D.Sc., Prof., Dmytro Peleshko, D.Sc., Prof., Olena Vynokurova

Лабораторна робота № 2.

Класифікація зображень. Застосування нейромереж для пошуку подібних зображень.

Мета - набути практичних навиків у розв’язанні задачі пошуку подібних зображень на прикладі організації CNN класифікації.

Вступ

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.

```
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
```

```
Using TensorFlow backend.
/opt/conda/lib/python3.6/importlib/_bootstrap.py:219: RuntimeWarning: compiletime version 3.
5 of module 'tensorflow.python.framework.fast_tensor_util' does not match runtime version 3.
6
  return f(*args, **kwds)
```

Завантаження та препроцесінг даних

Завантажуємо і упорядковуємо дані так, щоб їх можна було легко використовувати всередині моделей Keras

```
from sklearn.model_selection import train_test_split
data_train = pd.read_csv('../input/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())
```

```
Training (42000, 28, 28, 1) 1.0
Testing (18000, 28, 28, 1) 1.0
```

```
# 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])
```

```
train groups: [4165, 4155, 4162, 4196, 4258, 4246, 4239, 4184, 4230, 4165]
test groups: [1835, 1845, 1838, 1804, 1742, 1754, 1761, 1816, 1770, 1835]
```

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.

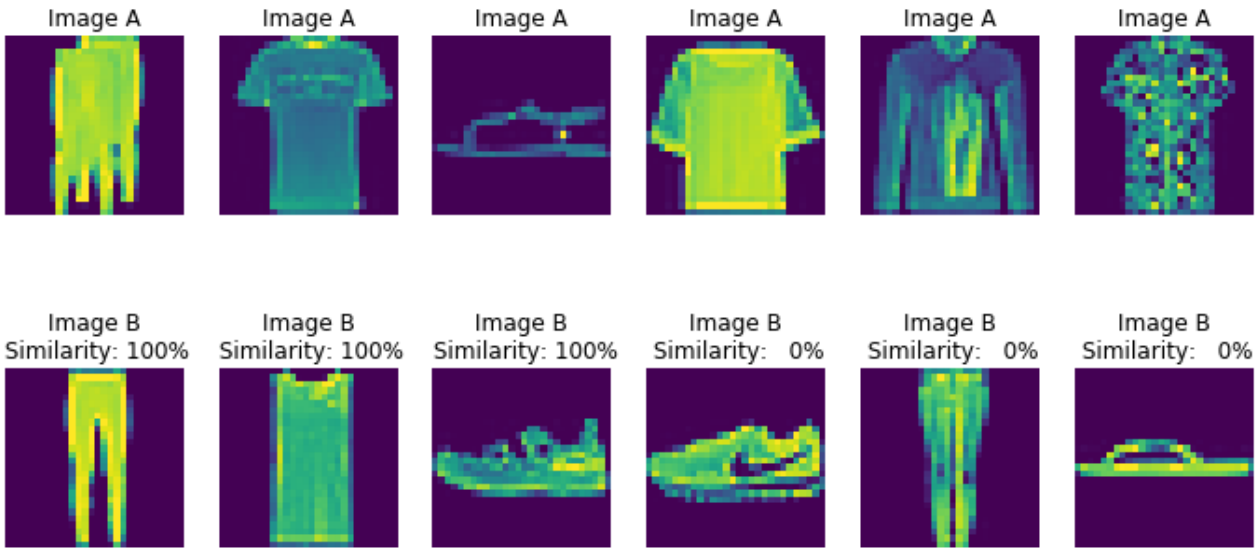
```
def gen_random_batch(in_groups, batch_halfsize = 8):
    out_img_a, out_img_b, out_score = [], [], []
    all_groups = list(range(len(in_groups)))
    for match_group in [True, False]:
        group_idx = np.random.choice(all_groups, size = batch_halfsize)
        out_img_a += [in_groups[c_idx]
[np.random.choice(range(in_groups[c_idx].shape[0]))] 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)
```

Валідація даних

Не етапі валідації переконуємось, що генератор робить щось “розумне і правильне”. Для цього виводяться зображення і відсоток їх подібності.

```
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')
```



Генерація фічей (ознак)

Для генерації фічей з зображень будуємо конволюційну мережу. Мережа ініціалізується випадковим чином і буде навчатись генерувати корисні вектори фічей з вхідних зображень.

