МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНОМУ УНІВЕРСИТЕТІ “ЛЬВІВСЬКА ПОЛІТЕХНІКА”

Кафедра систем штучного інтелекту

Лабораторна робота № 3

з дисципліни
«Обробка зображень методами штучного інтелекту»

Виконав:
студент групи КН-408
Мокрик Ярослав
Викладач:
Пелешко Д.Д.

Львів – 2022 р.

Тема: Класифікація зображень. Застосування нейромереж для пошуку подібних зображень

Мета: набути практичних навиків у розв'язанні задачі пошуку подібних зображень на прикладі організації CNN класифікації

Варіант 11

Завдання

Побудувати CNN на основі DenceNet 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

```
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
%tensorflow_version 2.x
import tensorflow as tf
import tensorflow_datasets as tfds
fashion_mnist = tf.keras.datasets.fashion_mnist
(x_train, y_train), (x_test, y_test) = 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())
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
```

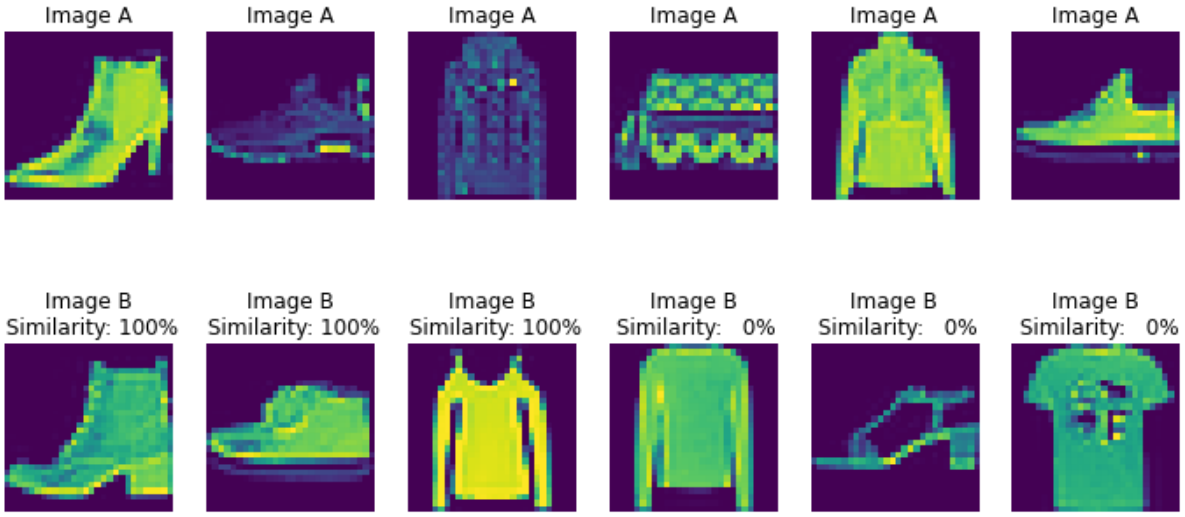
```
      out_img_b += [in_groups[c_idx][np.random.choice(range(in_groups[c_idx].shape[0]))] for c_idx in
b_group_idx]

  return np.stack(out_img_a,0), np.stack(out_img_b,0), np.stack(out_score,0)
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')
from keras.models import Model
from keras.layers import Input, Conv2D, BatchNormalization, MaxPool2D, Activation, Flatten, Dense,
Dropout, SpatialDropout2D, Concatenate, AveragePooling2D
from keras.regularizers import l2
from keras.initializers import he_normal, random_normal
# Hyperparameters
num_classes = 10
l = 12
num_filter = 24
compression = 0.5
dropout_rate = 0.2
wt_decay = 0.001
def add_denseblock(input, num_filter = 12, dropout_rate = 0.2):
  global compression
  temp = input
  for _ in range(l):
    BatchNorm = BatchNormalization()(temp)
    relu = Activation('relu')(BatchNorm)
         Conv2D_3_3  =  Conv2D(int(num_filter*compression),  (3,3),  use_bias=False  ,padding='same',
kernel_regularizer=l2(wt_decay)                                                                    ,
kernel_initializer=(random_normal(stddev=np.sqrt(2.0/(9*int(num_filter))))))(relu)
    Conv2D_3_3 = SpatialDropout2D(dropout_rate)(Conv2D_3_3)
    concat = Concatenate(axis=-1)([temp,Conv2D_3_3])

    temp = concat

  return temp
def add_transition(input, num_filter = 12, dropout_rate = 0.2):
  global compression
```

```
  BatchNorm = BatchNormalization()(input)
  relu = Activation('relu')(BatchNorm)
       Conv2D_BottleNeck   =   Conv2D(int(int(input.shape[-1])*compression),   (1,1),   use_bias=False
,padding='same',kernel_regularizer=l2(wt_decay)                                                    ,
kernel_initializer=(random_normal(stddev=np.sqrt(2.0/(9*int(num_filter*compression))))))(relu)
  Conv2D_BottleNeck = SpatialDropout2D(dropout_rate)(Conv2D_BottleNeck)
  avg = AveragePooling2D(pool_size=(2,2))(Conv2D_BottleNeck)

  return avg
def output_layer(input):
  global compression
  BatchNorm = BatchNormalization()(input)
  relu = Activation('relu')(BatchNorm)
  AvgPooling = AveragePooling2D(pool_size=(2,2))(relu)
  flat = Flatten()(AvgPooling)
  output = Dense(num_classes, activation='softmax', kernel_regularizer=l2(wt_decay))(flat)

  return output
img_in = Input(shape = x_train.shape[1:], name = 'FeatureNet_ImageInput')
First_Conv2D     =     Conv2D(int(num_filter),     (3,3),     use_bias=False     ,     padding='same',
kernel_regularizer=l2(wt_decay),
kernel_initializer=(random_normal(stddev=np.sqrt(2.0/(9*num_filter)))))(img_in)
First_Block = add_denseblock(First_Conv2D, num_filter, dropout_rate)
First_Transition = add_transition(First_Block, num_filter, dropout_rate)
Second_Block = add_denseblock(First_Transition, num_filter, dropout_rate)
Second_Transition = add_transition(Second_Block, num_filter, dropout_rate)
Third_Block = add_denseblock(Second_Transition, num_filter, dropout_rate)
Third_Transition = add_transition(Third_Block, num_filter, dropout_rate)
Last_Block = add_denseblock(Third_Transition,  num_filter, dropout_rate)
Last_Block = output_layer(Last_Block)

feature_model = Model(inputs = [img_in], outputs = [Last_Block], name = 'FeatureGenerationModel')
feature_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 = feature_model(img_a_in)
img_b_feat = feature_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
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')
```

