МІНІСТЕРСТВО ОСВІТИ І НАУКИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»

Кафедра систем штучного інтелекту

Лабораторна робота №2

З курсу “Обробка зображень методами штучного інтелекту”

Виконала:
студентка групи КН-411
Досяк Ірина

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.

Теоретичні відомості

Метод SIFT:

Алгоритм SIFT складається з п’яти основних етапів:

1. Виявлення масштабно-просторових екстремумів (Scale-space Extrema Detection) - основним завданням етапу є виділення локальних екстремальних точок засобом побудови пірамід гаусіанів (Gaussian) і різниць гаусіанів (Difference of Gaussian, DoG).

2. Локалізація ключових точок (Keypoint Localization) - основним завданням етапу є подальше уточнення локальних екстремумів з метою фільтрації їх набору - тобто видалення з подальшого аналізу точок, які є краєвими, або мають низьку контрастність.

3. Визначення орієнтації (Orientation Assignment) - для досягнення інваріантності повороту растра на цьому етапі кожній ключовій точці присвоюється орієнтація.

4. Дескриптор ключових точок (Keypoint Descriptor) - завданням етапу є побудова дескрипторів, які містяь інформацію про окіл особливої точки для задачі подальшого порівння на збіг.

5. Зіставлення по ключових точках (Keypoint Matching) - пошук збігів для вирішення завдання суміщення зображень.

Метод SURF:

Метод SURF (Speeded Up Robust Features) — це швидкий і надійний алгоритм локального, інваріантного подібності представлення та порівняння зображень.

Виконання

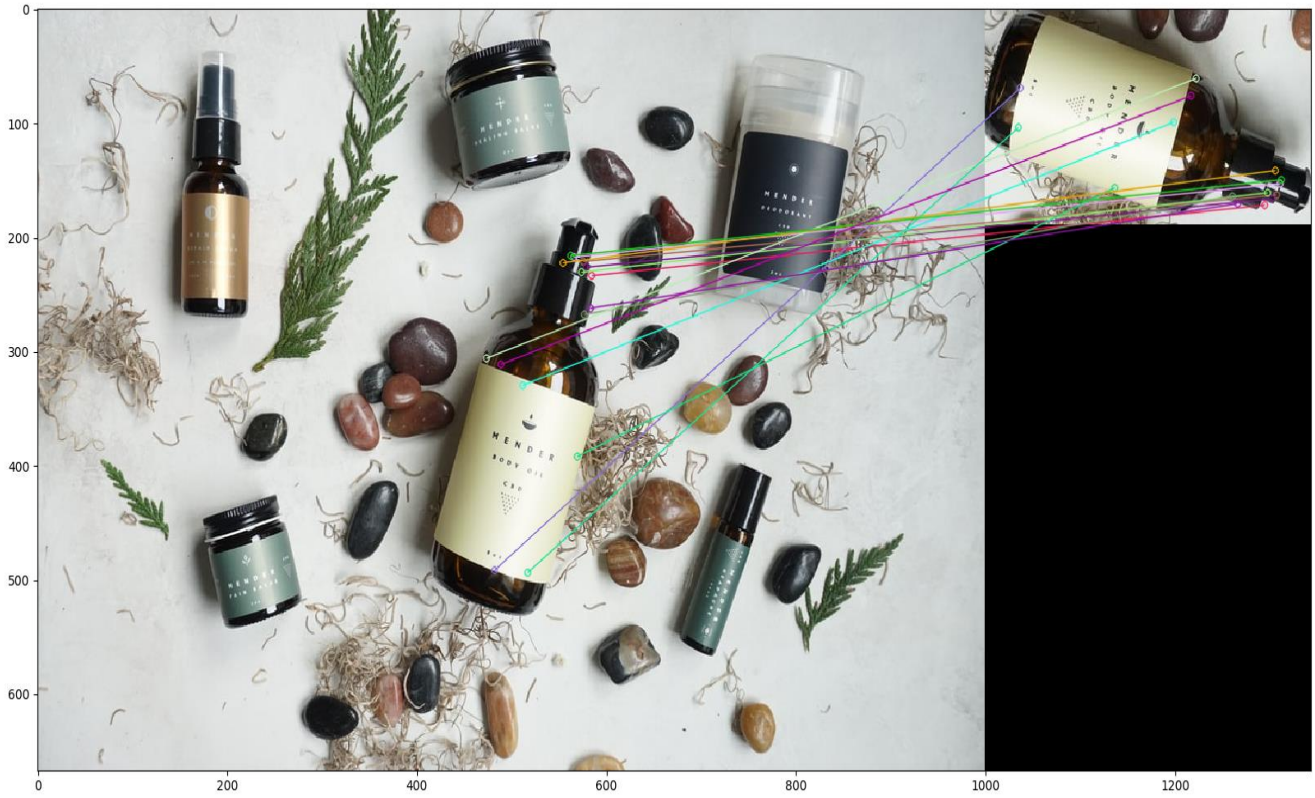
Варіант №6

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).