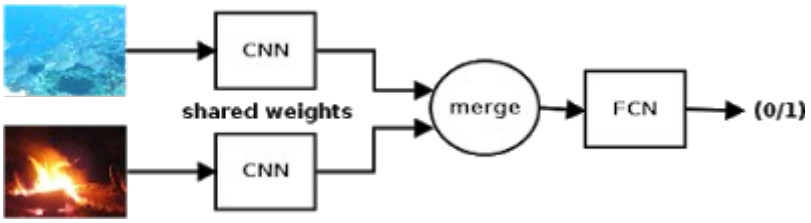
```
from keras.models import Model
from keras.layers import Input, Conv2D, BatchNormalization, MaxPool2D,
Activation, Flatten, Dense, Dropout
img_in = Input(shape = x_train.shape[1:], name = 'FeatureNet_ImageInput')
n_layer = img_in
for i in range(2):
    n_layer = Conv2D(8*2**i, kernel_size = (3,3), activation = 'linear')
(n_layer)
    n_layer = BatchNormalization()(n_layer)
    n_layer = Activation('relu')(n_layer)
    n_layer = Conv2D(16*2**i, kernel_size = (3,3), activation = 'linear')
(n_layer)
    n_layer = BatchNormalization()(n_layer)
    n_layer = Activation('relu')(n_layer)
    n_layer = MaxPool2D((2,2))(n_layer)
n_layer = Flatten()(n_layer)
n_layer = Dense(32, activation = 'linear')(n_layer)
n_layer = Dropout(0.5)(n_layer)
n_layer = BatchNormalization()(n_layer)
n_layer = Activation('relu')(n_layer)
feature_model = Model(inputs = [img_in], outputs = [n_layer], name =
'FeatureGenerationModel')
feature_model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
FeatureNet_ImageInput (Input (None, 28, 28, 1)         0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 26, 26, 8)         80
_________________________________________________________________
batch_normalization_1 (Batch (None, 26, 26, 8)         32
_________________________________________________________________
activation_1 (Activation)    (None, 26, 26, 8)         0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 24, 24, 16)        1168
_________________________________________________________________
batch_normalization_2 (Batch (None, 24, 24, 16)        64
_________________________________________________________________
activation_2 (Activation)    (None, 24, 24, 16)        0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 12, 12, 16)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 10, 10, 16)        2320
_________________________________________________________________
batch_normalization_3 (Batch (None, 10, 10, 16)        64
_________________________________________________________________
activation_3 (Activation)    (None, 10, 10, 16)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 8, 8, 32)          4640
_________________________________________________________________
batch_normalization_4 (Batch (None, 8, 8, 32)          128
_________________________________________________________________
activation_4 (Activation)    (None, 8, 8, 32)          0
```

```
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 4, 4, 32)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 32)                16416
_________________________________________________________________
dropout_1 (Dropout)          (None, 32)                0
_________________________________________________________________
batch_normalization_5 (Batch (None, 32)                128
_________________________________________________________________
activation_5 (Activation)    (None, 32)                0
=================================================================
Total params: 25,040
Trainable params: 24,832
Non-trainable params: 208
_________________________________________________________________
```

Siamese модель

Застосовуємо модель, яка генерує фічі, до обох зображень, а потім об'єднуємо їх разом, щоб спрогнозувати їх подібність. Ідея моделі є такою:



