

Міністерство освіти і науки України  
Національний технічний університет України  
«Київський політехнічний інститут імені Ігоря Сікорського»

Інститут прикладного системного аналізу  
Кафедра математичних методів системного аналізу

### Звіт

про виконання лабораторної роботи №3  
з дисципліни “Розпізнавання образів”

Виконали:  
Студенти 4 курсу  
Групи КА-76 і КА-72  
Борбела Артур  
Фалілеєва Дар’я

Київ – 2020

## Завдання

2) Маючи досягнення з попередньої серії, докрутити до лаби **класифікатор** на ваш особистий смак, натренувати його на ваш об'єкт (або ж якщо є час і натхнення то на інший), за необхідності розширюючи вибірку, і на об'єкт сусіда з таким же класифікатором на основі алгоритмів отримання характеристик зображення. Після того зробити дві речі: прогнати отримане комбо з класифікатора і дескриптора на тестовій вибірці збираючи як позитивні результати так і помилки першого і другого роду та замірюючи швидкодiю; і записати демонстраційне відео з вашим об'єктом та згодувати його цьому ж комбо записуючи у окремий файл результат роботи (скажімо те ж відео з текстовим оверлеєм). Далі знаючи особливості вашого дескриптора колективно з кимсь у кого такий же класифікатор проаналізувати особливості класифікатора в комбінації з вашими дескрипторами. Бонусні очки за використання кількох класифікаторів (видів класифікаторів) і/або запис відео з візуалізацією процесу розпізнавання/детекції.

**Обмеження** таке ж як в попередній лабі, пари алгоритмів мають бути унікальними в кожній команді. Команди в тому ж файлику, що й на попередню лабу, в іншій вкладці.

Докрутити алгоритм класифікації напряду до ключових точок ми не можемо.

1) оскільки кількість їхня різна а масив вхідних параметрів повинен бути для класифікатора однакової довжини.

2) ми можемо рахувати їхню кількість але це приведе до того ж результату що й в попередній роботі.

Вихід генерувати ключові точки й не матчити його з оригіналом, а спробувати їх кластеризувати розраховуючи на те що деякий кластер (або декілька схарактеризує нам шуканий об'єкт) тобто в нас буде масив довжина якого рівна кількості кластерів  $X_{train}$ .

Після того як ключові точки зібрані по кластерам ми маємо задати єдине значення для кластера (число а не масив точок) нам нічого не остається окрім взяти середня значення для координат цих ключових точок даного кластера.

Тобто на вхід нашому класифікатору будемо подавати дані в вигляді

```
X_train [9.10000000e+01 8.26666667e+01 7.80000000e+01
8.40000000e+01
5.00000000e+01 9.20000000e+01 1.27000000e+02 4.30000000e+01
1.16000000e+02 4.40000000e+01]
Y_train [1, 0, 0, 1, 0, 1, 1, 1]
```

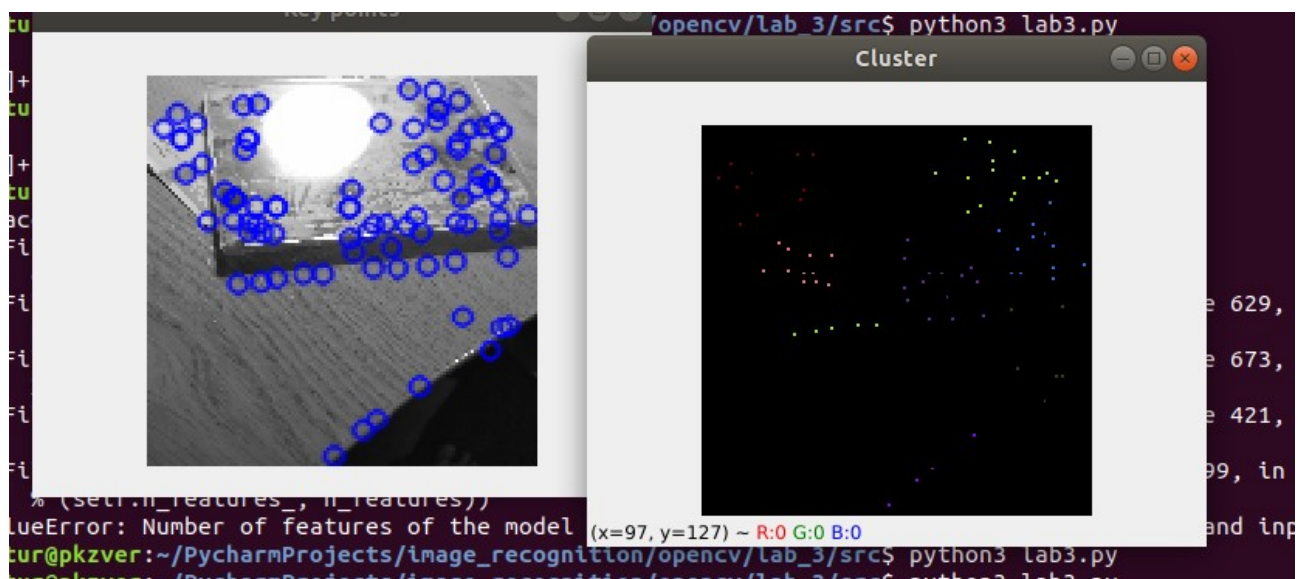
Для кластеризації був вибран алгоритм

```
from sklearn.cluster import Kmeans
```

