Київський національний університет імені Тараса Шевченка
Факультет комп'ютерних наук та кібернетики
Кафедра дослідження операцій

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА
за спеціальністю 113 «Прикладна математика»
на тему:

Пошук аналогів у зображенні

студента 4 курсу
Тадлі Михайла Костянтиновича

Науковий керівник:
доцент, кандидат фізико-математичних наук
Матвієнко В. Т.

Робота заслухана на засіданні кафедри дослідження операцій та рекомендована до захисту в ЕК, протокол № 9 від 23.05.2023 р.

Завідувач кафедри ДО проф. Іксанов О. М.

Київ – 2023

ВСТУП

Людина з малого віку вміє розрізняти предмети за їх зовнішнім виглядом та описувати їх властивості.

Виявлення бракованих деталей, підрахунок кількості клітин під мікроскопом, розпізнавання обличчя працівника організації для гарантування доступу до кабінету, розрахунок кількості, напрямку та швидкості руху ворожої колони техніки під час війни – приклади задач, які людина може виконувати за допомогою зору.

Бурхливий розвиток комп’ютерів за останні кілька десятків років змушують людей дедалі більше думати про заміну людської праці на комп’ютерну в різних областях для швидкого та точного вирішення поставлених проблем.

Для цього нам потрібно розібратися із проблемою навчання комп’ютера автоматично проводити аналіз та розпізнавання зображень, тобто знаходити підзображень на картинці, класифікувати їх, вилучати шуми, шукати деякі значущі характеристики, тощо. У цій роботі автор більш детально зупиняється на вирішенні першого завдання – пошук аналогів у зображенні.

РОЗДІЛ 1

ПОСТАНОВКА ЗАДАЧІ ПОШУКУ АНАЛОГІВ У ЗОБРАЖЕННІ

Для вирішення проблеми класифікації об'єкту нам потрібно визначити його місцезнаходження на зображенні. У пошуку таких місць і буде полягати задача пошуку аналогів у зображенні.

Для початку визначимо, що таке зображення. Цифрове зображення зберігається у комп'ютері у вигляді матриці розмірів $M \times N$, де M – це висота зображення, а N – довжина. Елементами матриці кольорового зображення є трійки цілих чисел r, g, b, кожне з яких лежить на відрізку [0, 255], вони є інтенсивностями вкладу червоного, зеленого та синього кольорів відповідно. Елементами матриці чорно-білого зображення є лише одне ціле число з діапазону [0, 255].

В роботі буде зручно ототожнювати зображення та двовимірну дискретну функцію визначену на множині $([0, M-1] \cap Z) \times ([0, N-1] \cap Z)$, значеннями в точці (x_0, y_0) будемо вважати вектор з трьох чисел для кольорового зображення, та одного числа для чорно-білого. Для таких функцій множення на скаляр, додавання та добуток визначаються у звичайний спосіб.

Формалізуємо постановку задачі. Задача пошуку аналогів у зображенні полягає у знаходженні усіх підзображень w_i на зображенні $f(x, y)$, які схожі на еталон $w(x, y)$. Розв'язок цієї задачі автор представляє у вигляді множини зображень, які описуються множиною точок $L = \{(x_1, y_1), (x_2, y_2), \dots, (x_k, y_k)\}$, де (x_i, y_i) відповідатиме за центр підзображення w_i. Розміри підзображень w_i співпадають з розмірами еталону.

РОЗДІЛ 2

МЕТОДИ ТА АЛГОРИТМИ ПОШУКУ АНАЛОГІВ У ЗОБРАЖЕННІ

2.1. Кореляційне порівняння зображень

Розглянемо дві функції $f(x,y)$ та $w(x,y)$ розмірами $M \times N$, визначені на множині $([0,M-1] \cap Z) \times ([0,N-1] \cap Z)$, та які мають область значень $[0{,}255] \times [0{,}255] \times [0{,}255]$.

Кореляцією двох дискретних функцій $f(x,y)$ та $w(x,y)$ будемо вважати наступний вираз:

$$w(x,y) \circ f(x,y) = \frac{1}{MN} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} w(m,n) f(x+m, y+n), \qquad (2.1.1)$$

де $w(m,n)f(x+m,y+n)$ позначає скалярний добуток векторів значень функцій.

Основною задачею, яку вирішує кореляція, є задача суміщення, тобто точного виявлення місця, де знаходиться еталон $w(x,y)$ на зображенні $f(x,y)$. Якщо зображення $f(x,y)$ містить $w(x,y)$, то кореляція між ними буде мати максимум в точці де буде повне співпадіння.

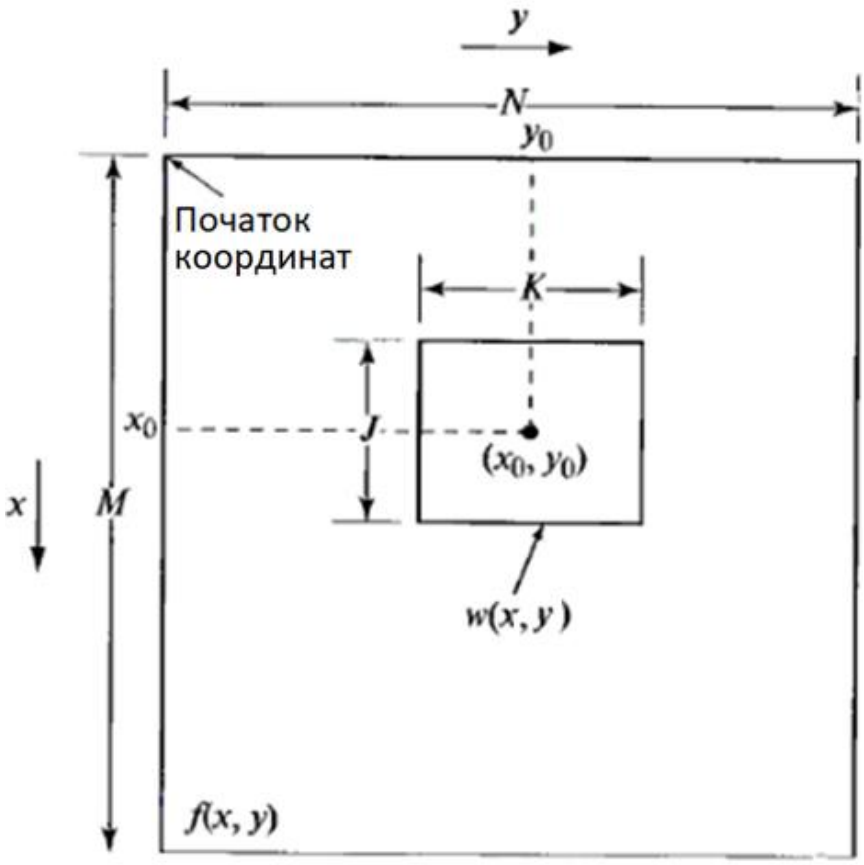
Тут і далі розглядаємо випадок кольорових зображень, оскільки для чорно-білих все буде аналогічно.

Кореляція для двох зображень $f(x,y)$ та $w(x,y)$ з розмірами $M \times N$ та $J \times K$ відповідно, задається виразом:

$$c(x,y) = \sum_s \sum_t w(s,t) f(x+s, y+t) \qquad (2.1.2)$$

для $x = 0{,}1, \dots, M-1$, $y = 0{,}1, \dots, N-1$, а сумування ведеться по області перетину зображень f та w.

Нижче наведена ілюстрація застосування кореляції, якщо вважати, що початок координат лежить у верхньому лівому кутку. Для кожної точки (x_0, y_0) зображення f підраховується $c(x_0, y_0)$.



Мал. №1

Кореляція також може бути застосована і для задачі пошуку аналогів у зображенні. В точках, де зображення $f(x,y)$ містить схоже на $w(x,y)$ підзображення w_i значення кореляції буде локальним максимумом, а отже їх можна буде виділити і отримати розв'язок.