Кінцева ідея полягає в тому, що при отриманні нового зображення можна розрахувати для нього функціональний вектор за допомогою FeatureGenerationModel. Всі існуючі зображення були попередньо розраховані і збережені в базі даних векторів-фіч. Модель може бути застосована з використанням декількох векторних додавань і множень для визначення найбільш схожих растрів. Ці операції можуть бути реалізовані у вигляді збереженої процедури або всередині самої бази даних.

```
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()
```

```
------
Layer (type)                 Output Shape              Param #   Connected to
==================================================================================================
======
ImageA_Input (InputLayer)    (None, 28, 28, 1)         0
--------------------------------------------------------------------------------------------------
------
ImageB_Input (InputLayer)    (None, 28, 28, 1)         0
--------------------------------------------------------------------------------------------------
------
FeatureGenerationModel (Model) (None, 32)              25040     ImageA_Input[0][0]
                                                                 ImageB_Input[0][0]
--------------------------------------------------------------------------------------------------
------
merge_features (Concatenate)  (None, 64)               0         FeatureGenerationModel[1]
[0]
                                                                 FeatureGenerationModel[2]
[0]
--------------------------------------------------------------------------------------------------
------
dense_2 (Dense)              (None, 16)                1040      merge_features[0][0]
--------------------------------------------------------------------------------------------------
------
batch_normalization_6 (BatchNor (None, 16)             64        dense_2[0][0]
--------------------------------------------------------------------------------------------------
------
activation_6 (Activation)    (None, 16)                0         batch_normalization_6[0][0]
--------------------------------------------------------------------------------------------------
------
dense_3 (Dense)              (None, 4)                 68        activation_6[0][0]
--------------------------------------------------------------------------------------------------
------
batch_normalization_7 (BatchNor (None, 4)              16        dense_3[0][0]
--------------------------------------------------------------------------------------------------
------
dense_4 (Dense)              (None, 1)                 5         activation_7[0][0]
==================================================================================================
======
Total params: 26,233
Trainable params: 25,985
Non-trainable params: 248
```

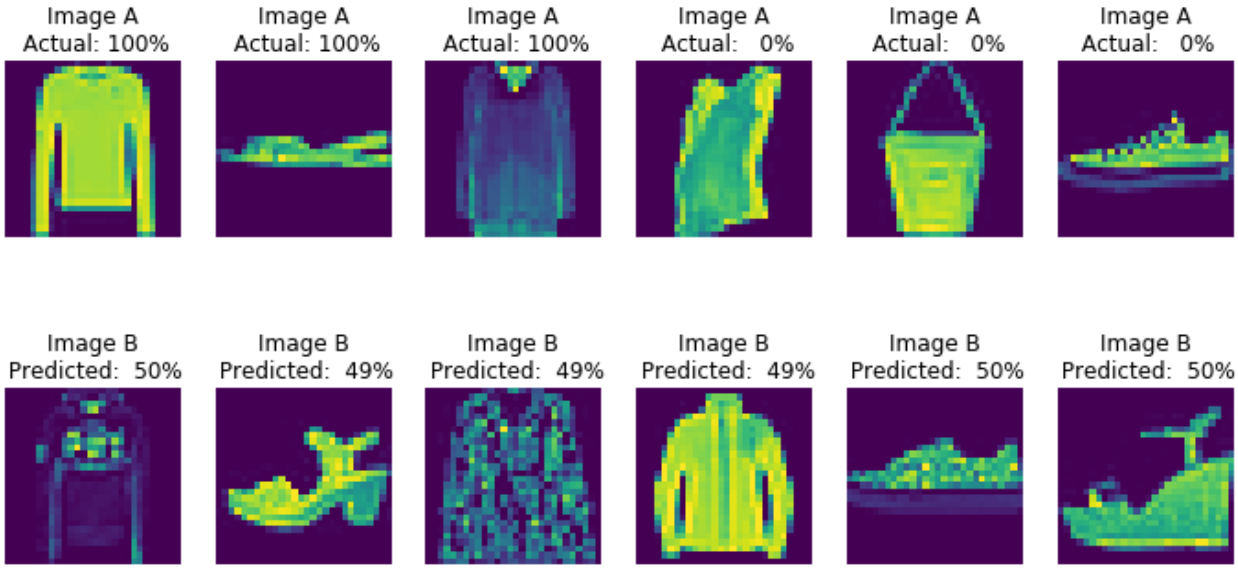
```
# setup the optimization process
similarity_model.compile(optimizer='adam', loss = 'binary_crossentropy',
metrics = ['mae'])
```

Візуалізація результатів

Взявши невелику вибірку випадково вибраних зображень A і B першої половини з однієї і тієї ж категорії, а другий - з різних категорій, візуалізуємо роботу моделі.