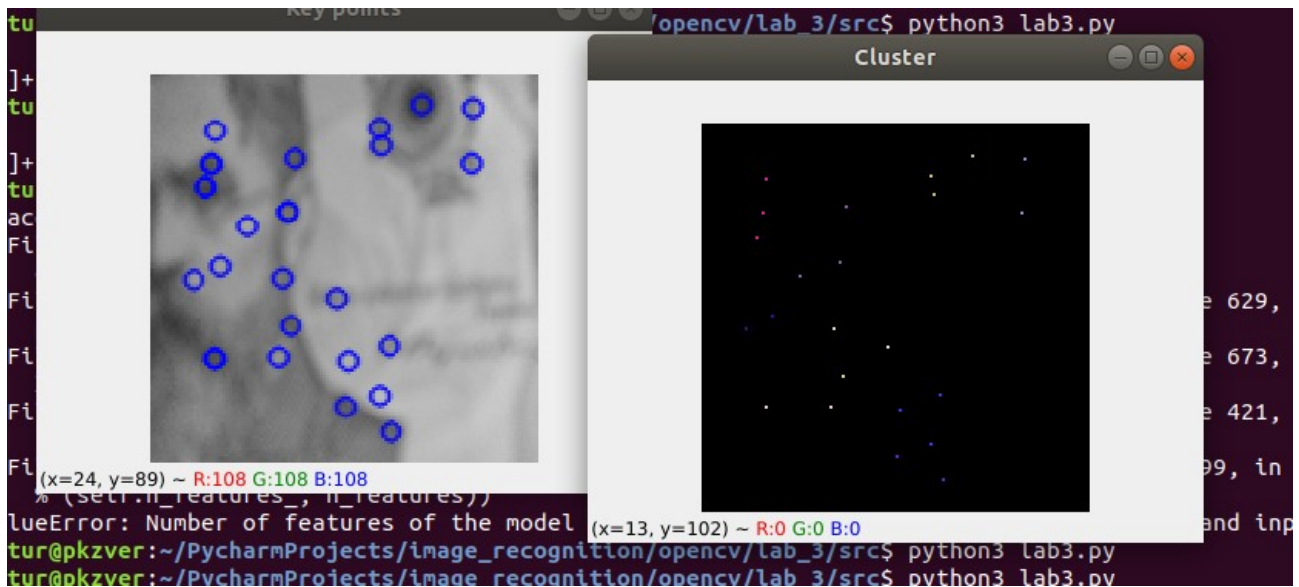
Перша проблема була в тому що для різних датасетів потрібно було різне значення кластерів

Наприклад для першого датасету ключові точки можна було з лехкістю розбити на 8 кластерів.