З Мал. №1 видно, що при обчисленні значення кореляції на краях картинки обчислення ведуться поза областю визначення f. Для подолання цієї проблеми є кілька рішень, ось деякі з них:

1. Доповнити f константними значеннями (наприклад (0,0,0)).
2. Відобразити (наприклад продублювати або віддзеркалити) кордони f для доповнення.
3. Не вести підрахунок кореляції для випадків коли виникає невизначеність.

Перші два варіанти призводять до похибок кореляції пропорційній до частини площу еталона який виходить за межі недоповненої f. Останнє рішення призведе до втрати інформації.

Кореляція задана рівнянням (2.1.2) є чутливою до змін амплітуд f та w. Тобто при збільшенні значення f $c(x, y)$ також збільшиться пропорційно, а отже і відобразиться на пошуку максимумів, що призведе до неточностей. Щоб позбутись цього ефекту часто користуються нормованим відносно змін амплітуд функцій коефіцієнтом кореляції, який задається так:

$$\gamma(x,y) = \frac{\sum_s \sum_t [f(x+s, y+t) - \bar{f}(x+s, y+t)][w(s,t) - \overline{w}\,(s,t)]}{\sqrt{\sum_s \sum_t [f(x+s, y+t) - \bar{f}(x+s, y+t)]^2 \sum_s \sum_t [w(s,t) - \overline{w}\,(s,t)]^2}}, \quad (2.1.3)$$

де $\overline{w}$ – середнє значення еталону, а $\bar{f}$ – середнє значення елементів в області, для якої обраховується коефіцієнт, $s \in [0, J-1], t \in [0, K-1]$.

Коефіцієнт кореляції змінюється від -1 до 1 та не залежить від зміни амплітуд f та w. Чим ближче значення коефіцієнту до 1 в точ, тим більш схожим вважається область, для якої воно підраховувалося, на еталон.

Після підрахунку коефіцієнту кореляції потрібно знайти локальні максимуми значень цього коефіцієнту. Це робиться для виявлення найкращих співпадінь.

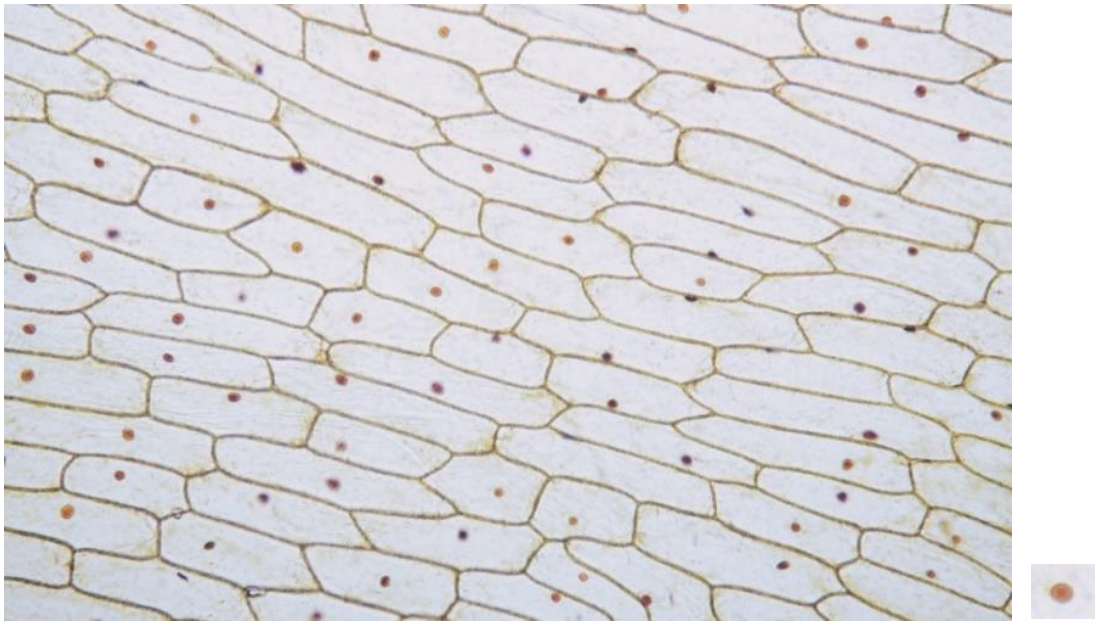
Для знаходження локальних екстремумів потрібно пройтись рамочкою $J \times K$ по всьому зображенню, та перевірити чи є центральна точка кожної рамочки максимальною в ній.

Для того, щоб не виявляти співпадіння в областях де коефіцієнт кореляції досить малий, потрібно відкидати такі значення, для цього можна обрати порогове число, усі значення, які будуть нижче за це число розглядатись не будуть.

Нормування відносно зміни апмлітуд досягається досить просто. Проте досягнення того самого відносно зміни масштабів або повороту не є простим. Для правильної зміни масштабу потрібно знати правильне співвідношення розмірів еталона та шуканого об’єкту, що не є відомим без

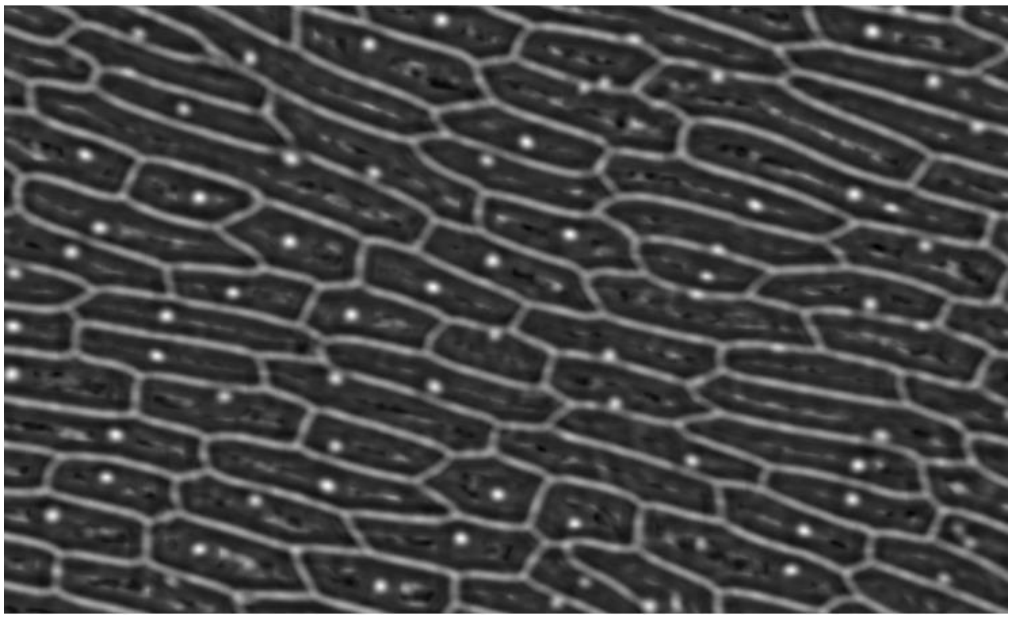
додаткової інформації про зображення. Нормування відносно повороту аналогічно потребує додаткових знань про картинку. В загальному ж випадку таке нормування призводить до трудомістких обчислень додатковими методами пошуку ознак на картинці, вказано у [2].

Приклад: На рисунку нижче представлено застосування коефіцієнту кореляції. Мал. №2.1 – зображення f, Мал. №2.2 – еталонне зображення w. На Мал. №3 представлено обчислені значення коефіцієнту кореляції. Задля уникнення невизначеностей та похибок підрахунки велись лише для областей повного перетину еталонного зображення та f. Чим світліша точка на малюнку, тим ближчий коефіцієнт до 1. На Мал. №4 обведені усі області w_i. Порогове значення було обране за 0.65.