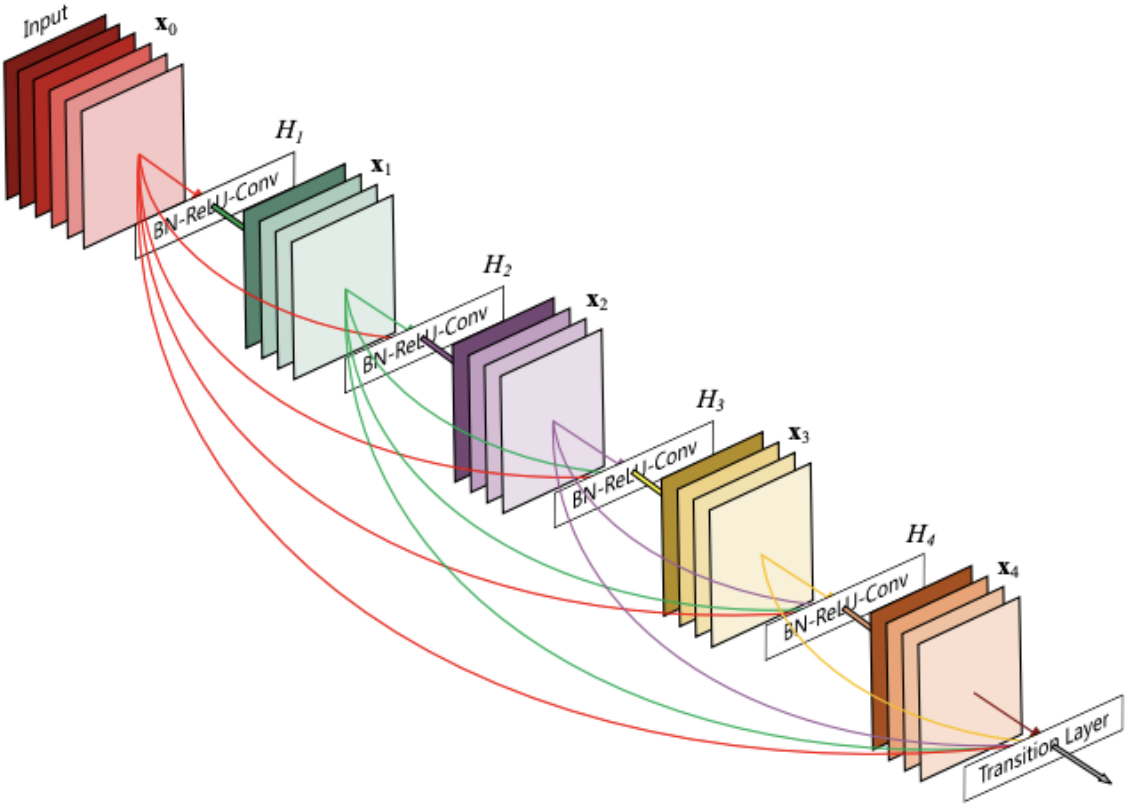
```
  return fig
# a completely untrained model
_ = show_model_output()
```