## SIFT

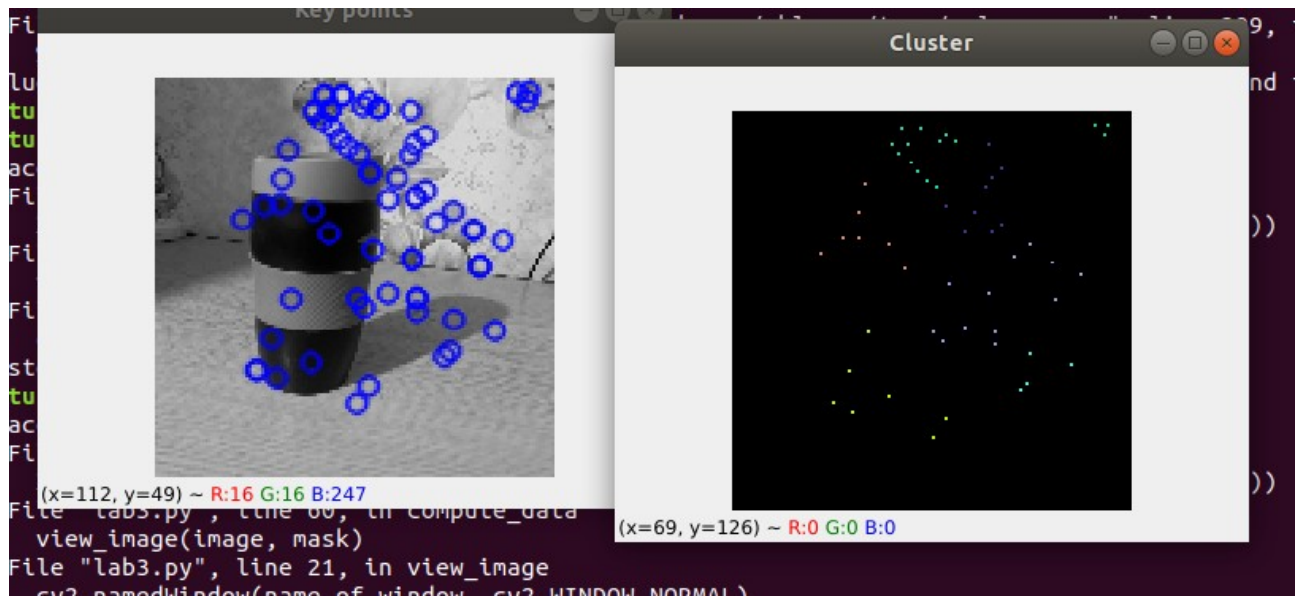


## ORB

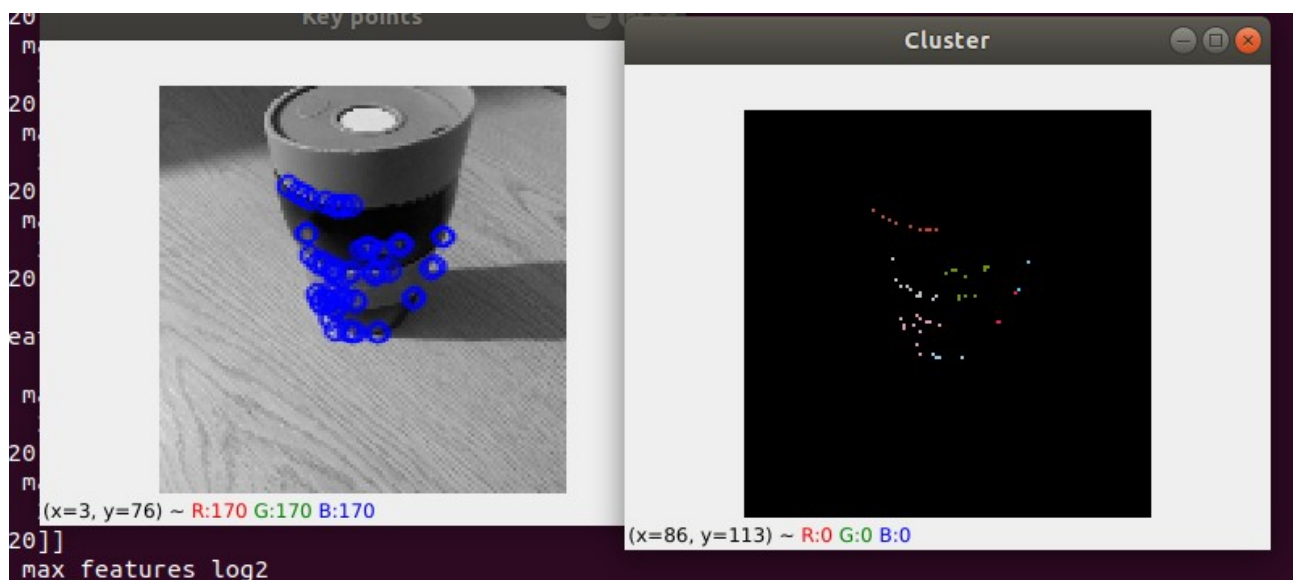


А для другого датасету це виявилось набагато складніше

SIFT



ORB



Причина цьому генерація малої кількості ключових точок

Яка при поєднанні кластеризації отримувала

**ConvergenceWarning: Number of distinct clusters (6) found smaller than n\_clusters (8). Possibly due to duplicate points in X.**

Дуже інформативний warning який виявився повідомленням про те що в даному випадку кластеризувати на вказану певну кількість кластерів алгоритм не може що в подальшому зашкоджує логіці алгоритму. (той випадок коли в нас 8 кластерів і 8 точок і одна з них майже повністю налягується на іншу).

Прийшлося вибирати оптимальне число кластерів при якому я втрачав найменше фотографій з датасету 2. Оптимальна кількість кластерів вийшла 8 (при ній я втрачав лише 3 фотографії з дескриптором SIFT, для ORB все було краще)

Для алгоритму класифікації був вибран

`from sklearn import ensemble RandomForestClassifier`

(тому що вже був певний досвід роботи з ним)

Для налаштування параметрів алгоритму класифікації була вибрана метрика

```
from sklearn.metrics import confusion_matrix
```

Параметри класифікації

Параметр `max_depth` — визначає максимальну глибину дерева.

Тобто при задачі класифікації вибірка об'єктів розділяється по певному критерію поки елементи в листі дерева не будуть належати до одного класу, що спричиняє перенавчання (RandomForest дуже схильний до перенавчання) параметр `max_depth` задає зупинку розбиття, тобто після кількох ітерацій розбиття закінчується.