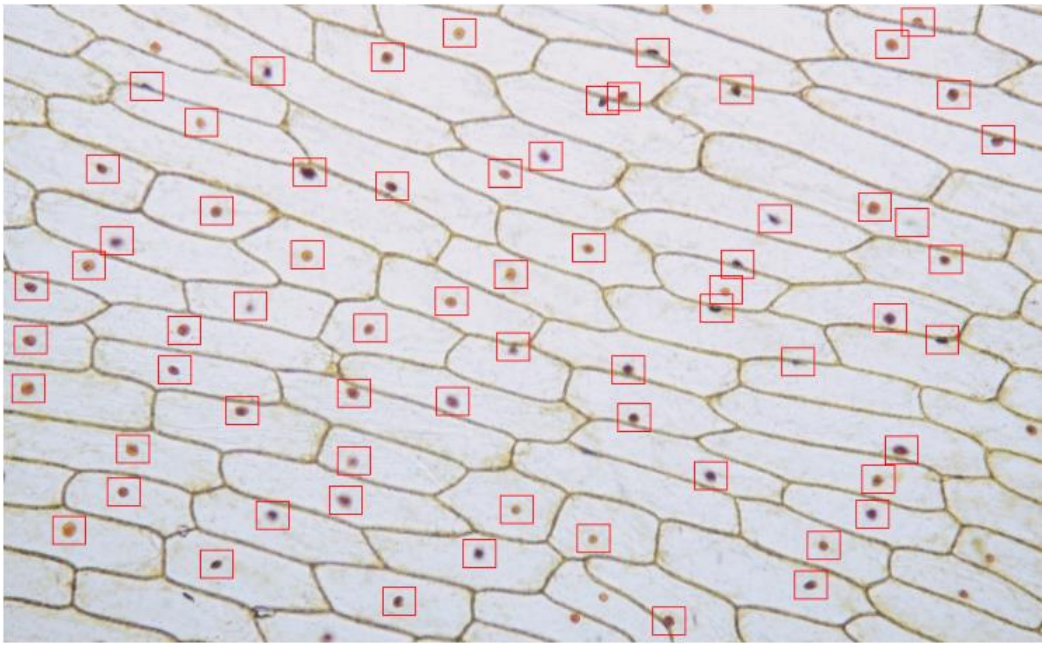


1 2

Мал. №2



Мал. №3



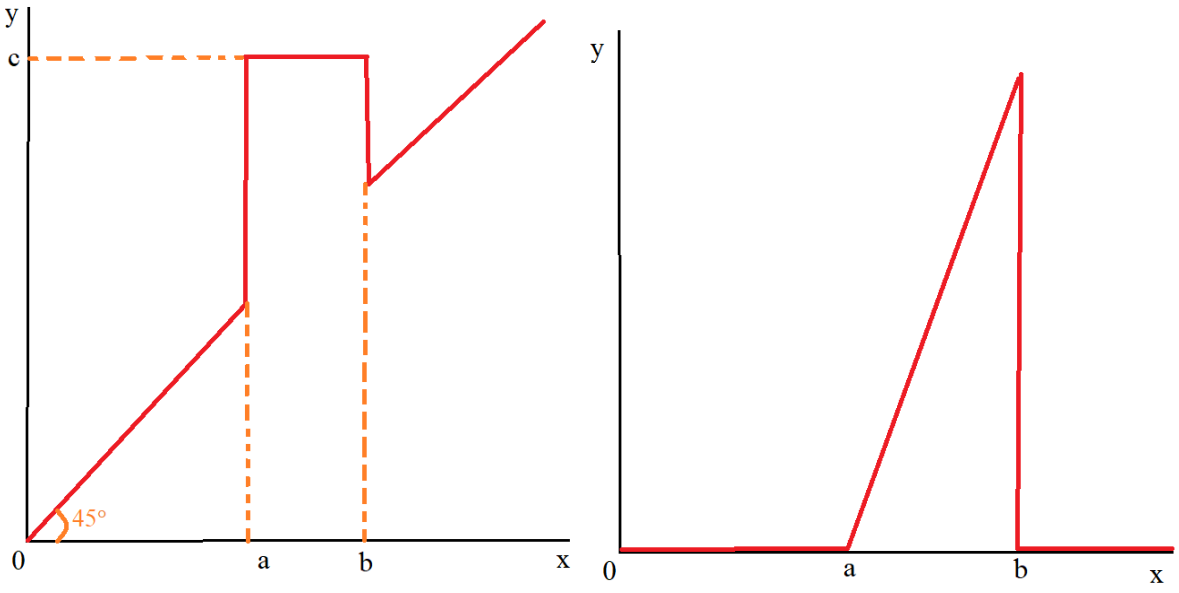
Мал. №4

2.2 Препарування зображення

Операція препарування являє собою клас поелементних перетвореннь зображень. Їх застосовують для зменшення перепадів інтенсивностей в деяких областях та підвищення діапазону зміни інтенсивностей в інших, з метою позбутись від шумів, зробити картинку більш контрастною, покращити сприйняття картинки людиною. Зазвичай таку операцію проводять для чорно-білого зображення.

Наприклад графік на Мал. №5.1 відображає операцію препарування, яка для кожного значення інтенсивності x оригінального зображення переводить його в y, у випадку цього графіка, будуть збережені усі інтенсивності оригінального зображення, окрім тих, що лежать на проміжку [a,b]. Їх значення на новому зображенні будуть рівні c.

Мал. №5.2 зображує операцію препарування, яка кожна значення інтенсивності x оригінального зображення, яке не лежить на відрізку [a,b], перетворить у 0. Значення, які лежать на [a,b], будуть відображені на весь діапазон [0,255].

