Міністерство освіти і науки України

Національний університет «Львівська політехніка»

Інститут комп’ютерних наук та інформаційних технологій

Кафедра «Системи штучного інтелекту»

**Лабораторна робота №3**

# З дисципліни «Обробка зображень методами штучного інтелекту»

Виконав студент групи КН-408 Горностай Б. Я.

Прийняв Пелешко Д. Д.

Львів 2022

Варіант 10

# Завдання

# Побудувати CNN на основі ResNet-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

from keras.layers import Conv2D, Dense, BatchNormalization, Activation, MaxPool2D, GlobalAveragePooling2D, Add, Input

from keras.preprocessing.image import ImageDataGenerator

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

from keras.models import Model

import matplotlib.pyplot as plt

import tensorflow as tf

import pandas as pd

import numpy as np

import os

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

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

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

y\_train = y\_train.astype('int')

y\_test = y\_test.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')

################################################################################

class ResidualBlock(Model):

    def \_\_init\_\_(self, channel\_in = 64, channel\_out = 256):

        super().\_\_init\_\_()

        channel = channel\_out // 4

        self.conv1 = Conv2D(channel, kernel\_size = (1, 1), padding = "same")

        self.bn1 = BatchNormalization()

        self.av1 = Activation(tf.nn.relu)

        self.conv2 = Conv2D(channel, kernel\_size = (3, 3), padding = "same")

        self.bn2 = BatchNormalization()

        self.av2 = Activation(tf.nn.relu)

        self.conv3 = Conv2D(channel\_out, kernel\_size = (1, 1), padding = "same")

        self.bn3 = BatchNormalization()

        self.shortcut = self.\_shortcut(channel\_in, channel\_out)

        self.add = Add()

        self.av3 = Activation(tf.nn.relu)

    def call(self, x):

        h = self.conv1(x)

        h = self.bn1(h)

        h = self.av1(h)

        h = self.conv2(h)

        h = self.bn2(h)

        h = self.av2(h)

        h = self.conv3(h)

        h = self.bn3(h)

        shortcut = self.shortcut(x)

        h = self.add([h, shortcut])

        y = self.av3(h)

        return y

    def \_shortcut(self, channel\_in, channel\_out):

        if channel\_in == channel\_out:

            return lambda x : x

        else:

            return self.\_projection(channel\_out)

    def \_projection(self, channel\_out):

        return Conv2D(channel\_out, kernel\_size = (1, 1), padding = "same")

class ResNet50(Model):

    def \_\_init\_\_(self, input\_shape, output\_dim):

        super().\_\_init\_\_()

        self.\_layers = [

            # conv1

            Conv2D(64, input\_shape = input\_shape, kernel\_size = (7, 7), strides=(2, 2), padding = "same"),

            BatchNormalization(),

            Activation(tf.nn.relu),

            # conv2\_x

            MaxPool2D(pool\_size = (3, 3), strides = (2, 2), padding = "same"),

            ResidualBlock(64, 256),

            [

                ResidualBlock(256, 256) for \_ in range(2)

            ],

            # conv3\_x

            Conv2D(512, kernel\_size = (1, 1), strides=(2, 2)),

            [

                ResidualBlock(512, 512) for \_ in range(4)

            ],

            # conv4\_x

            Conv2D(1024, kernel\_size = (1, 1), strides=(2, 2)),

            [

                ResidualBlock(1024, 1024) for \_ in range(6)

            ],

            # conv5\_x

            Conv2D(2048, kernel\_size = (1, 1), strides=(2, 2)),

            [

                ResidualBlock(2048, 2048) for \_ in range(3)

            ],

            # last part

            GlobalAveragePooling2D(),

            Dense(32)

            # Dense(1000, activation = tf.nn.relu),

            #Dense(output\_dim, activation = tf.nn.softmax)

        ]

    def call(self, x):

        for layer in self.\_layers:

            if isinstance(layer, list):

                for l in layer:

                    x = l(x)

            else:

                x = layer(x)

        return x

model = ResNet50((28, 28, 1), 10)

model.build(input\_shape = (None, 28, 28, 1))

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()

# setup the optimization process

opt = tf.keras.optimizers.Adam(learning\_rate=0.01)

similarity\_model.compile(optimizer=opt, 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()

early\_stopping = tf.keras.callbacks.EarlyStopping(monitor='val\_loss', patience=3)

reduce\_lr = tf.keras.callbacks.ReduceLROnPlateau(monitor='val\_loss', factor=0.2,

                              patience=0, min\_lr=0.0001, verbose=1)

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),

                                    callbacks=[early\_stopping, reduce\_lr],

                                    epochs=10,

                                    verbose=True)