Результати

Для початку, завантажуємо зображення з датасету fashion-mnist і проводимо препроцесинг даних. Створюємо генератор вибірок зображень, де є N пар зображень однакових класів і N пар зображень різних класів. Для прикладу, перевіряємо як виглядає генерація однієї такої вибірки при N=3:



Далі, будуємо конволюційну мережу на основі моделі DenseNet. У цій моделі є декілька Dense блоків, які містять в собі кілька шарів, кожен з яких з’єднаний з усіма попередніми. Всередині кожного Dense блоку проводиться Batch Normalization, ReLU активація, згортка з розміром ядра 3*3 і Dropout. Всі входи в ці шари всередині Dense блоку конкатенуються між собою, таким чином вирішуючи проблему зникаючих градієнтів. Через те що кожен з цих шарів отримає на вхід всі попередні, розмір самих шарів може бути відносно менший, пришвидшуючи роботу алгоритму без втрати глибини чи точності.



Між кожним Dense блоком є Transition шари, кожен з яких містить шар згортки з розміром ядра 3*3 і шар пулінгу. Побудувавши таку модель в Keras, можемо перевірити її архітектуру:

```
Model: "FeatureGenerationModel"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 FeatureNet_ImageInput (InputLa  [(None, 28, 28, 1)]  0          []
 yer)

 conv2d_1 (Conv2D)              (None, 28, 28, 24)   216         ['FeatureNet_ImageInput[0][0]']

 batch_normalization (BatchNorm  (None, 28, 28, 24)  96          ['conv2d_1[0][0]']
 alization)

 activation (Activation)        (None, 28, 28, 24)   0           ['batch_normalization[0][0]']

 conv2d_2 (Conv2D)              (None, 28, 28, 12)   2592        ['activation[0][0]']

 spatial_dropout2d (SpatialDrop  (None, 28, 28, 12)  0           ['conv2d_2[0][0]']
 out2D)

 concatenate (Concatenate)      (None, 28, 28, 36)   0           ['conv2d_1[0][0]',
                                                                  'spatial_dropout2d[0][0]']

 batch_normalization_1 (BatchNo  (None, 28, 28, 36)  144         ['concatenate[0][0]']
 rmalization)
```

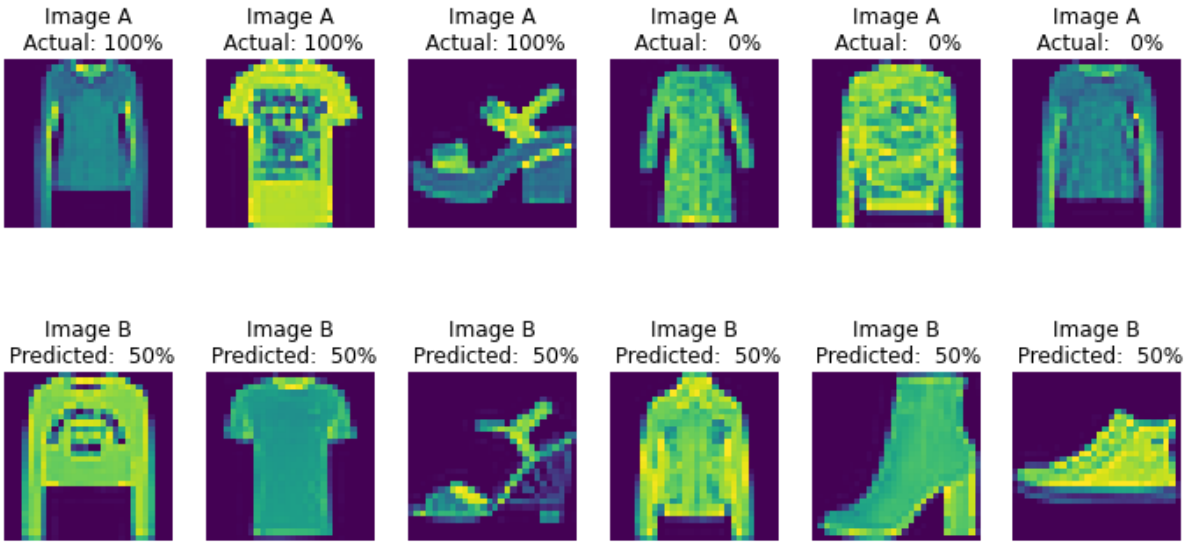
...

```
 flatten (Flatten)              (None, 273)          0           ['average_pooling2d_3[0][0]']

 dense (Dense)                  (None, 10)           2740        ['flatten[0][0]']

==================================================================================================
Total params: 906,610
Trainable params: 889,996
Non-trainable params: 16,614
__________________________________________________________________________________________________
```