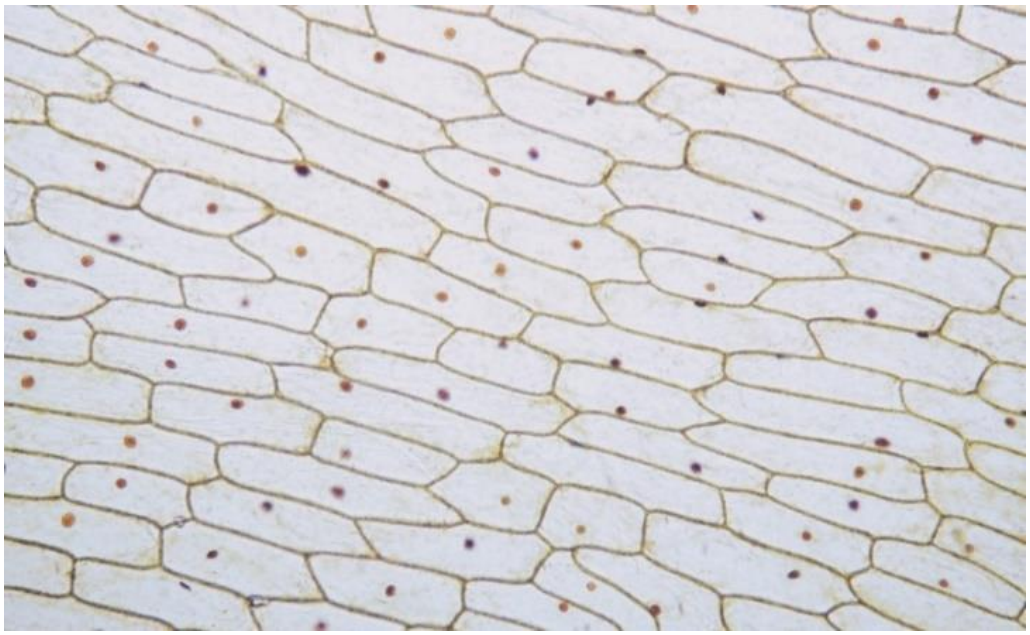


1 2

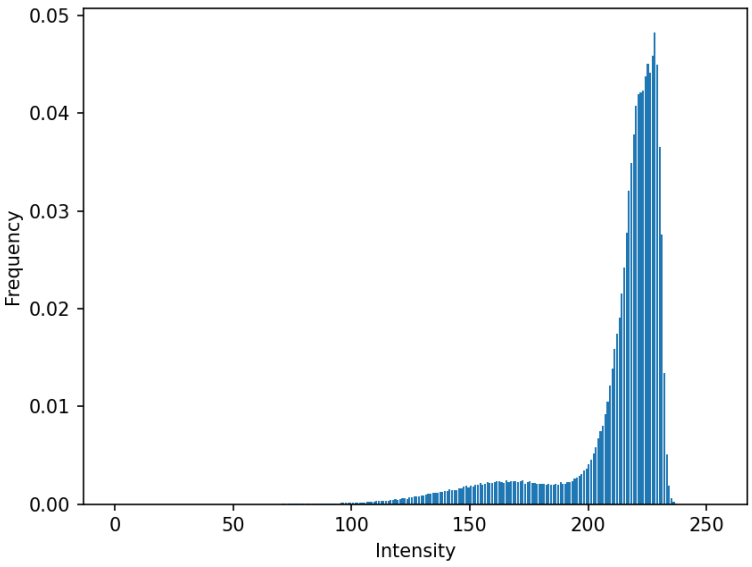
Мал. №5

Приклад: Далі наведений приклад застосування препарування зображення операцією з Мал. №5.2.

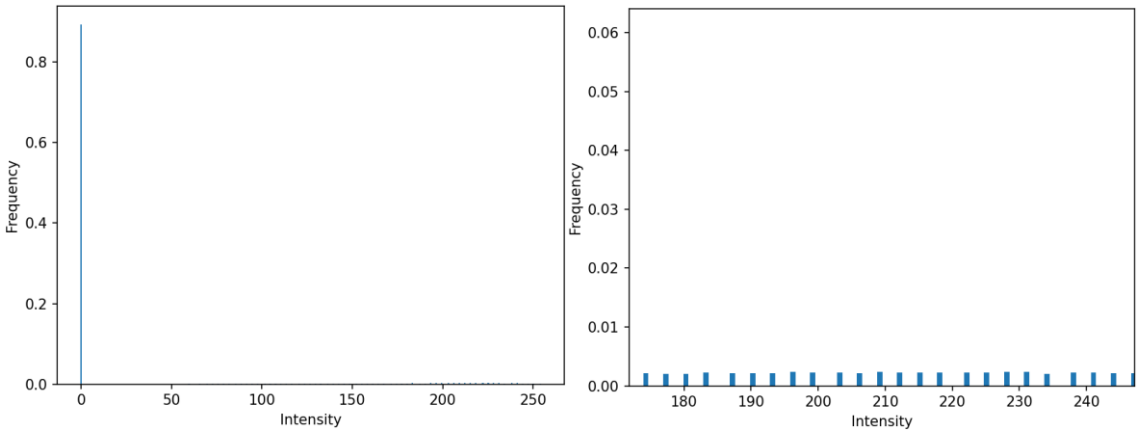
Мал. №6 – оригінальне зображення. Мал. №7– гістограма частот інтенсивностей оригінального зображення. Мал. №8.1 – гістограма частот після препарування з Мал. №5.2, де a=100, b=170, на Мал. №8.2 показана збільшена область гістограми для демонстрації інтенсивностей у тому проміжку. На Мал. №9 зображення після препарації. Значення a та b були визначені, як локальні мінімуми гістограми на проміжку [100,200].



Мал. №6

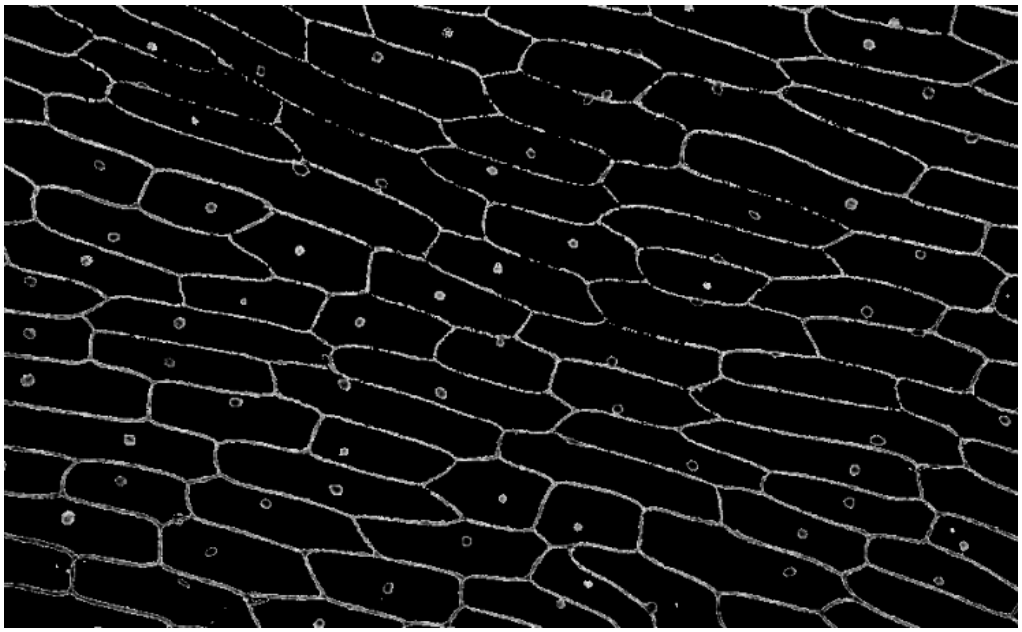


Мал. №7



1 2

Мал. №8



Мал. №9

Гістограма частот – графік, на якому зображені частоти кожної інтенсивності, яка є на зображенні.

2.3 Інваріантні моментні ознаки

Розглянемо чорно-біле зображення f розмірами $M \times N$, визначене на множині $([0, M-1] \cap Z) \times ([0, N-1] \cap Z)$ та областю значень [0,255]. Тоді двовимірний момент $(p+q)$-го порядку для цього зображення визначається як у працях [1, 2]:

$$m_{pq} = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} x^p y^q f(x, y), \tag{2.3.1}$$

де p, q цілі невід'ємні числа. Центральний момент порядку $(p+q)$:

$$\mu_{pq} = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} (x - \bar{x})^p (y - \bar{y})^q f(x, y), \tag{2.3.2}$$

де $\bar{x} = \frac{m_{10}}{m_{00}}$, $\bar{y} = \frac{m_{01}}{m_{00}}$.

Нормовані центральні моменти визначаються так:

$$\eta_{pq} = \frac{\mu_{pq}}{\mu_{00}^{\gamma}}, \tag{2.3.3}$$

де $\gamma = \frac{p+q}{2} + 1$.

Інваріантні моменти відносно зсуву, дзеркальному відображенню, повороту та масштабу:

$$\phi_1 = \eta_{20} + \eta_{02} \tag{2.3.4}$$

$$\phi_2 = (\eta_{20} - \eta_{02})^2 + 4\eta_{11}^2 \tag{2.3.5}$$

Схожим чином визначаються ще 5 моментів, проте використовують їх досить рідко.

Такі моменти можна використовують як класифікатор зображення. Хоча вирішення задач класифікації не є головною темою цієї роботи, автор хоче продемонструвати можливі подальші дії, які можна робити після вирішення задачі пошуку аналогів у зображенні.

Приклад: Підрахуємо та порівняємо інваріантні функції для наступних зображень:

1

2

3

4

Мал. №10

	ϕ_1	ϕ_2
Мал. №10.1	5.95441	12.51183
Мал. №10.2	5.93288	12.43804
Мал. №10.3	5.93702	12.44771
Мал. №10.4	5.78262	12.24016

Для кращого сприйняття значення ϕ_1 та ϕ_2 записані після перетворення

$$\phi_i = -\ln(\phi_i)$$

Значення функцій є досить близькими, не дивлячись на те, що відносно оригінального об’єкту Мал. №10.1 об'єкти на Мал. №10.2, Мал. №10.3 були повернуті, зменшені, дзеркально відображені та трохи зміщені. Мал. №10.4 показує інший об’єкт, не схожий на попередні. Дані наведені у таблиці вище для порівняння значень.

