**МІНІСТЕРСТВО ОСВІТИ І НАУКИ**

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»**

**Інститут комп'ютерних наук та інформаційних технологій**

**Кафедра систем штучного інтелекту**

****

**Звіт до лабораторної №3**

з дисципліни

“Обробка зображень методами штучного інтелекту”

**Виконав**:

ст. гр. КН-410

Жигайло Ярослав

**Викладач:**

Пелешко Д. Д.

Львів – 2022

**Лабораторна робота №3**

**Класифікація зображень. Застосування нейромереж для пошуку подібних зображень**

**Варіант 8**

**Мета –** набути практичних навіків у розв’язанні задачі пошуку подібних зображень на прикладі організації CNN класифікації.

**Завдання** - Побудувати CNN на основі Inception 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.

**Хід роботи**

Для початку, після проведення підготовки даних, подивимось на зображення, схожість яких потрібно шукати та на ідеальнізовані результати роботи мережі:

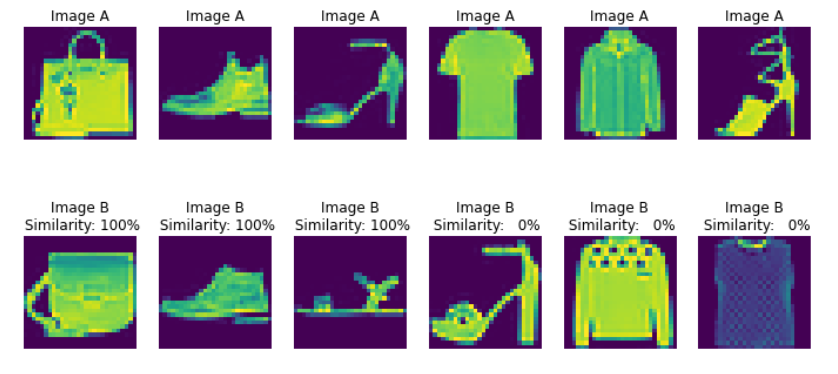


Рис. 1. Ідеальний результат роботи мережі

Далі, поглянемо на роботу не тренованої мережі (рис 2).

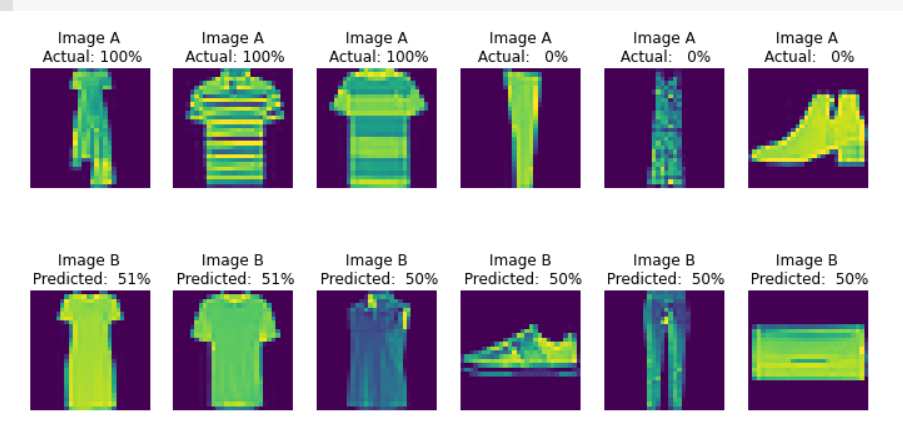


Рис. 2. Результат роботи мережі до тренування

Проведемо тренування та тестування мережі та поглянемо на результати після цього:

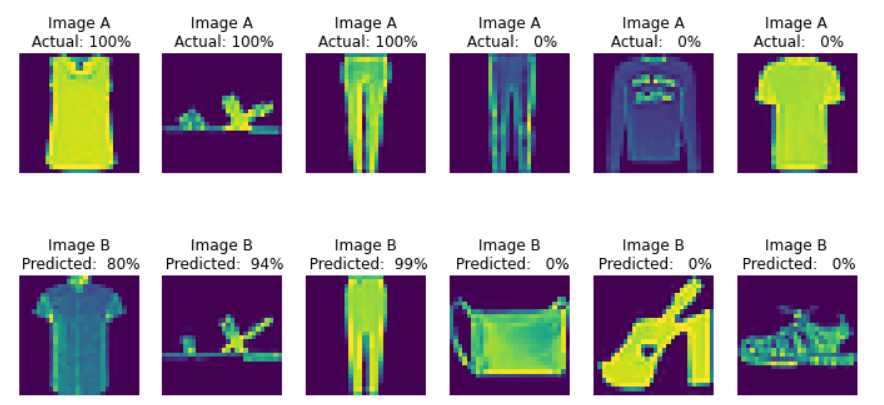


Рис. 3. Результат роботи мережі після тренування

Як бачимо, результати близькі до ідеальних. Тепер, застосуємо TSNE, для зменшення розмірності отриманих векторів фіч, та візуалізуємо отримані кластери (рис 4).

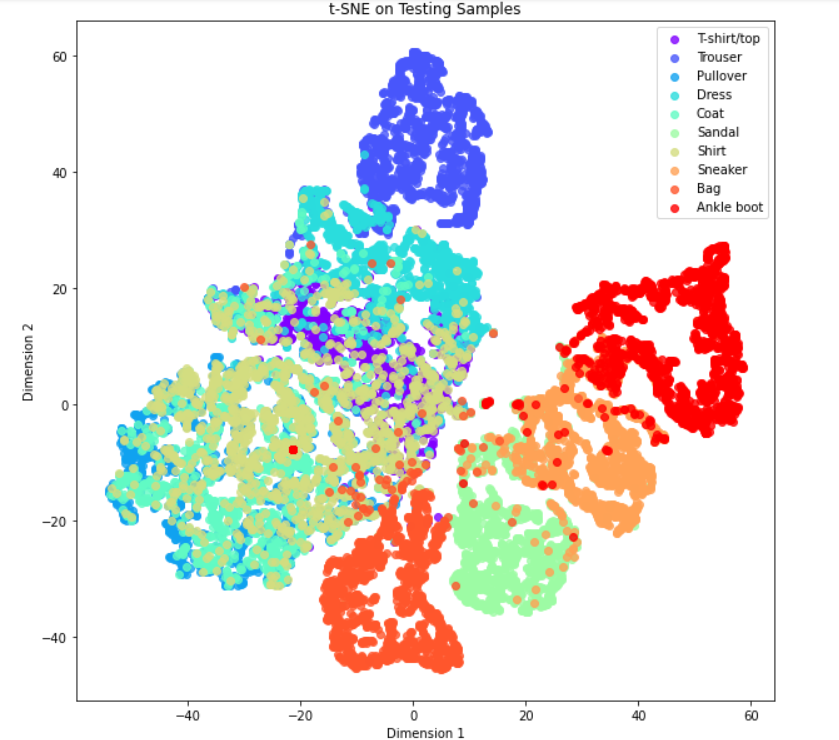


Рис. 4. Візуалізація отриманих кластерів

Як бачимо з рис. 4, наша мережа впоралась доволі непогано та кластеризувала класи зображень.

**Висновок:** у ході виконання даної лабораторної роботи я набув практичних навіків у розв’язанні задачі пошуку подібних зображень на прикладі організації CNN класифікації

**Код програми**

import numpy as np

import os

import pandas as pd

from keras.preprocessing.image import ImageDataGenerator

from keras.utils.np\_utils import to\_categorical

import matplotlib.pyplot as plt

from keras.models import Model

from keras import layers

from keras.layers import Input, Conv2D, BatchNormalization, MaxPool2D, Activation, Flatten, Dense, Dropout

from sklearn.model\_selection import train\_test\_split

data\_train = pd.read\_csv('fashion-mnist\_train.csv')

X\_full = data\_train.iloc[:,1:]

y\_full = data\_train.iloc[:,:1]

x\_train, x\_test, y\_train, y\_test = train\_test\_split(X\_full, y\_full, test\_size

= 0.3)

x\_train = x\_train.values.reshape(-1, 28, 28, 1).astype('float32') / 255

x\_test = x\_test.values.reshape(-1, 28, 28, 1).astype('float32') / 255

y\_train = y\_train.values.astype('int')