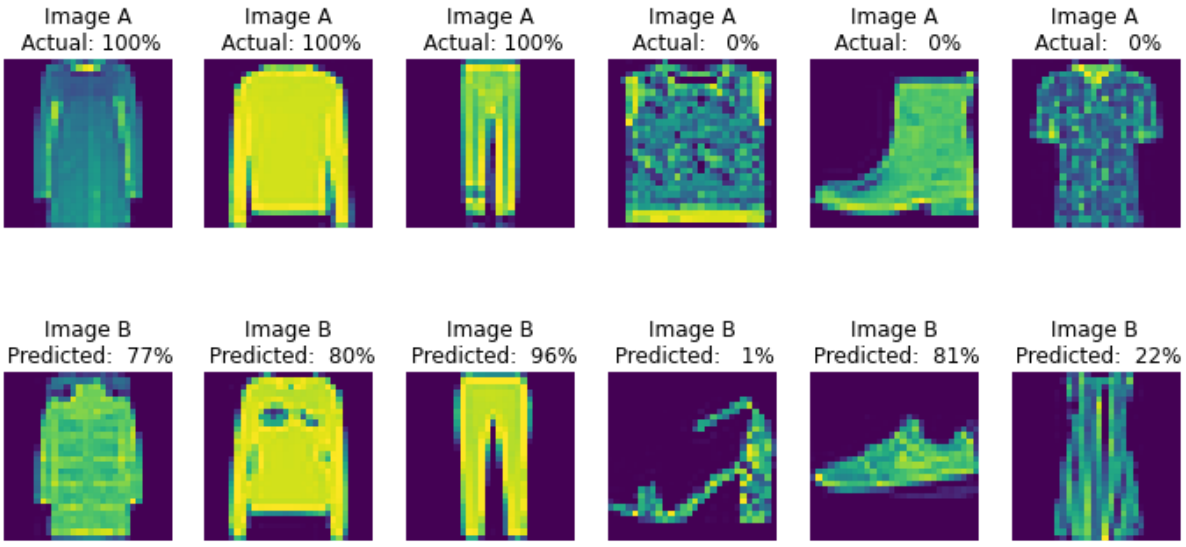
Така модель з чотирма Dense блоками і трьома Transition блоками між ними містить приблизно 900,000 параметрів. Далі, будуємо сіамську модель з двома моделями, які генерують ознаки (DenseNet), параметри яких однакові, тому кількість параметрів не дуже зростає.

```
Model: "Similarity_Model"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 ImageA_Input (InputLayer)      [(None, 28, 28, 1)]  0           []

 ImageB_Input (InputLayer)      [(None, 28, 28, 1)]  0           []

 FeatureGenerationModel (Functi  (None, 10)          906610      ['ImageA_Input[0][0]',
 onal)                                                            'ImageB_Input[0][0]']

 merge_features (Concatenate)   (None, 20)           0           ['FeatureGenerationModel[0][0]',
                                                                  'FeatureGenerationModel[1][0]']

 dense_1 (Dense)                (None, 16)           336         ['merge_features[0][0]']

 batch_normalization_52 (BatchN  (None, 16)          64          ['dense_1[0][0]']
 ormalization)

 activation_52 (Activation)     (None, 16)           0           ['batch_normalization_52[0][0]']

 dense_2 (Dense)                (None, 4)            68          ['activation_52[0][0]']

 batch_normalization_53 (BatchN  (None, 4)           16          ['dense_2[0][0]']
 ormalization)

 activation_53 (Activation)     (None, 4)            0           ['batch_normalization_53[0][0]']

 dense_3 (Dense)                (None, 1)            5           ['activation_53[0][0]']

==================================================================================================
Total params: 907,099
Trainable params: 890,445
Non-trainable params: 16,654
__________________________________________________________________________________________________
```