РОЗДІЛ 3

РОЗРОБКА ПРОГРАМНО-АЛГОРИТМІЧНОГО КОМПЛЕКСУ ДЛЯ РОЗВ'ЯЗКУ ЗАДАЧІ ТА РЕЗУЛЬТАТИ ЙОГО РОБОТИ

3.1 Програмна реалізація методів пошуку аналогів на зображенні

Програмно-алгоритмічний комплекс складається з чотирьох Python програм. Зупинимось на перших трьох. На вхід main.py програми подається два зображення формату .jpg чи .png перше з яких це початкове зображення f з розмірами $M \times N$, друге – еталонне w, з розмірами $J \times K$, яке потрібно відшукати на f, програма має повертати 2 зображення, одне, чорно-біле, відповідає підрахованих коефіцієнтам кореляції, інше – початковому зображенню з обведеними областями, де відбулися найкращі співпадіння зображення f та w. Якщо альфа-канал присутній, при проведенні розрахунків до уваги він не береться.

Після отримання зображень, main.py має 2 опції, перша – проводити розрахунки з кольоровим зображенням, або з чорно-білим. За необхідності можна перетворити кольорове зображення на чорно-біле за допомогою такої формули :

$$Y = 0.299R + 0.587G + 0.114B, \tag{3.1.1}$$

де Y – це значення яскравості нового пікселя, R,G,B – відповідно значення червоного, зеленого та синього кольорів у пікселя старого зображення. На практиці різниця між обчисленням коефіцієнту кореляції для кольорового зображення та для чорно-білої версії майже непомітна. Після визначення потрібного типу, інформація передається до calculations.py, де проводяться всі розрахунки.

За необхідності перед початком підрахунків можна провести препарування для підвищення контрастності та виділення потрібного об'єкту на еталонному зображенні. Для проведення препарування потрібно намалювати гістограму частот еталонного зображення, визначити які частоти потрібно виділити на ньому.

Для отримання зі значення коефіцієнта кореляції в точці (x,y) значення яскравості пікселя скористаємось наступною формулою:

$$p(x,y) = \frac{\gamma(x,y)+1}{2} * 255 \quad (3.1.2)$$

Таким чином, при значенні $\gamma(x,y) = 1$ будемо мати піксель зі значенням 255, яке відповідає білому кольору.

Для обчислення коефіцієнту кореляції ми скористаємось формулою

$$\gamma(x,y) = \frac{\sum_s \sum_t [f(x+s,y+t) - \bar{f}(x+s,y+t)][w(s,t) - \bar{w}\,(s,t)]}{\sqrt{\sum_s \sum_t [f(x+s,y+t) - \bar{f}(x+s,y+t)]^2 \sum_s \sum_t [w(s,t) - \bar{w}\,(s,t)]^2}}, \quad (3.1.3)$$

при цьому обраховуємо значення $w(s,t) - \bar{w}\,(s,t)$ та $\sum_s \sum_t [w(s,t) - \bar{w}\,(s,t)]^2$ лише 1 раз на початку програми, таким чином пришвидшуємо підрахунок коефіцієнту для кожної точки (x,y).

В даній програмі автор вирішив не підраховувати значення коефіцієнту кореляції, обчислення яких проводяться не повністю у межах початкового зображення f, задля уникнення додаткових хибних результатів.

Після підрахунку коефіцієнту кореляції, потрібно визначити локальні максимуми. Для цього для кожної точки (x,y) f перевіряємо чи відповідний коефіцієнт кореляції є максимальним серед сусідніх значень у прямокутнику. Прямокутник має розміри $\left[\frac{J}{2}\right] * 2 + 1 \times \left[\frac{K}{2}\right] * 2 + 1$, для нього точка (x,y) є центральною. Для знайдених максимумів перевіряємо чи є вони більші за деяке порогове значення і, якщо так, позначаємо їх як «правильні».

Після визначення «правильних» точок програма передає ці дані до outputs.py, у цій програмі відбувається малювання навколо цих точок прямокутників червоного кольору розмірів $\left[\frac{J}{2}\right] * 2 + 1 \times \left[\frac{K}{2}\right] * 2 + 1$.

Після чого картинки зі значеннями коефіцієнту кореляції та з обведеними точками виводяться на екран.

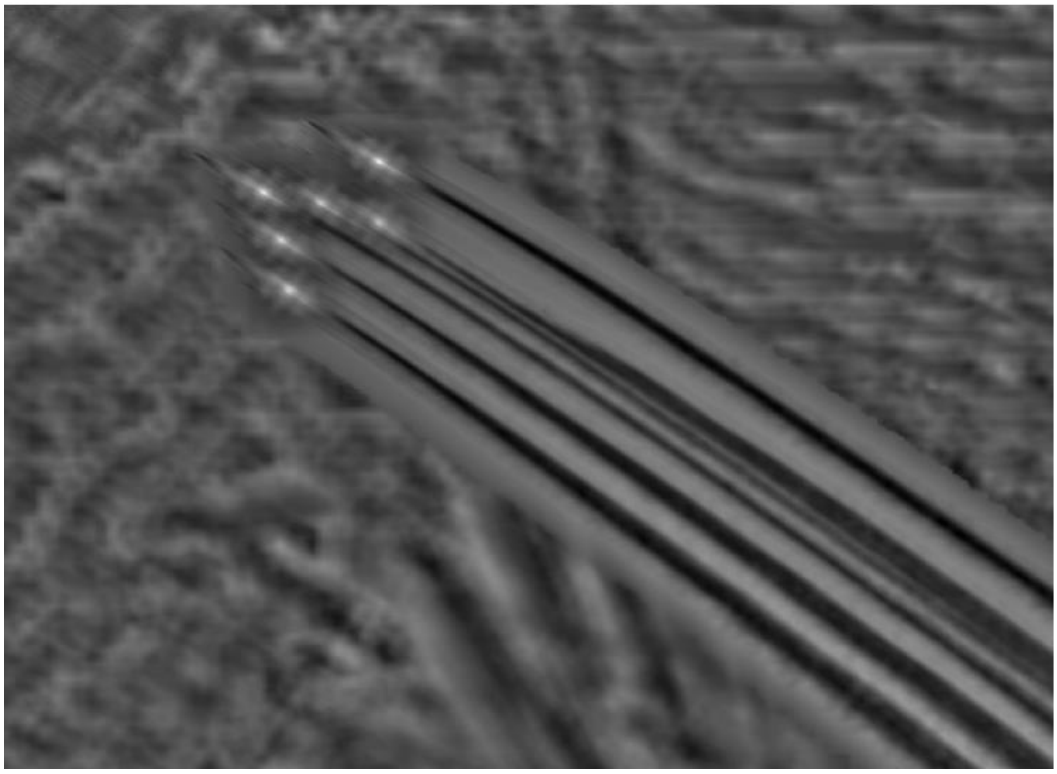
Остання програма потрібна для підрахунку моментних функцій та класифікації за ними чи є схоже на еталон зображення на картинці. На вхід вона приймає массив чорно-білих зображень отриманих шляхом вирізання виділених на попередньому кроці червоними прямокутниками підзображень f. Для правильної роботи цього методу краще обрати кілька правильних зображень та кілька зображень, які не містять схожого на наш об’єкт. Після цього можна буде відносити об’єкт до того класу (правильних зображень та ні) до якого він найближчий.