`Min_samples_split` параметр який задає мінімальну кількість розбиття.

`Max_features` (за замовчуванням = «auto»))

Кількість функцій які потрібно врахувати при пошуці кращого розбиття

- Якщо «auto», то `max_features=sqrt(n_features)`.
- Якщо «sqrt», то `max_features=sqrt(n_features)`.
- Якщо «log2», то `max_features=log2(n_features)`.
- Якщо немає то `max_features=n_features`.

Оскільки RandomForest має схильність до перенавчання велике значення `max_depth`, `Min_samples_split` погано сказується на тестову оцінку.

Тренувальна вибірка для кращого аналізу роботи класифікатора була двох видів (тестова так же само)

- 1) 29% - wrong data; 71% - correct data
- 2) 60% - wrong data; 40% - correct data

Дані знаходяться в файлі randomForestClassifier.txt

Приклад

// DATASET\_1 ORB X\_train: 29% - wrong data; 71% - correct data

Max\_depth

Param max\_depth 1

[[ 1 3]

[ 9 15]]

Param max\_depth 2

[[ 2 1]

[ 8 17]]

Param max\_depth 3

[[ 2 2]

[ 8 16]]

Param max\_depth 4

[[ 2 2]

[ 8 16]]

Param max\_depth 5

[[ 2 2]

[ 8 16]]

Param max\_depth 6

[[ 2 2]

[ 8 16]]

Param max\_depth 7

[[ 2 2]

[ 8 16]]

Param max\_depth 8

[[ 2 3]

[ 8 15]]

Param max\_depth 9

[[ 2 3]

[ 8 15]]

Param max\_depth 10

[[ 2 3]

[ 8 15]]

Param max\_depth 11

[[ 2 3]

[ 8 15]]

Param max\_depth 12

[[ 2 3]

[ 8 15]]

Param max\_depth 13

[[ 2 3]

[ 8 15]]

Param max\_depth 14

[[ 2 3]

[ 8 15]]

Param max\_depth 15

[[ 2 3]

[ 8 15]]

Param max\_depth 16

[[ 2 3]

[ 8 15]]

Param max\_depth 17

[[ 2 3]

[ 8 15]]

Param max\_depth 18

[[ 2 3]

[ 8 15]]

