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:

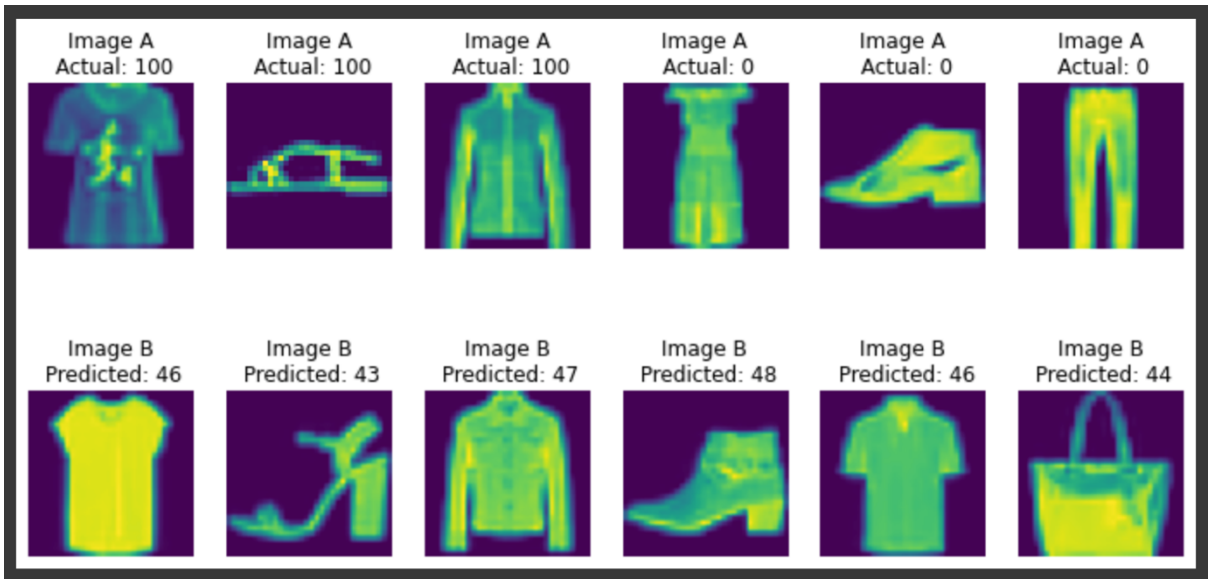

Рис. 1 Результат роботи власної моделі.

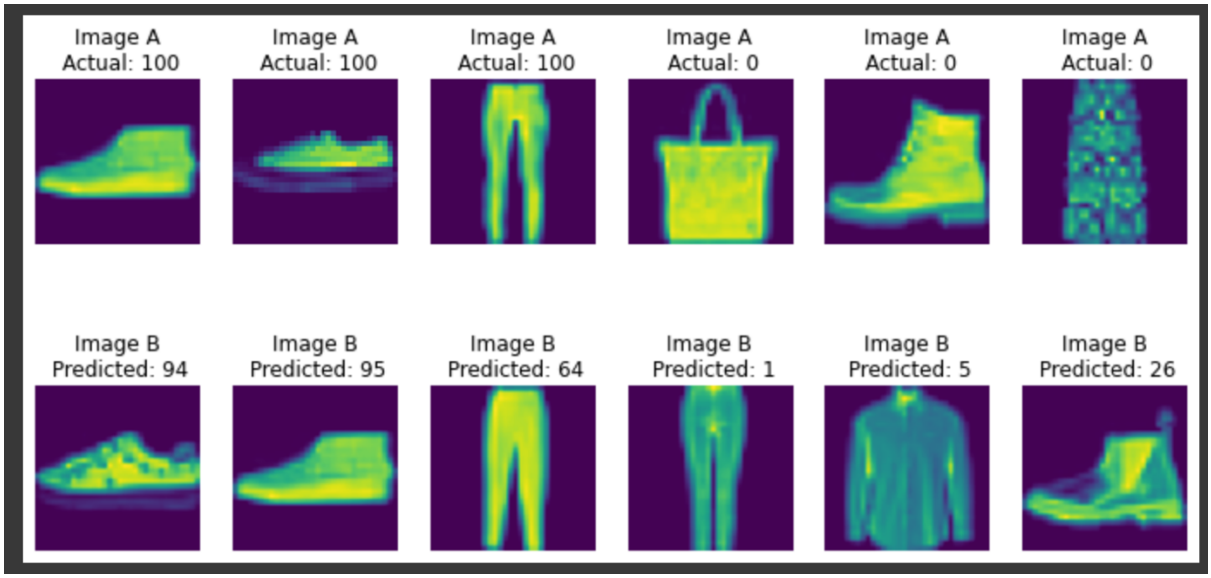

Рис. 2 Результат роботи siam gen.