3.2 Результати роботи програми

Мал. №11 – зображення f, Мал. №12 – еталонне зображення w. Нище наведено результати роботи програми. На Мал. №13 представлено обчислені значення коефіцієнту кореляції. На Мал. №14 обведено усі локальні максимуми на картинці Мал. №13, які більші за порогове значення 0.55. Класифікація та препарування не застосовувались.

Мал. №11

Мал. №12



Мал. №13

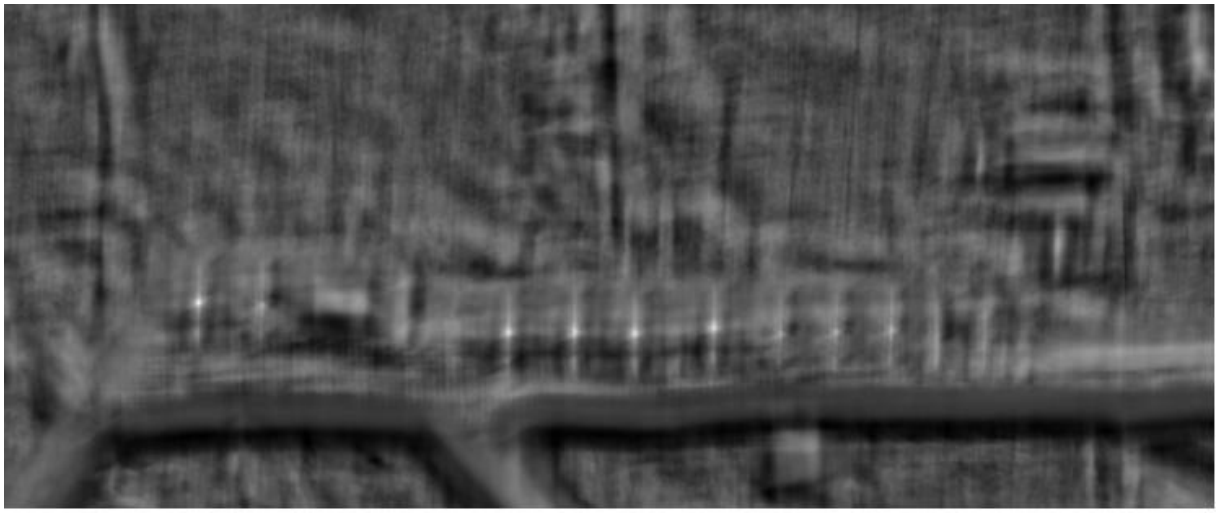
Мал. №14

На Мал. №13 чітко видно 6 максимумів яскравості, які і були відмічені на Мал. №14.

Розглянемо ще один приклад без застосування препарування та класифікації. На Мал. №15, 16 вхідні дані, на Мал. №17, 18 результати роботи програми з пороговим значенням 0.4.

Мал. № 15

Мал. №16



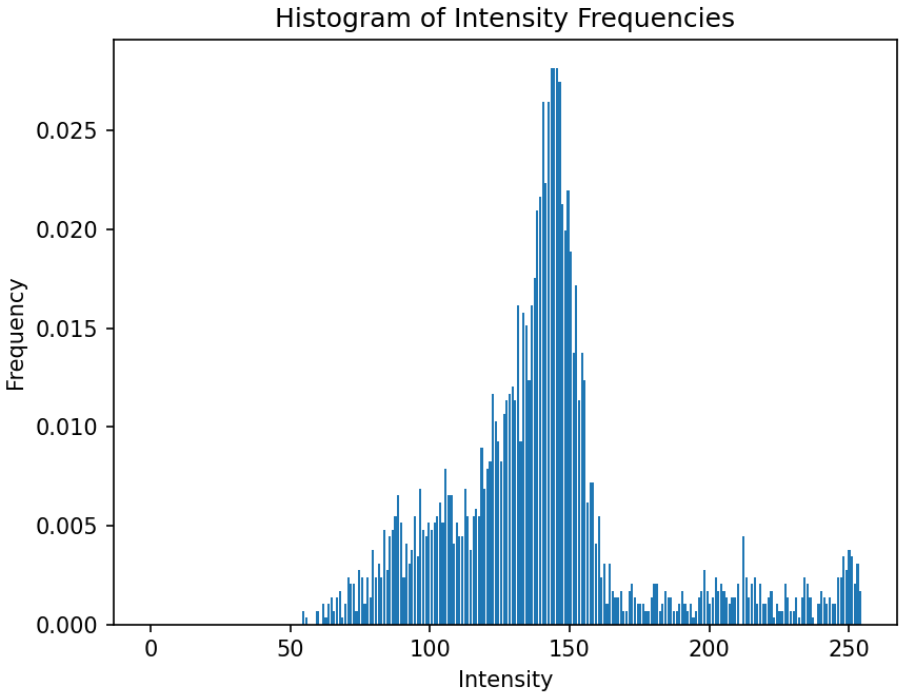
Мал. №17

Мал. №18

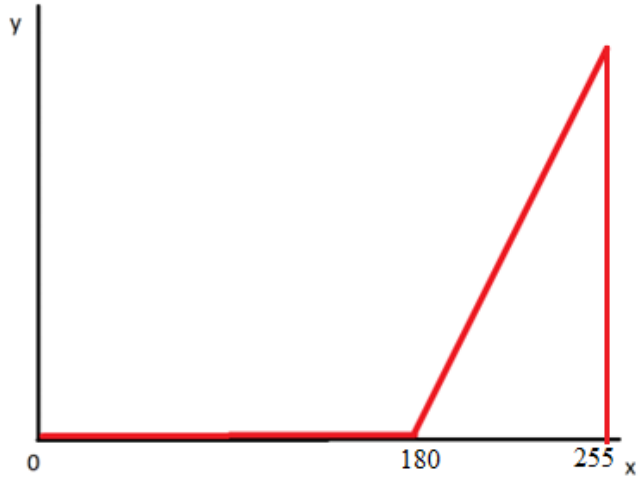
При підвищені порогового значення будуть втрачатись знайдені літаки. Як бачимо, в даній ситуації обчислення одного коефіцієнту кореляції не є достатнім для отримання гарних результатів. Спробуємо застосувати операцію препарування та класифікацію. На Мал. №19, 20 зображено чорно-біле перетворення зображень №15,16. На Мал. №21 зображено гістограму частот для Мал. №20. Оскільки літак білий, то значить основна інформація про нього зберігається в значеннях ближчих до 255. Локальний мінімум після 170 лежить у 180. Отже для препарування обираємо a=180, b=255. Воно зображено на Мал. №22. Результати застосування на Мал. №23,24.