Також виведемо фактичну відстань (0 для тієї ж категорії і 1 для різних категорій), а також прогнозовану відстань моделі. Перший прогін буде з повністю непідготовленою мережею, тому значні по якості результати не очікуються.

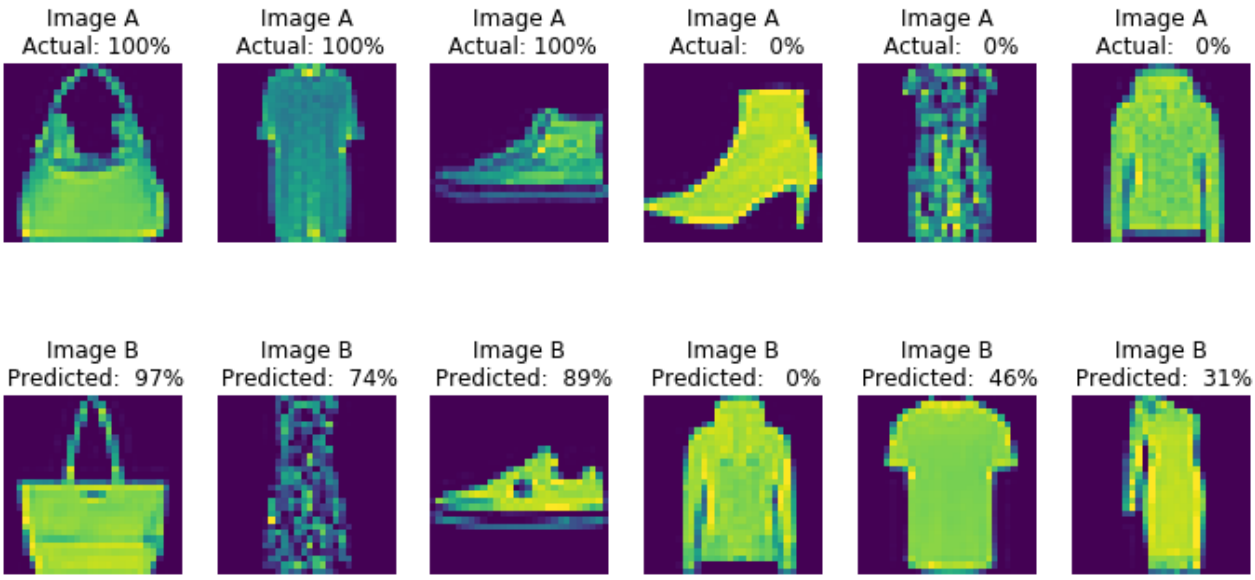
```
def show_model_output(nb_examples = 3):
    pv_a, pv_b, pv_sim = gen_random_batch(test_groups, nb_examples)
    pred_sim = similarity_model.predict([pv_a, pv_b])
    fig, m_axs = plt.subplots(2, pv_a.shape[0], figsize = (12, 6))
    for c_a, c_b, c_d, p_d, (ax1, ax2) in zip(pv_a, pv_b, pv_sim, pred_sim,
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
# we want a constant validation group to have a frame of reference for model
performance
valid_a, valid_b, valid_sim = gen_random_batch(test_groups, 1024)
loss_history = similarity_model.fit_generator(siam_gen(train_groups),
                                steps_per_epoch = 500,
                                validation_data=([valid_a, valid_b],
valid_sim),
                                               epochs = 10,
                                              verbose = True)
```

```
Epoch 1/10
500/500 [==============================] - 61s 122ms/step - loss: 0.6475 - mean_absolute_err
or: 0.4596 - val_loss: 0.5082 - val_mean_absolute_error: 0.3765
Epoch 2/10
500/500 [==============================] - 62s 125ms/step - loss: 0.5057 - mean_absolute_err
or: 0.3619 - val_loss: 0.4097 - val_mean_absolute_error: 0.2911
Epoch 3/10
500/500 [==============================] - 62s 124ms/step - loss: 0.4534 - mean_absolute_err
or: 0.3099 - val_loss: 0.3535 - val_mean_absolute_error: 0.2392
Epoch 4/10
500/500 [==============================] - 63s 126ms/step - loss: 0.4163 - mean_absolute_err
or: 0.2806 - val_loss: 0.3348 - val_mean_absolute_error: 0.2129
Epoch 5/10
500/500 [==============================] - 62s 124ms/step - loss: 0.4000 - mean_absolute_err
or: 0.2643 - val_loss: 0.3252 - val_mean_absolute_error: 0.2093
Epoch 6/10
500/500 [==============================] - 62s 124ms/step - loss: 0.3865 - mean_absolute_err
or: 0.2524 - val_loss: 0.3139 - val_mean_absolute_error: 0.2002
Epoch 7/10
500/500 [==============================] - 57s 114ms/step - loss: 0.3862 - mean_absolute_err
or: 0.2520 - val_loss: 0.3087 - val_mean_absolute_error: 0.2068
Epoch 8/10
500/500 [==============================] - 54s 108ms/step - loss: 0.3654 - mean_absolute_err
or: 0.2395 - val_loss: 0.3098 - val_mean_absolute_error: 0.1921
Epoch 9/10
500/500 [==============================] - 53s 106ms/step - loss: 0.3677 - mean_absolute_err
or: 0.2368 - val_loss: 0.3099 - val_mean_absolute_error: 0.1943
Epoch 10/10
500/500 [==============================] - 54s 108ms/step - loss: 0.3660 - mean_absolute_err
or: 0.2347 - val_loss: 0.3044 - val_mean_absolute_error: 0.1942
```

```
_ = show_model_output()
```