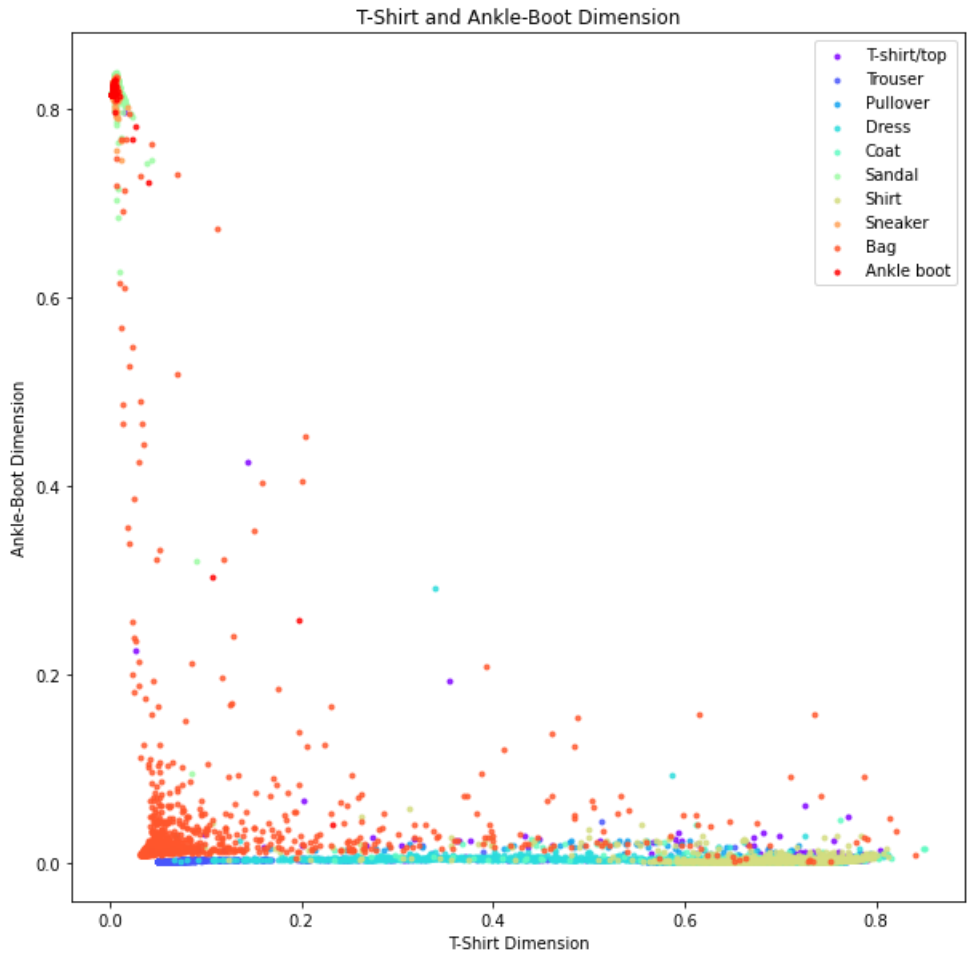
Спочатку, перевіримо, як виглядає результат роботи нетренованої мережі.