Мал. №19

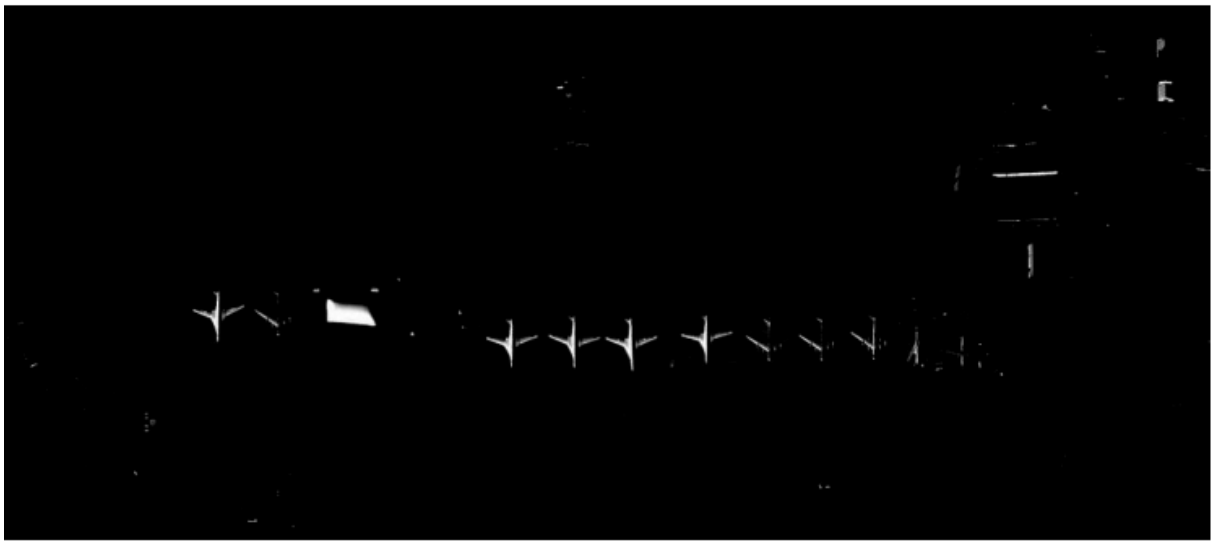
Мал. №20



Мал. №21



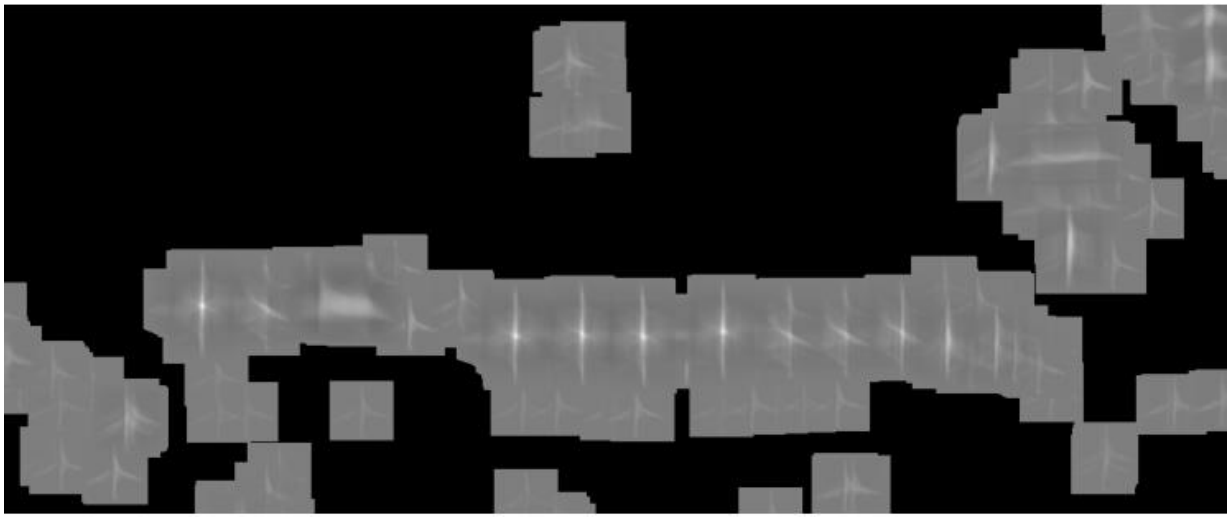
Мал. №22



Мал. №23

Мал. №24

Тепер подивимось на результати кореляційного співставлення після препарування. Воно зображено на Мал. № 25,26. Кількість невірних збігів зменшилась,



Мал. №25

Мал. №26

Тобто це те, що можна отримати застосовуючи лише методи пошуку аналогів. Застосуємо тепер класифікацію за допомогою першої інваріантної функції.

Для цього вирізаємо з зображення Мал. №15 декілька хатинок та літаків Мал. №27. Після цього обраховуємо перші інваріантні функції для цих зображень, а далі класифікуємо кожне підзображення з Мал. №26. Результати зображені на Мал. №28.

Мал. №27

Мал. №28

ВИСНОВКИ

В роботі розглянуто задачу пошуку аналогів у зображенні поставлена в пункті 1 розділу 1. Застосований програмний алгоритм є ефективним та простим. Результати для об’єктів на однорідному фоні, з високою контрастністю були точними. Але такі умови трапляються не часто. І для того, щоб більш точно визначити області схожих на еталон об’єктів на зображеннях з менш контрастним фоном, та деяким шумом, доводиться застосовувати препарування.

Загалом, якщо задача стоїть у спостереженні над одним і тим самим середовищем, можна відкалібрувати роботу алгоритму так, щоб точність була максимальною, але зазвичай це потребує додаткової інформації та людського втручання.

Алгоритм кореляційного співставлення зображень зміг виділити всі області зі схожими об’єктами на еталонний, тобто з поставленою задачею він впорався. Хоча він і не робить це ідеально, виділяючи об’єкти, які не відповідають еталонному, його робота досить сильно спрощує задачу класифікації та аналізу зображень у подальшому.

ЛІТЕРАТУРА

1. Reinhard Klette [2014]. Concise Computer Vision: An Introduction into Theory and Algorithms. vol 3, pp. 118

2. Gonzalez and. Richard E. Woods' [2018] Digital Image Processing, Fourth Edition, Global Edition. vol 11-12 pp. 858,915

Додаток

main.py

```
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
from calculations import calculate_correlation_colors,calculate_correlation_wb
R = (255, 0, 0)
R_T = (255, 0, 0, 255)
CORRELATION_COEF_THRESHOLD = 0.3

BOARDLESS=True
fl='planes'

def detect_similar_objects(im1, im2):


    correlation,im1_to_show = calculate_correlation_wb(im1, im2,fl)
    correlation_to_show=np.array((correlation+1)*255/2,dtype=np.uint8)

    plt.imshow(im1_to_show)
    plt.title("im1_to_show")
    plt.axis("off")
    plt.figure()

    plt.imshow(correlation_to_show, cmap='gray')
    plt.title("Correlation Result")
    plt.axis("off")
    if __name__ != '__main__':
        plt.show()

if __name__ == '__main__':
        im1 = Image.open('./pictures/planes.jpg')
        im2 = Image.open('./pictures/plane.png')
    plt.show()
```