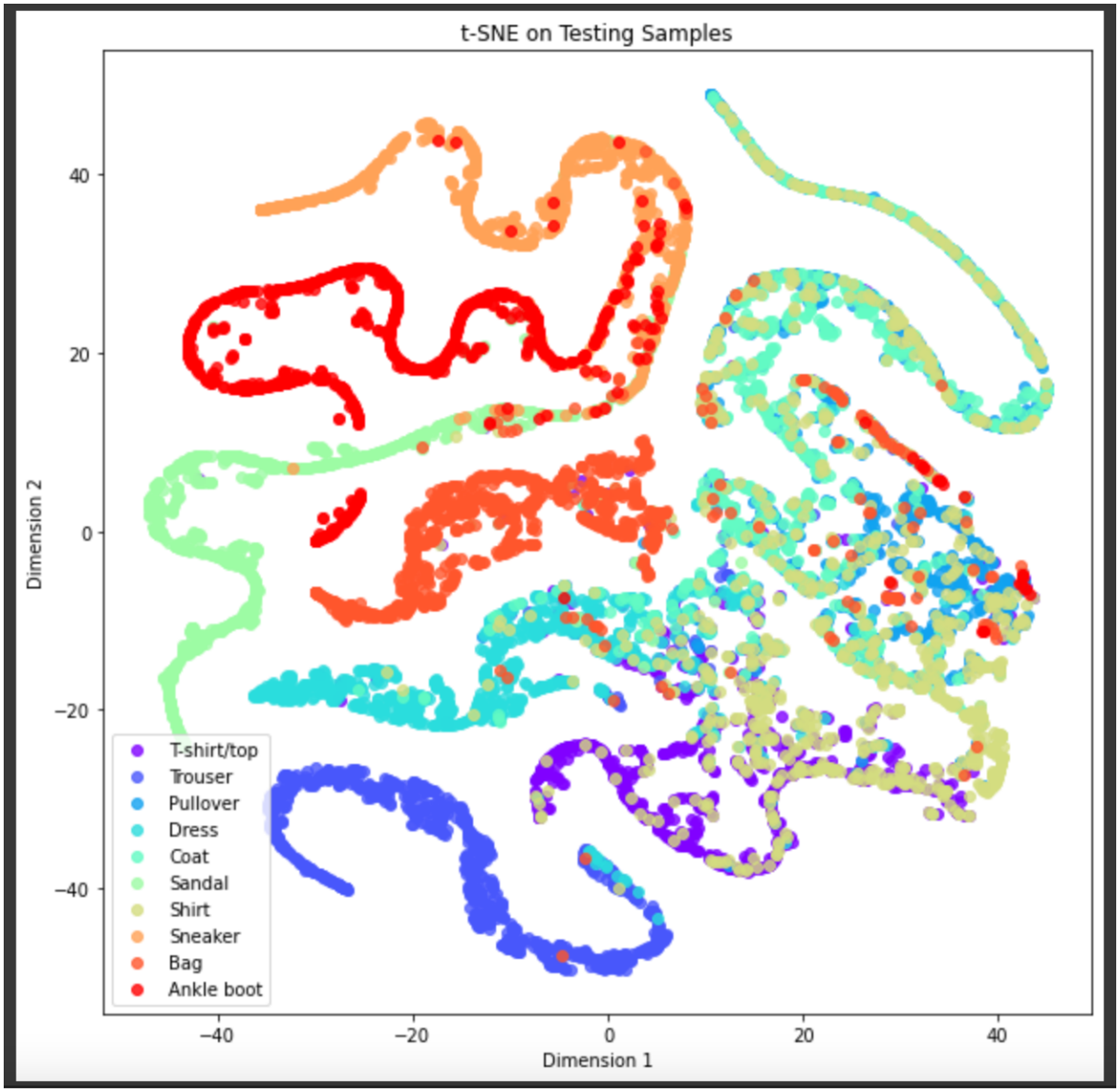

Рис. 3 Візуалізація отриманих результати t-SNE.

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.