МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ “ЛЬВІВСЬКА ПОЛІТЕХНІКА”
ІНСТИТУТ КОМП’ЮТЕРНИХ НАУК ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
кафедра Систем Штучного Інтелекту

Лабораторна робота №2

з курсу “Обробка зображень методами штучного інтелекту”

Виконала:
студентка групи КН-409
Дипко Олександра

Викладач:
Пелешко Д. Д.

Львів-2022

Тема: Суміщення зображень на основі використання дескрипторів.

Мета: Навчитись вирішувати задачу суміщення зображень засобом видобування особливих точок і використання їх в процедурах матчінгу.

Хід роботи

Варіант 8

Вибрати з інтернету набори зображень з різною контрастністю і різним флуктуаціями освітленості. Для кожного зображення побудувати варіант спотвореного (видозміненого зображення). Для кожної отриманої пари побудувати дескриптор і проаналізувати можливість суміщення цих зображень і з визначення параметрів геметричних перетворень (кут повороту, зміщень в напрямку x і напрямку y).

Результати

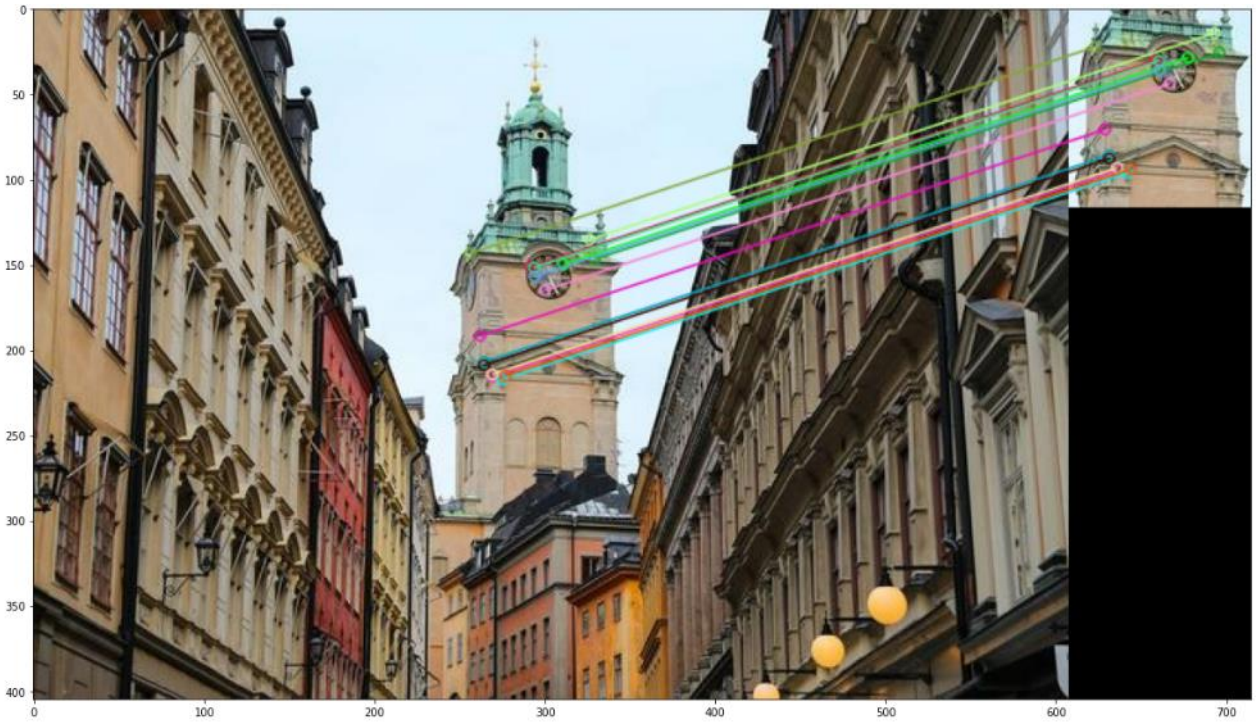

Рис 1. Результат матчінгу зображень методом Brute Force

Рис 2. Результат метчінгу зображень власним методом

Рис 3. Результат метчінгу зображень методом Brute Force

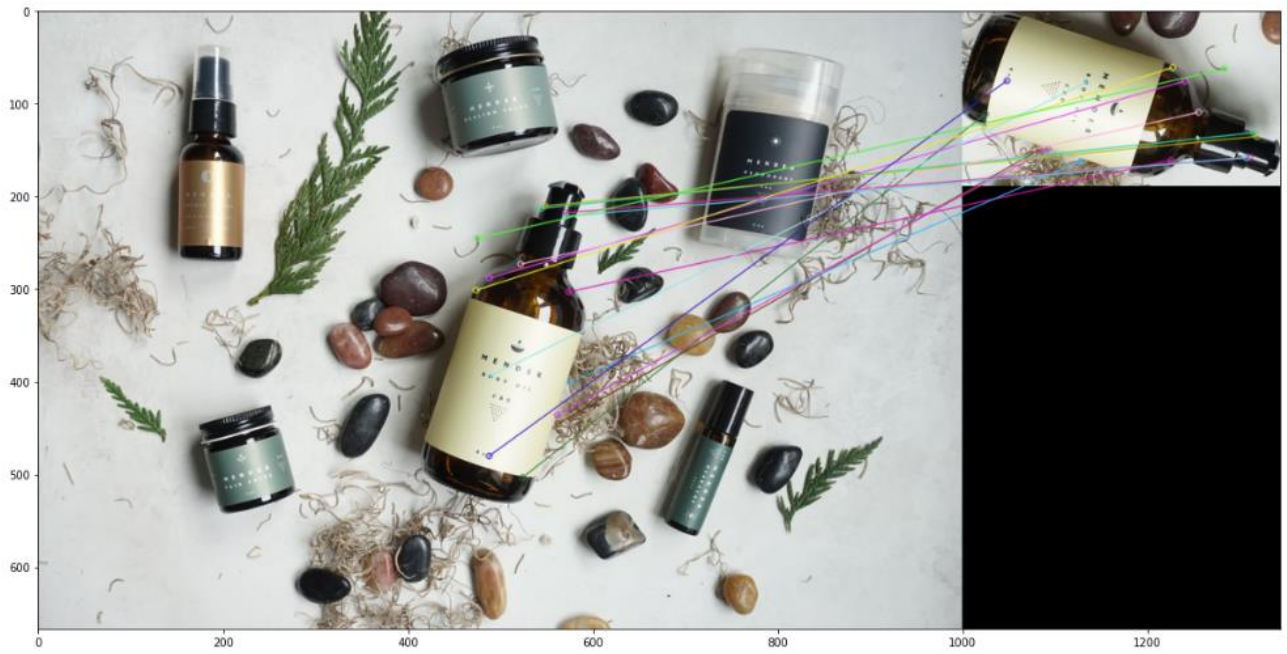

Рис 4. Результат метчінгу зображень власним методом

Висновок

Brute-Force matcher (BF-matcher) є реалізовує простий матчінг метод. Він приймає дескриптор однієї ознаки в першій множині і зіставляє за деякою метрикою з усіма ознаками в другій множині. Як результат повертається найближчий дексриптор (ознака).

Для використання BF-matcher спочатку використовуючи функцію cv.BFMatcher () необхідно створити об'єкт BFMatcher. Функція має два необов'язкові параметри. Перший - normType. Він задає тип метрики для вимірювання відстані між дескрипторами. За замовчуванням використовується метрика L2 (cv.NORM_L2). Для SIFT та SURF і т.д. також рекомендується використовувати метрику L1 (cv.NORM_L1).

Для дескрипторів, заснованих на бінарних рядках, таких як ORB, BRIEF, BRISK і т.д., треба використовувати метрику Хемінга (cv.NORM_HAMMING). Якщо ORB використовує WTA_K == 3 або 4, то слід використовувати cv.NORM_HAMMING2.

Додаток

```
import cv2 as cv
import matplotlib.pyplot as plt
from scipy.spatial.distance import hamming
```

```