SURF

Для перевірки збігів необхідно написати власну функцію матчінгу, а результати її роботи перевірити засобами OpenCV. Якщо повної реалізації дескриптора не має в OpenCV, то такий необхідно створити власну функцію побудови цих дискрипторів. У цьому випадку матчінг можна здійснювати стандартними засобами (якщо це можливо).

Результати




0
100
200
300
400
500
600
0
200
400
600
800
1000
1200


0
50
100
150
200
250
300
350
400
0
100
200
300
400
500
600
700

Коментарі: Спершу я зчитую зображення. Для визначення особливих точок я використала Surf Detection, вбудований в opencv. Після цього провела процедуру матчингу двома способами – власним матчером та Brute Force матчером. У кінці я показала результати матчингів на зображеннях, порівнюючи їх.

Висновок

Порівнюючи результати роботи вбудованого матчера та власного, можна сказати, що працюють вони доволі схоже. Зображення, які ми отримали в результаті, дуже схожі між собою, проте є деякі відмінності. Власний матчер приймає параметр ratio, який задає «прискіпливість» алгоритму. В процесі матчингу було застосовано матричну норму, а також було реалізовано кросматчінг, який спершу обраховує матчі в двох напрямках – forward та backward, а потім вибирає лише ті, які співпадають при обрахунках.

Додаток

```
import cv2 as cv
import numpy as np
```

```
from matplotlib import pyplot as plt


def draw_matches(matches, key1, key2, image1, image2):
    output = cv.drawMatches(img1=image1,
                            keypoints1=key1,
                            img2=image2,
                            keypoints2=key2,
                            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):
    # Convert the training image to gray scale
    image1 = cv.cvtColor(img1, cv.COLOR_BGR2GRAY)
    image2 = cv.cvtColor(img2, cv.COLOR_BGR2GRAY)
    # Create a Surf Matcher object
    brisk = cv.xfeatures2d.SURF_create()
    key1, desc1 = brisk.detectAndCompute(image1, None)
    key2, desc2 = brisk.detectAndCompute(image2, None)
    print('Number of key_points detected on distorted image:', len(key2), "\n")
    return key1, desc1, key2, desc2


def brute_force_matcher(desc1, desc2):
    # Create a Brute Force Matcher object
    BFMatcher = cv.BFMatcher(normType=cv.NORM_L1, crossCheck=True)
    # Perform the matching between the Surf descriptors of the training image
and the test image
    matches = BFMatcher.match(queryDescriptors=desc1, trainDescriptors=desc2)
    # The matches with shorter distance are the ones we want
    matches = sorted(matches, key=lambda x: x.distance)
    return matches


def own_matcher(desc1, desc2):
    matches = []
    for i, k1 in enumerate(desc1):
        for j, k2 in enumerate(desc2):
            matches.append(cv.DMatch(_distance=np.linalg.norm(k1 - k2),
_imgIdx=0, _queryIdx=i, _trainIdx=j))
    matches = sorted(matches, key=lambda x: x.distance)
    return matches


images = [{"train": 'images/train.png', "test": 'images/test.png'},
          {"train": 'images/train1.png', "test": 'images/test1.png'}]
for img in images:
    original_image = cv.cvtColor(cv.imread(img["train"]), cv.COLOR_BGR2RGB)
    distorted_image = cv.cvtColor(cv.imread(img["test"]), cv.COLOR_BGR2RGB)
    key1, desc1, key2, desc2 = brisk_descriptor(original_image, distorted_image)
    matches1 = brute_force_matcher(desc1, desc2)
    draw_matches(matches1, key1, key2, original_image, distorted_image)
    matches2 = own_matcher(desc1, desc2)
    draw_matches(matches2, key1, key2, original_image, distorted_image)
```