Param max\_depth 19

[[ 2 3]

[ 8 15]]

Max\_features

Param max\_features auto

[[ 2 3]

[ 8 15]]

Param max\_features sqrt

[[ 2 3]

[ 8 15]]

Param max\_features log2

[[ 2 3]

[ 8 15]]

Min\_samples\_split

Param min\_samples\_split 0.1

[[ 2 2]

[ 8 16]]

Param min\_samples\_split 0.2

[[ 2 4]

[ 8 14]]

Param min\_samples\_split 0.3

[[ 2 3]

[ 8 15]]

Param min\_samples\_split 0.4

[[ 2 3]



```
[ 8 15]]
Param min_samples_split 0.5
[[ 1  3]
 [ 9 15]]
Param min_samples_split 0.6
[[ 1  3]
 [ 9 15]]
Param min_samples_split 0.7
[[ 0  0]
 [10 18]]
Param min_samples_split 0.8
[[ 0  0]
 [10 18]]
Param min_samples_split 0.9
[[ 0  0]
 [10 18]]
```

Confusion\_matrix[0][0] — правильно выбрано wrong data  
Confusion\_matrix[0][1] — помилка першого роду  
Confusion\_matrix[1][0] — помилка другого роду  
Confusion\_matrix[1][1] — правильно выбрано correct data

При тренувальному наборі 29% - wrong data; 71% - correct data

Найкращий показник для ORB

DATASET\_1

```
[[ 2  2]
 [ 8 16]]
```

DATASET\_2

[[ 7 5]

[ 2 15]]

З певного порога min\_samples\_split 0.7

[[ 0 0]

[ 9 20]]

[[ 0 0]

[10 18]] — що значно гірше оскільки алгоритм завжди каже correct data (із-за перенавчання) хоч accuracy\_score більше

Найкращий показник для SIFT

DATASET\_1

[[ 0 2]

[ 9 17]]

DATASET\_2

[[ 1 1]

[ 4 23]]

З певного порога в нього також сама проблема як і в ORB

При тренувальному наборі : 60% - wrong data; 40% - correct data

Найкращий показник для ORB

DATASET\_1

[[26 3]

[ 1 14]]

DATASET\_2

[[22 3]

[ 2 17]]

Найкращий показник для SIFT

DATASET\_1

[[20 6]

[ 6 12]]

DATASET\_2

[[19 4]

[ 4 17]]

Аналізуючи даний результат можна сказати що класифікатор працює але такий результат являється поганим. Якщо його перенавчити то він тупо вказує на щось одне типу такий предмет є.

Параметри для класифікатора які виявились найкращими це  
min\_samples\_split 0.3 ; max\_depth=5;

ORB для всіх випадків виявився дещо кращим за SIFT. Що в принципі можна пояснити тим що обидва дескриптора генерували багатенько ключових точок (в попередній роботі SIFT добре матчився а ORB ні).

accuracy\_score, f1\_score , recall\_score — можна було для статистики теж виводити але confusion\_matrix дає найкраще представлення оскільки данні метрики генеруються на основі неї.

Тестова вибірка в мене була 20% тому що це такий певно мовити баланс, оскільки чим більше ми витрачаєм на навчальну вибірку тим модель краще працює але постає проблема неточного результату на тестовій вибірці. Впринципі завжди сходяться до того що 80/20 краще співвідношення.

Про залежність від вмісту на фотографії:

Розмитість, затемненість та всі інші фактори які були порівнянні в попередній роботі тут також грають важливу роль. Ці негативні фактори(освітленість, розмитість) безпосередньо погано впливають на обидва дескриптора.( Тепер чим більше ключових точок згенерує дескриптор тим більша ймовірність що даний предмет розпізнється). Відповідно якщо є негативні фактори то ключових точок генерується набагато менше.

З попередньої роботи відомо що ORB відпрацьовує набагато швидше за SIFT, із-за цього можна сказати що перемога на користь ORB. Тому що головною вимогою до обох цих дескрипторів є генерування однакової кучності ключових точок на один і той самий предмет в незалежності від інших факторів а чи будуть вони матчитись в нас не грає ролі.

Тому для аналізу відео використовував ORB.

Відео знаходяться в папці src

Хотя судячи з відео результату майже не видно. Ужасно