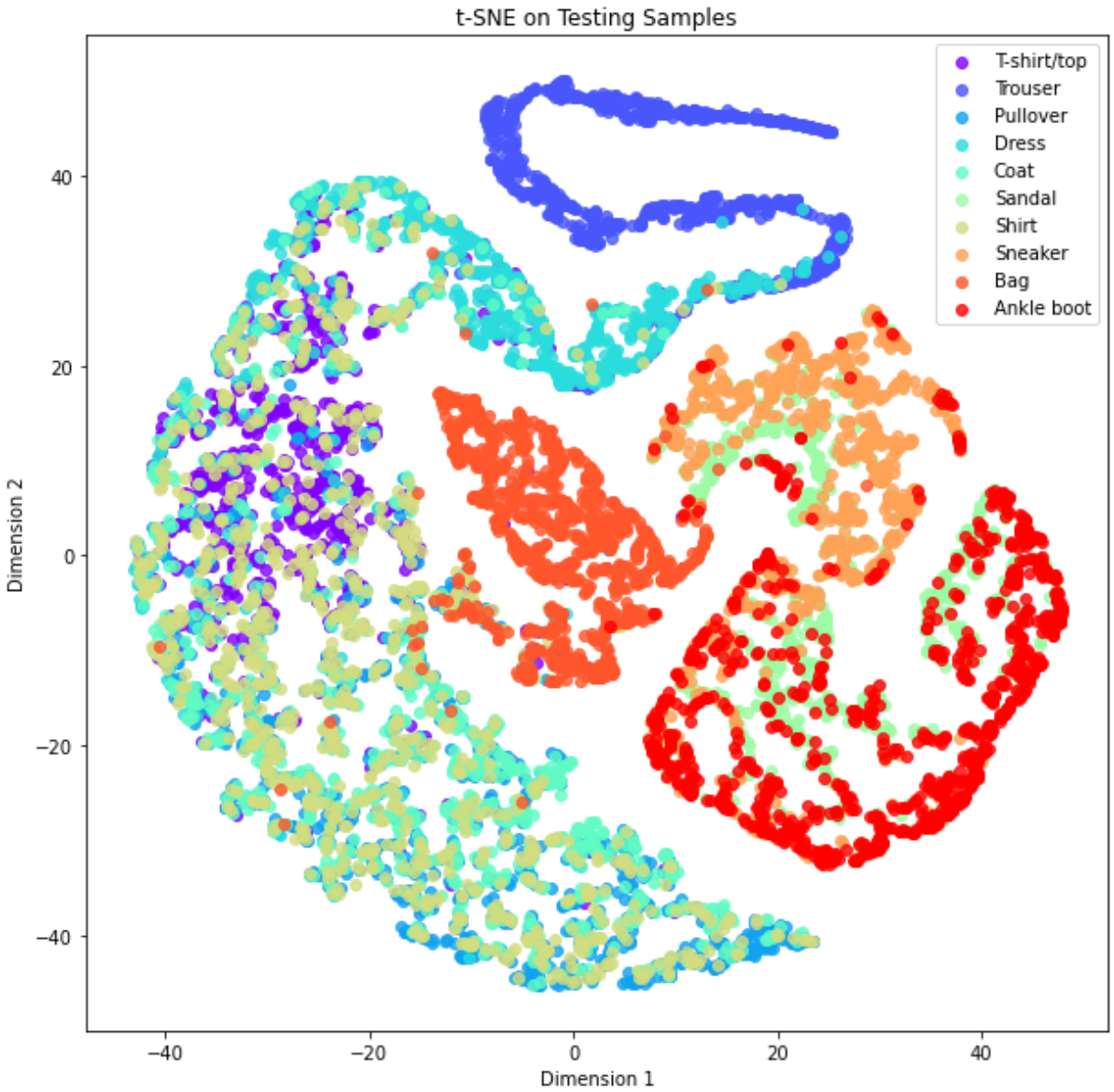
Як бачимо, нетренована модель завжди передбачає 50% ймовірність схожості двох зображень. Далі, проводимо тренування моделі (10 епох).



Після 10 епох результат передбачення набагато кращий, модель загалом дає високу вірогідність схожості для зображень однакового класу, і низьку для зображень різних класів. Далі, поглянемо, як виглядає діаграма схожості різних зображень до класів Ankle boot чи T-shirt.



Як бачимо, більшість зображень типу Ankle boot мають високу вірогідність бути схожими на інші екземпляри цього ж класу, і відносно низьку вірогідність бути схожими на зображення класу T-shirt, і навпаки. Використаємо також TSNE щоб візуалізувати, наскільки добре кластеризуються зображення різних класів за допомогою моделі.



Бачимо, що об’єкти класів Bag і Trouser кластеризувались найбільш відмінно від інших. Класи Ankle boot, Sandal і Sneaker є досить близькими в кластеризації, що можна пояснити тим, що вони всі є видами взуття. Також, між класами T-shirt, Pullover, Dress і Coat є чимало перетину.

Висновки

Під час виконання цієї лабораторної роботи я навчився вирішувати задачу побудови CNN на основі DenseNet для класифікації зображень і будувати систему пошуку подібних зображень на основі Siamese Networks в датасеті fasion-mnist.