\_ = show\_model\_output()

t\_shirt\_vec = np.stack([train\_groups[0][0]]\*x\_test.shape[0],0)

t\_shirt\_score = similarity\_model.predict([t\_shirt\_vec, x\_test], verbose =

True, batch\_size = 128)

ankle\_boot\_vec = np.stack([train\_groups[-1][0]]\*x\_test.shape[0],0)

ankle\_boot\_score = similarity\_model.predict([ankle\_boot\_vec, x\_test], verbose

= True, batch\_size = 128)

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(t\_shirt\_score[np.where(y\_test == c\_group), 0],

              ankle\_boot\_score[np.where(y\_test == c\_group), 0],

              marker='.',

              color=c\_color,

              linewidth='1',

              alpha=0.8,

              label=c\_label)

plt.xlabel('T-Shirt Dimension')

plt.ylabel('Ankle-Boot Dimension')

plt.title('T-Shirt and Ankle-Boot Dimension')

plt.legend(loc='best')

plt.savefig('tshirt-boot-dist.png')

plt.show(block=False)

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=1000,

                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)

# Результати роботи

Для перегляду результатів щодо подібностей зображень, вибираємо випадкові пари та виводимо їх разом з передбаченим шансом подібності, як бачимо, модель покищо не натренована:

Зображення, що містить текст, екран, встановлений, лінія

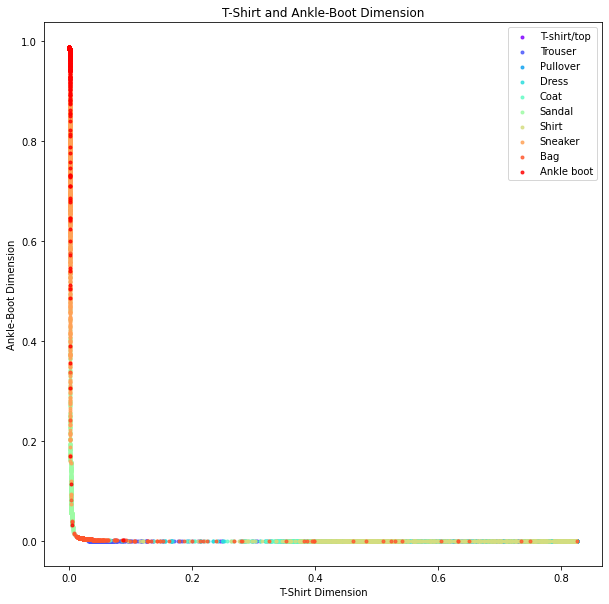
Автоматично згенерований опис

Після тренерування моделі, можемо побачити, що вона відрізняє зображення, однак кількість помилок в негативних відповідях висока, це можна пояснити тим, що ResNet50 э не найбільш підходящою мережею для цієї задачі.

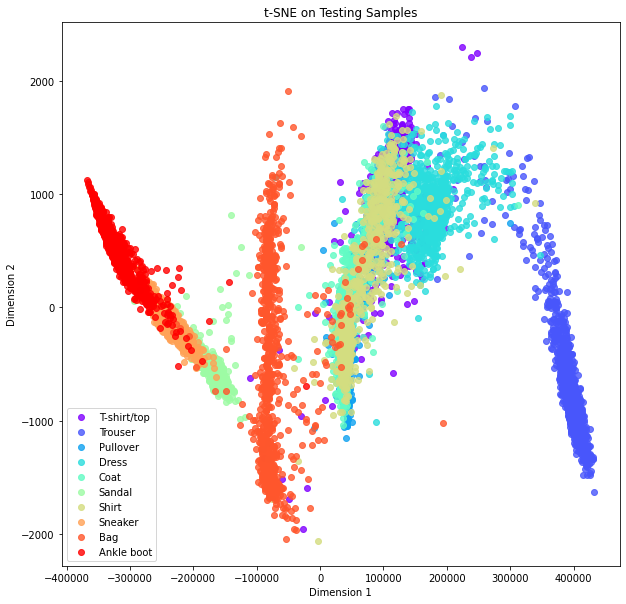
Зображення, що містить текст, екран, монітор, дисплей

Автоматично згенерований опис

Якщо порівняти різницю між різними класами, можемо побачити, що нейронна мережа вказує значну відстань між найбільш різними класами.



Як бачимо, модель знаходить подібними ознаки усіх типів взуття а також знаходить подібними деякі ознаки класів одягу, інші ознаки сильно відрізняються.



# Висновок

Під час виконання цієї лабораторної роботи було створено ембедінги зображень на базі згорткової мережі ResNet-50 та було реалізовано алгоритм визначення подібності за допомогою Siamese моделі.