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

print('Training', x\_train.shape, x\_train.max())

print('Testing', x\_test.shape, x\_test.max())

# reorganize by groups

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')

def conv2D\_bn\_relu(x, filters, kernel\_size, strides, padding='valid', kernel\_initializer='glorot\_uniform', name=None):

x = layers.Conv2D(filters=filters,

kernel\_size=kernel\_size,

strides=strides,

padding=padding,

kernel\_initializer=kernel\_initializer,

name=name,

use\_bias=False)(x)

x = layers.BatchNormalization(scale=False)(x)

return layers.Activation('relu')(x)

def inception\_module\_A(x, filters=None, kernel\_initializer='glorot\_uniform'):

if filters is None:

filters = int(x.shape[-1])

branch\_filters = filters // 4

b1 = conv2D\_bn\_relu(x,

filters=(branch\_filters // 3) \* 2,

kernel\_size=1,

strides=1,

kernel\_initializer=kernel\_initializer)

b1 = conv2D\_bn\_relu(b1,

filters=branch\_filters,

kernel\_size=3,

strides=1,

padding='same',

kernel\_initializer=kernel\_initializer)

b2 = conv2D\_bn\_relu(x,

filters=(branch\_filters // 3) \* 2,

kernel\_size=1,

strides=1,

kernel\_initializer=kernel\_initializer)

b2 = conv2D\_bn\_relu(b2,

filters=branch\_filters,

kernel\_size=3,

strides=1,

padding='same',

kernel\_initializer=kernel\_initializer)

b2 = conv2D\_bn\_relu(b2,

filters=branch\_filters,

kernel\_size=3,

strides=1,

padding='same',

kernel\_initializer=kernel\_initializer)

b3 = conv2D\_bn\_relu(x,

filters=branch\_filters,

kernel\_size=1,

strides=1,

kernel\_initializer=kernel\_initializer)

pool = layers.AveragePooling2D(pool\_size=(3, 3), strides=1, padding='same')(x)

pool = conv2D\_bn\_relu(pool,

filters=branch\_filters,

kernel\_size=1,

strides=1,

kernel\_initializer=kernel\_initializer)

return layers.concatenate([b1, b2, b3, pool])

def inception\_module\_C(x, filters=None, kernel\_initializer='glorot\_uniform'):

if filters is None:

filters = int(x.shape[-1])

branch\_filters = filters // 6

b1 = conv2D\_bn\_relu(x,

filters=(branch\_filters // 2) \* 3,

kernel\_size=1,

strides=1,

kernel\_initializer=kernel\_initializer)

b1a = conv2D\_bn\_relu(b1,

filters=branch\_filters,

kernel\_size=(1, 3),

strides=1,

padding='same',

kernel\_initializer=kernel\_initializer)

b1b = conv2D\_bn\_relu(b1,

filters=branch\_filters,

kernel\_size=(3, 1),

strides=1,

padding='same',

kernel\_initializer=kernel\_initializer)

b2 = conv2D\_bn\_relu(x,

filters=(branch\_filters // 2) \* 3,

kernel\_size=1,

strides=1,

kernel\_initializer=kernel\_initializer)

b2 = conv2D\_bn\_relu(b2,

filters=(branch\_filters // 4) \* 7,

kernel\_size=(1, 3),

strides=1,

padding='same',

kernel\_initializer=kernel\_initializer)

b2 = conv2D\_bn\_relu(b2,

filters=branch\_filters \* 2,

kernel\_size=(3, 1),

strides=1,

padding='same',

kernel\_initializer=kernel\_initializer)

b2a = conv2D\_bn\_relu(b2,

filters=branch\_filters,

kernel\_size=(1, 3),

strides=1,

padding='same',

kernel\_initializer=kernel\_initializer)

b2b = conv2D\_bn\_relu(b2,

branch\_filters,

kernel\_size=(3, 1),

strides=1,

padding='same',

kernel\_initializer=kernel\_initializer)

b3 = conv2D\_bn\_relu(x,

filters=branch\_filters,

kernel\_size=1,

strides=1,

kernel\_initializer=kernel\_initializer)