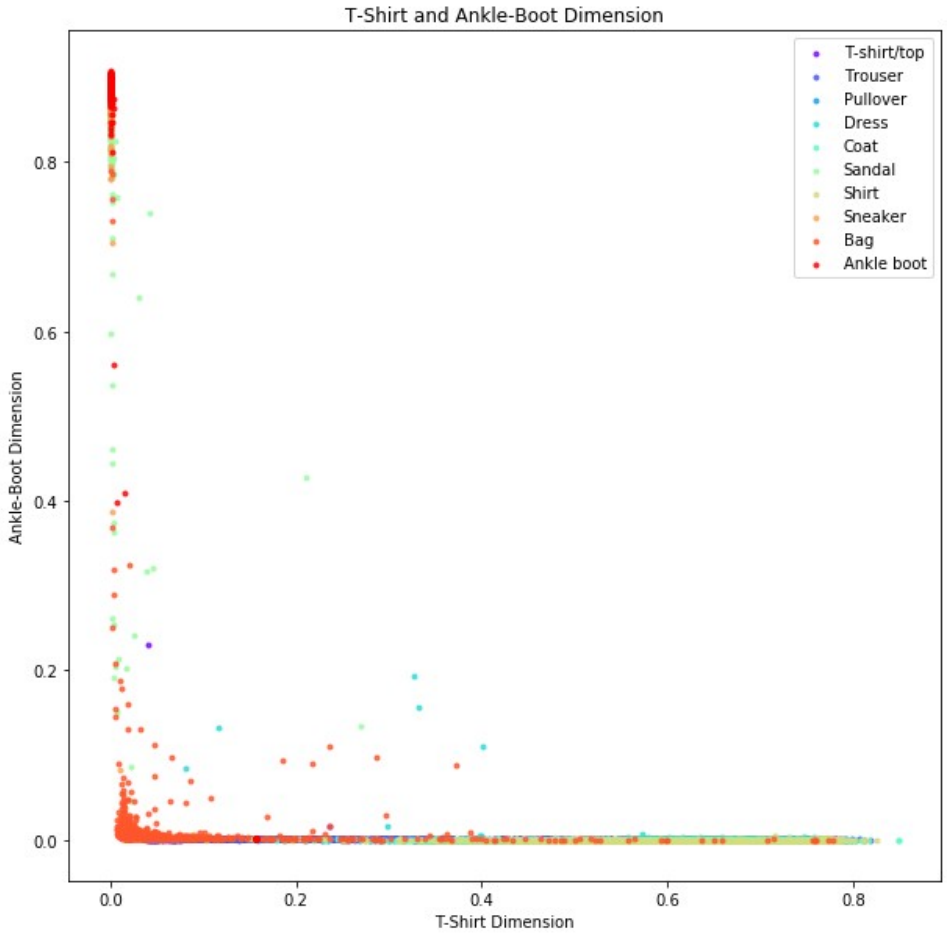
T-Shirt vs Ankle Boot-Plot

Беремо випадкові образи футболки і черевика (категорії 0 і 9) і обчислюємо відстань по мережі до інших образів

```
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)
```

```
18000/18000 [==============================] - 21s 1ms/step
18000/18000 [==============================] - 20s 1ms/step
```

```
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)
```



Перевірка фічей

Одне із найбільш важливих питань - чи за допомогою моделі Feature Generation генерувались корисні фічі. І як можна це візуалізувати

```
x_test_features = feature_model.predict(x_test, verbose = True,
batch_size=128)
```

```
18000/18000 [==============================] - 11s 612us/step
```

Для цього використовується TSNE, який дає можливість візуалізувати на 2D площині групи кластерів. Для прикладу цього використовуємо тестові дані.

