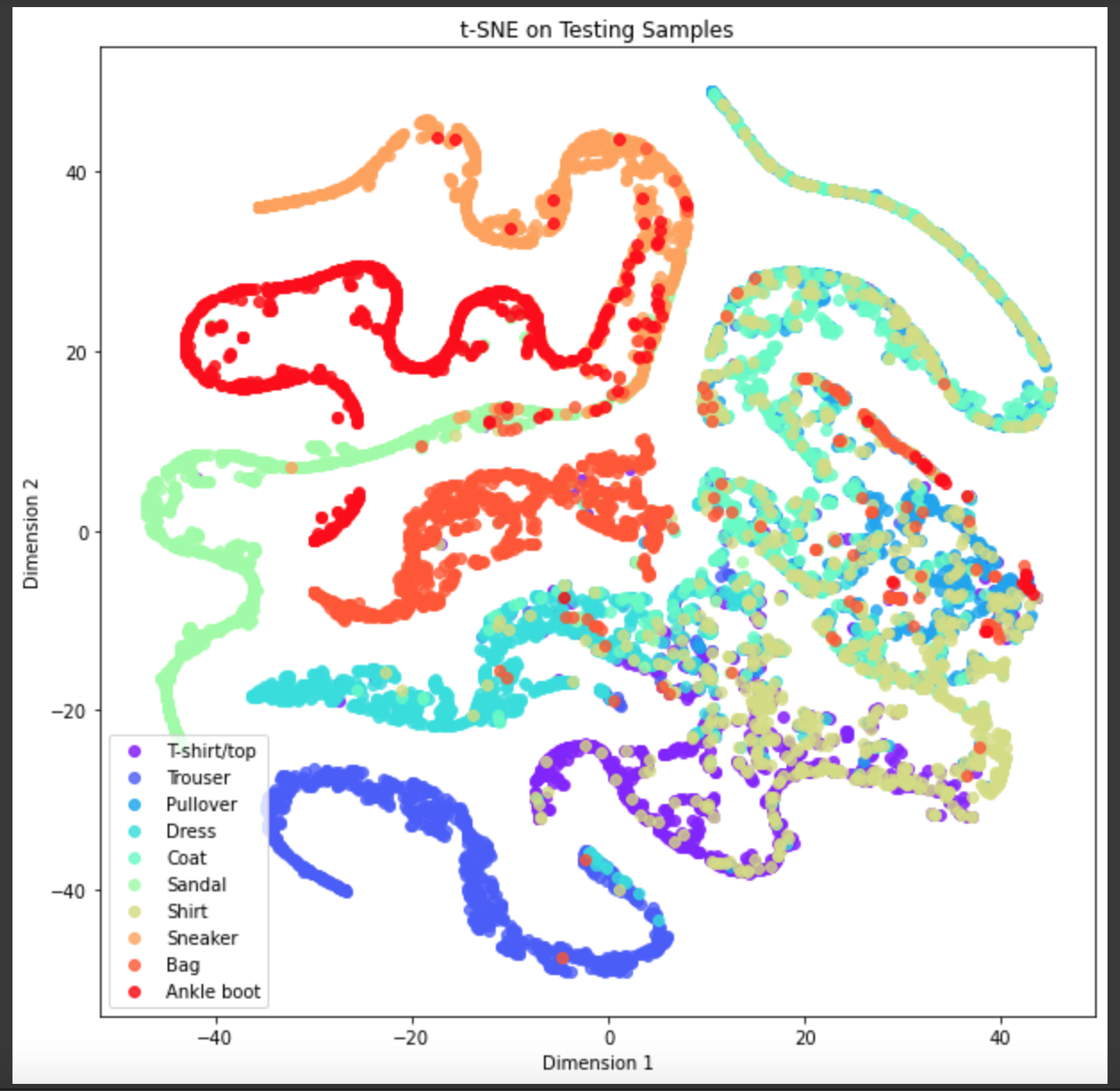


Рис. 3 Візуалізація отриманих результати t-SNE.

**Висновок:** виконавши дану роботу я набув практичних навиків у розв’язанні задачі пошуку подібних зображень на прикладі організації CNN класифікації. Також було застосовано модель ResNeXt-50 для класифікації зображень.