pool = layers.AveragePooling2D(pool\_size=(3, 3), strides=1, padding='same')(x)

pool = conv2D\_bn\_relu(pool,

filters=branch\_filters,

kernel\_size=1,

strides=1,

kernel\_initializer=kernel\_initializer)

return layers.concatenate([b1a, b1b, b2a, b2b, b3, pool])

def reduction\_module\_A(x, filters, kernel\_initializer='glorot\_uniform'):

branch\_filters = (filters - int(x.shape[-1])) // 2

b1 = conv2D\_bn\_relu(x,

filters=branch\_filters,

kernel\_size=3,

strides=2,

padding='same',

kernel\_initializer=kernel\_initializer)

b2 = conv2D\_bn\_relu(x,

filters=(branch\_filters // 3) \* 2,

kernel\_size=1,

strides=1,

kernel\_initializer=kernel\_initializer)

b2 = conv2D\_bn\_relu(b2,

filters=(branch\_filters // 6) \* 5,

kernel\_size=3,

strides=1,

padding='same',

kernel\_initializer=kernel\_initializer)

b2 = conv2D\_bn\_relu(b2,

filters=branch\_filters,

kernel\_size=3,

strides=2,

padding='same',

kernel\_initializer=kernel\_initializer)

pool = layers.MaxPooling2D(pool\_size=(3, 3), strides=2, padding='same')(x)

return layers.concatenate([b1, b2, pool])

stem\_width = 128

inputs = layers.Input(shape=x\_train.shape[1:])

x = conv2D\_bn\_relu(inputs,

filters=stem\_width,

kernel\_size=5,

strides=1,

padding='same',

name='conv\_1')

x = reduction\_module\_A(x, filters=int(2\*stem\_width))

x = layers.SpatialDropout2D(0.3)(x)

x = inception\_module\_A(x, filters=int(2\*stem\_width))

x = inception\_module\_A(x, filters=int(2\*stem\_width))

x = reduction\_module\_A(x, filters=int(3\*stem\_width))

x = layers.SpatialDropout2D(0.5)(x)

x = inception\_module\_C(x, filters=int(3\*stem\_width))

x = inception\_module\_C(x, filters=int(3\*stem\_width))

x = layers.GlobalAveragePooling2D()(x)

x = layers.Dropout(0.5)(x)

x = layers.Dense(32)(x)

# x = layers.Activation('softmax', name='softmax')(x)

model = Model(inputs=inputs, outputs=x)

model.summary()

from keras.layers import concatenate

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

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

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

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

combined\_features = concatenate([img\_a\_feat, img\_b\_feat], name =

'merge\_features')

combined\_features = Dense(16, activation = 'linear')(combined\_features)

combined\_features = BatchNormalization()(combined\_features)

combined\_features = Activation('relu')(combined\_features)

combined\_features = Dense(4, activation = 'linear')(combined\_features)

combined\_features = BatchNormalization()(combined\_features)

combined\_features = Activation('relu')(combined\_features)

combined\_features = Dense(1, activation = 'sigmoid')(combined\_features)

similarity\_model = Model(inputs = [img\_a\_in, img\_b\_in], outputs =

[combined\_features], name = 'Similarity\_Model')

similarity\_model.summary()

similarity\_model.compile(optimizer='adam', loss = 'binary\_crossentropy',

metrics = ['mae'])

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

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

pred\_sim = similarity\_model.predict([pv\_a, 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('Image A\n Actual: %3.0f%%' % (100\*c\_d))

ax1.axis('off')

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

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

ax2.axis('off')

return fig

# a completely untrained model

\_ = show\_model\_output()

# make a generator out of the data

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

while True:

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

yield [pv\_a, pv\_b], pv\_sim

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

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

steps\_per\_epoch = 500,

validation\_data=([valid\_a, valid\_b],

valid\_sim),

epochs = 10,

verbose = True)

\_ = show\_model\_output()

x\_test\_features = model.predict(x\_test, verbose = True,

batch\_size=128)

%%time

from sklearn.manifold import TSNE

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

init='pca',

random\_state=101,

method='barnes\_hut',

n\_iter=500,

verbose=2)

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)