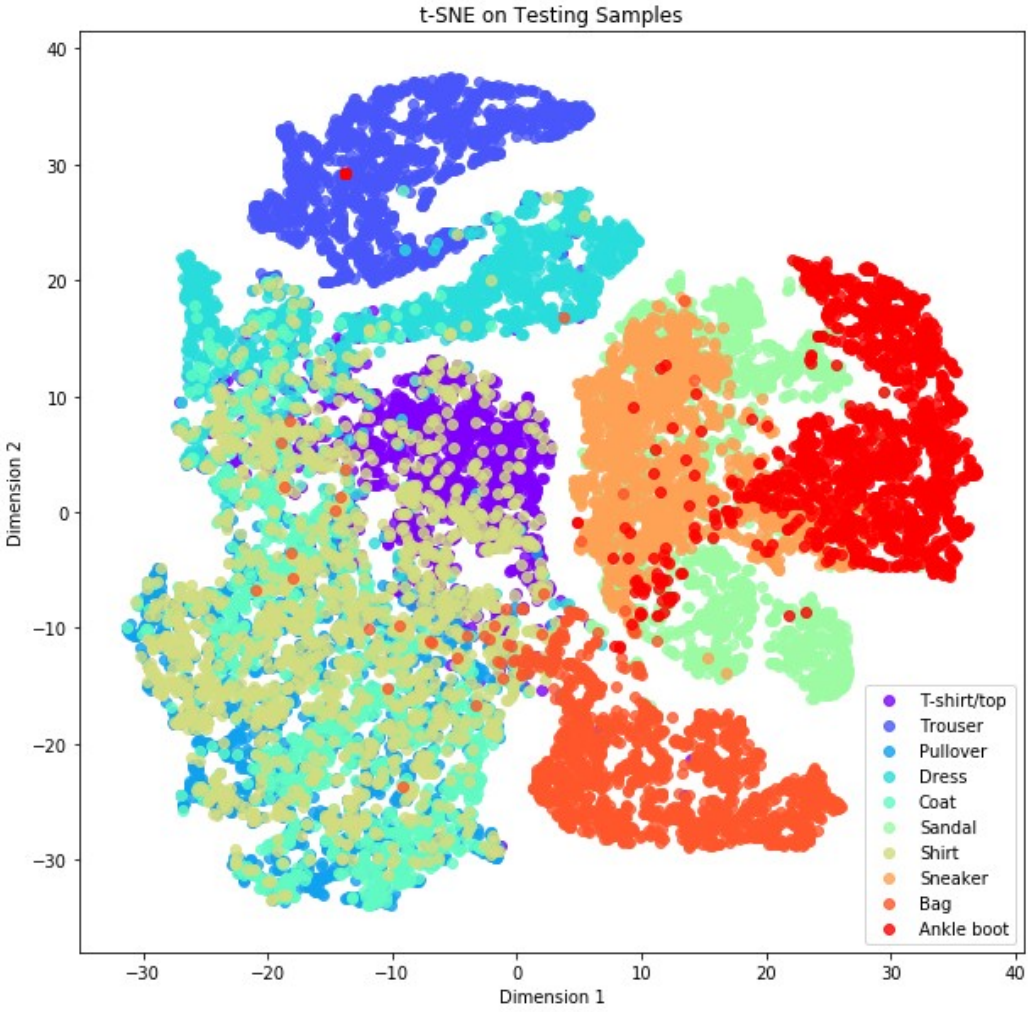
```
%%time
from sklearn.manifold import TSNE
tsne_obj = TSNE(n_components=2,
                         init='pca',
                         random_state=101,
                         method='barnes_hut',
                         n_iter=500,
                         verbose=2)
tsne_features = tsne_obj.fit_transform(x_test_features)
```

```
[t-SNE] Computing 91 nearest neighbors...
[t-SNE] Indexed 18000 samples in 0.226s...
[t-SNE] Computed neighbors for 18000 samples in 4.028s...
[t-SNE] Computed conditional probabilities for sample 1000 / 18000
[t-SNE] Computed conditional probabilities for sample 2000 / 18000
[t-SNE] Computed conditional probabilities for sample 3000 / 18000
[t-SNE] Computed conditional probabilities for sample 4000 / 18000
[t-SNE] Computed conditional probabilities for sample 5000 / 18000
[t-SNE] Computed conditional probabilities for sample 6000 / 18000
[t-SNE] Computed conditional probabilities for sample 7000 / 18000
[t-SNE] Computed conditional probabilities for sample 8000 / 18000
[t-SNE] Computed conditional probabilities for sample 9000 / 18000
[t-SNE] Computed conditional probabilities for sample 10000 / 18000
[t-SNE] Computed conditional probabilities for sample 11000 / 18000
[t-SNE] Computed conditional probabilities for sample 12000 / 18000
[t-SNE] Computed conditional probabilities for sample 13000 / 18000
[t-SNE] Computed conditional probabilities for sample 14000 / 18000
[t-SNE] Computed conditional probabilities for sample 15000 / 18000
[t-SNE] Computed conditional probabilities for sample 16000 / 18000
[t-SNE] Computed conditional probabilities for sample 17000 / 18000
[t-SNE] Computed conditional probabilities for sample 18000 / 18000
[t-SNE] Mean sigma: 0.097702
[t-SNE] Computed conditional probabilities in 1.213s
[t-SNE] Iteration 50: error = 82.1846161, gradient norm = 0.0019173 (50 iterations in 27.468
s)
[t-SNE] Iteration 100: error = 80.4134293, gradient norm = 0.0010669 (50 iterations in 26.79
2s)
[t-SNE] Iteration 150: error = 79.5910645, gradient norm = 0.0007335 (50 iterations in 27.38
2s)
[t-SNE] Iteration 200: error = 79.0950394, gradient norm = 0.0005696 (50 iterations in 27.34