def draw_matches(matches, keypoints1, descriptors1, keypoints2, descriptors2,
image1, image2):
    output = cv.drawMatches(img1=image1,keypoints1=keypoints1,
img2=image2,keypoints2=keypoints2, matches1to2=matches[:15], outImg=None,
                            flags=cv.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS)
    plt.figure(figsize=(20,20))
    plt.imshow(output)
    plt.show()


def brisk_descriptor(img1, img2):
    image1 = cv.cvtColor(img1, cv.COLOR_BGR2GRAY)
    image2 = cv.cvtColor(img2, cv.COLOR_BGR2GRAY)

    BRISK = cv.BRISK_create()
    keypoints1, descriptors1 = BRISK.detectAndCompute(image1, None)
    keypoints2, descriptors2 = BRISK.detectAndCompute(image2, None)

    print('Number of keypoints Detected on distorted image:', len(keypoints2),
"\n")
    return keypoints1, descriptors1, keypoints2, descriptors2


def brute_force_matcher(keypoints1, descriptors1, keypoints2, descriptors2):
    BFMatcher = cv.BFMatcher(normType=cv.NORM_HAMMING, crossCheck=True)
    matches = BFMatcher.match(queryDescriptors=descriptors1,
trainDescriptors=descriptors2)

    matches = sorted(matches, key=lambda x: x.distance)
    return matches


def own_matcher(keypoints1, descriptors1, keypoints2, descriptors2):
    matches = []
    for i, k1 in enumerate(descriptors1):
        for j, k2 in enumerate(descriptors2):
            matches.append(cv.DMatch(_distance=hamming(k1, k2) * len(k1),
_imgIdx=0, _queryIdx=i, _trainIdx=j))

    matches = sorted(matches, key=lambda x: x.distance)
    return matches

original_image = cv.cvtColor(cv.imread('original_image1.png'), cv.COLOR_BGR2RGB)
distorted_image = cv.cvtColor(cv.imread('distorted_image1.png'),
cv.COLOR_BGR2RGB)
keypoints1, descriptors1, keypoints2, descriptors2 =
brisk_descriptor(original_image, distorted_image)

matches1 = brute_force_matcher(keypoints1, descriptors1, keypoints2,
descriptors2)
draw_matches(matches1, keypoints1, descriptors1, keypoints2, descriptors2,
original_image, distorted_image)
matches2 = own_matcher(keypoints1, descriptors1, keypoints2, descriptors2)
draw_matches(matches2, keypoints1, descriptors1, keypoints2, descriptors2,
original_image, distorted_image)
original_image = cv.cvtColor(cv.imread('original_image.png'), cv.COLOR_BGR2RGB)
distorted_image = cv.cvtColor(cv.imread('distorted_image.png'),
cv.COLOR_BGR2RGB)
```