4s)
[t-SNE] Iteration 250: error = 78.7620468, gradient norm = 0.0004646 (50 iterations in 27.55
6s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 78.762047
```

```
[t-SNE] Computing 91 nearest neighbors...
[t-SNE] Indexed 18000 samples in 0.226s...
[t-SNE] Computed neighbors for 18000 samples in 4.028s...
[t-SNE] Computed conditional probabilities for sample 1000 / 18000
[t-SNE] Computed conditional probabilities for sample 2000 / 18000
[t-SNE] Computed conditional probabilities for sample 3000 / 18000
[t-SNE] Computed conditional probabilities for sample 4000 / 18000
[t-SNE] Computed conditional probabilities for sample 5000 / 18000
[t-SNE] Computed conditional probabilities for sample 6000 / 18000
[t-SNE] Computed conditional probabilities for sample 7000 / 18000
[t-SNE] Computed conditional probabilities for sample 8000 / 18000
[t-SNE] Computed conditional probabilities for sample 9000 / 18000
[t-SNE] Computed conditional probabilities for sample 10000 / 18000
[t-SNE] Computed conditional probabilities for sample 11000 / 18000
[t-SNE] Computed conditional probabilities for sample 12000 / 18000
[t-SNE] Computed conditional probabilities for sample 13000 / 18000
[t-SNE] Computed conditional probabilities for sample 14000 / 18000
[t-SNE] Computed conditional probabilities for sample 15000 / 18000
[t-SNE] Computed conditional probabilities for sample 16000 / 18000
[t-SNE] Computed conditional probabilities for sample 17000 / 18000
[t-SNE] Computed conditional probabilities for sample 18000 / 18000
[t-SNE] Mean sigma: 0.097702
[t-SNE] Computed conditional probabilities in 1.213s
[t-SNE] Iteration 50: error = 82.1846161, gradient norm = 0.0019173 (50 iterations in 27.468
s)
[t-SNE] Iteration 100: error = 80.4134293, gradient norm = 0.0010669 (50 iterations in 26.79
2s)
[t-SNE] Iteration 150: error = 79.5910645, gradient norm = 0.0007335 (50 iterations in 27.38
2s)
[t-SNE] Iteration 200: error = 79.0950394, gradient norm = 0.0005696 (50 iterations in 27.34
4s)
[t-SNE] Iteration 250: error = 78.7620468, gradient norm = 0.0004646 (50 iterations in 27.55
6s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 78.762047
[t-SNE] Iteration 300: error = 3.2698922, gradient norm = 0.0012206 (50 iterations in 27.653
s)
[t-SNE] Iteration 350: error = 2.7692475, gradient norm = 0.0006349 (50 iterations in 27.760
s)
[t-SNE] Iteration 400: error = 2.4634285, gradient norm = 0.0004027 (50 iterations in 27.423
s)
[t-SNE] Iteration 450: error = 2.2674994, gradient norm = 0.0002826 (50 iterations in 27.257
s)
[t-SNE] Iteration 500: error = 2.1313024, gradient norm = 0.0002141 (50 iterations in 26.836
s)
[t-SNE] Error after 500 iterations: 2.131302
CPU times: user 9min 7s, sys: 1min 10s, total: 10min 18s
Wall time: 4min 39s
```

```
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)
```



Завдання

1. Побудувати CNN на основі **LeNet-5** для класифікації зображень на основі датасету fashion-mnist.
   Зробити налаштування моделі для досягнення необхідної точностіНа базі Siamese networks побудувати систему для пошуку подібних зображень в датасеті fashion-mnist. Візуалізувати отримані результати t-SNE.
2. Побудувати CNN на основі **AlexNet** для класифікації зображень на основі

датасету fashion-mnist.
Зробити налаштування моделі для досягнення необхідної точностіНа базі Siamese networks побудувати систему для пошуку подібних зображень в датасеті fashion-mnist. Візуалізувати отримані результати t-SNE.

3. Побудувати CNN на основі **VGG-16** 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.
4. Побудувати CNN на основі Inception-v1 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.
5. Побудувати CNN на основі Inception-v3 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.
6. 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.
7. Побудувати CNN на основі Xception 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.
8. Побудувати CNN на основі Inception-v4 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.
9. Побудувати CNN на основі Inception-ResNets 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.
10. 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.
11. 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.

Інформацію про типи CNN можна знайти за лінкою
https://towardsdatascience.com/illustrated-10-cnn-architectures-95d78ace614d