calculations.py

```
import numpy as np
from PIL import Image
from outputs import BOARDLESS,draw_rectangles,show_3d_plot,hist
from classification import classify_images,preparation
from outputs import draw_rectangles as d_r
```

```
from outputs import draw_rectangles_classified as d_r_c

CORRELATION_COEF_THRESHOLD = 0.55


def extract_channels(image):

    arr = np.array(image)

    red_channel = np.array(arr[:, :, 0], dtype=np.float32)
    green_channel = np.array(arr[:, :, 1], dtype=np.float32)
    blue_channel = np.array(arr[:, :, 2], dtype=np.float32)

    return red_channel, green_channel, blue_channel

def calculate_correlation_colors(im1: Image, im2: Image):

    im1_r,im1_g,im1_b=extract_channels(im1)
    im2_r,im2_g,im2_b=extract_channels(im2)

    im2_height, im2_width = im2_r.shape
    padding = ((im2_height // 2, im2_height // 2), (im2_width // 2, im2_width //
2))
    im1_r_padded = np.pad(im1_r, padding, mode='constant',
constant_values=0)
    im1_g_padded = np.pad(im1_g, padding, mode='constant',
constant_values=0)
    im1_b_padded = np.pad(im1_b, padding, mode='constant',
constant_values=0)

    im2_r_norm = im2_r - np.mean(im2_r)
    im2_g_norm = im2_g - np.mean(im2_g)
    im2_b_norm = im2_b - np.mean(im2_b)
    im2_r_sq = im2_r_norm * im2_r_norm
    im2_g_sq = im2_g_norm * im2_g_norm
    im2_b_sq = im2_b_norm * im2_b_norm
    sig2 = np.sum(im2_r_sq+im2_g_sq+im2_b_sq)

    height, width = im1_g.shape
    correlation = np.zeros((height, width))

    for i, j in np.ndindex(height, width):
        patch_r = im1_r_padded[i:i + im2_height, j:j + im2_width]
```

```
        patch_g = im1_g_padded[i:i + im2_height, j:j + im2_width]
        patch_b = im1_b_padded[i:i + im2_height, j:j + im2_width]

        patch_norm_r = patch_r - np.mean(patch_r)
        patch_norm_g = patch_g - np.mean(patch_g)
        patch_norm_b = patch_b - np.mean(patch_b)

        patch_norm_r_sq = patch_norm_r ** 2
        patch_norm_g_sq = patch_norm_g ** 2
        patch_norm_b_sq = patch_norm_b ** 2
        sig1 = np.sum(patch_norm_r_sq+patch_norm_g_sq+patch_norm_b_sq)

        if sig1 == 0:
            sig1 = 1e-9

        corr_values = np.sum(patch_norm_r * im2_r_norm+patch_norm_g *
im2_g_norm+patch_norm_b * im2_b_norm) / np.sqrt(sig1 * sig2)

        correlation[i, j] = corr_values

    print('after_corr')
    if BOARDLESS:

        correlation=correlation[im2_height//2:height-im2_height//2,
im2_width//2:width-im2_width//2]

    local_maxima = find_local_extrema(correlation,h=im2_height,
w=im2_width,)


    show_3d_plot((correlation+1)*255/2)

    print(f'after_extrem {np.count_nonzero(local_maxima)}')

    im1_with_detected=draw_rectangles(im1, h=im2_height, w=im2_width,
bool_array=local_maxima)



    return correlation,im1_with_detected

def calculate_correlation_wb(im1: Image, im2: Image,fl):
```

```
    if np.array(im1).shape[2]>1:

        im1_wb=np.array(im1.convert('L'))
        im2_wb=np.array(im2.convert('L'))


        im1_wb=preparation(im1_wb)
        im2_wb=preparation(im2_wb)

    else:
        im1_wb=np.array(im1,dtype=np.uint8)
        im2_wb=np.array(im2,dtype=np.uint8)


    im2_height, im2_width = im2_wb.shape
    padding = ((im2_height // 2, im2_height // 2), (im2_width // 2, im2_width //
2))
    im1_wb_padded = np.pad(im1_wb, padding, mode='constant',
constant_values=0)

    im2_wb_norm = im2_wb - np.mean(im2_wb)

    im2_wb_sq = im2_wb_norm * im2_wb_norm

    sig2 = np.sum(im2_wb_sq)

    height, width = im1_wb.shape
    correlation = np.zeros((height, width))

    for i, j in np.ndindex(height, width):
        patch_wb = im1_wb_padded[i:i + im2_height, j:j + im2_width]


        patch_norm_wb = patch_wb - np.mean(patch_wb)


        patch_norm_wb_sq = patch_norm_wb ** 2

        sig1 = np.sum(patch_norm_wb_sq)

        if sig1 == 0:
            corr_values = -1
        else:
```

```
            corr_values = np.sum(patch_norm_wb * im2_wb_norm) / np.sqrt(sig1 *
sig2)

            correlation[i, j] = corr_values

    print('after_corr')
    if BOARDLESS:
        correlation=correlation[im2_height//2:height-im2_height//2,
im2_width//2:width-im2_width//2]

    local_maxima = find_local_extrema(correlation,h=im2_height,
w=im2_width,)
    flag=True
    if flag:

        rects=extract_rectangles(local_maxima,h=im2_height,
w=im2_width,img_array= im1_wb)
        print(f'after_extrem {np.count_nonzero(local_maxima)}')
        classified_array=classify_images(rects,fl)
        im1_with_detected=d_r_c(im1, h=im2_height, w=im2_width,
bool_array=classified_array)
    else:
        im1_with_detected=d_r(im1,h=im2_height, w=im2_width,
bool_array=local_maxima)


    return correlation,im1_with_detected


def find_local_extrema(arr,h,w):

    height, width = arr.shape
    padding = ((h // 2, h // 2), (w // 2, w // 2))
    arr_p= np.pad(arr, padding, mode='constant', constant_values=-1)

    # Initialize arrays to store local maxima and minima
    local_maxima = np.zeros_like(arr, dtype=bool)

    # Iterate over each pixel in the image
    for i in range(height):
        for j in range(width):
            # Get the value of the current pixel
            current_pixel = arr[i, j]
```

```
            if current_pixel>CORRELATION_COEF_THRESHOLD:
                # Get the values of the neighboring pixels
                neighbors = arr_p[i:i+h, j:j+w].flatten()

                # Check if the current pixel is a local maximum or minimum
                if np.all(current_pixel >= neighbors):
                    local_maxima[i, j] = True


    return local_maxima

def extract_rectangles(bool_array, h, w, img_array):
    rectangles = []

    rows, cols = bool_array.shape

    for i in range(rows):
        for j in range(cols):
            if bool_array[i, j]:
                top = max(0, i)
                bottom = min(rows, i + h )
                left = max(0, j)
                right = min(cols, j + w )
                rectangle = img_array[top:bottom, left:right]
                rectangles.append((rectangle,(i,j)))

    return rectangles
```

outputs.py

```
import numpy as np
import matplotlib.pyplot as plt
from PIL import ImageDraw

BOARDLESS=True

def draw_rectangles_classified(im, h, w, bool_array):
    if bool_array:
        num_colors = max([i[1] for i in bool_array])
    else:
        return
    colors = np.random.randint(0, 256, size=(num_colors, 3))
```

```
    draw = ImageDraw.Draw(im)

    # Get the dimensions of the image
    width, height = im.size
    for i in bool_array:
        y,x=i[0]
        x_min = max(0, x)
        y_min = max(0, y)
        x_max = min(width - 1, x + w )
        y_max = min(height - 1, y + h)

        # Draw the rectangle
        draw.rectangle([(x_min, y_min), (x_max, y_max)], outline='red')


    return im

def draw_rectangles(im, h, w, bool_array):


    draw = ImageDraw.Draw(im)

    # Get the dimensions of the image
    width, height = im.size
    if BOARDLESS:
        # Iterate over each pixel and check if any of the bool arrays are True
        for y in range(height-h):
            for x in range(width-w):
                if bool_array[y, x]:
                    # Calculate the coordinates for the rectangle
                    x_min = x
                    y_min = y
                    x_max = min(width - 1 + (w // 2), x+ (w // 2)*2)
                    y_max = min(height - 1 + (h // 2), y+ (h // 2)*2)

                    # Draw the rectangle
                    draw.rectangle([(x_min, y_min), (x_max, y_max)], outline="red")
    else:
        for y in range(height):
            for x in range(width):
                if bool_array[y, x]:
                    # Calculate the coordinates for the rectangle
                    x_min = max(0, x - (w // 2))
```

```
            y_min = max(0, y - (h // 2))
            x_max = min(width - 1, x + (w // 2))
            y_max = min(height - 1, y + (h // 2))

            # Draw the rectangle
            draw.rectangle([(x_min, y_min), (x_max, y_max)], outline="red")



    return im

def show_3d_plot(array_2d):
    # Get the dimensions of the array
    x_dim, y_dim = array_2d.shape

    # Create a meshgrid for the x, y coordinates
    x, y = np.meshgrid(range(x_dim), range(y_dim), indexing='ij')

    # Create a meshgrid for the x, y coordinates
    x, y = np.meshgrid(np.arange(x_dim), np.arange(y_dim), indexing='ij')

    # Create a 3D figure and axis
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')

    # Plot the 3D surface
    ax.plot_surface(x, y, array_2d, cmap='gray')

    # Set labels and title
    ax.set_xlabel('X')
    ax.set_ylabel('Y')
    ax.set_zlabel('Z')
    ax.set_title('2D Array as 3D Plot')

    # Show the plot
    plt.figure()
def hist(im):
    histogram, intensities = np.histogram(im.flatten(), bins=256, range=[0, 255])
    total_pixels = im.size
    relative_histogram = histogram / total_pixels
```

```
    # Plot the histogram
    plt.bar(intensities[:-1], relative_histogram)


    # Set the labels and title
    plt.xlabel("Intensity")
    plt.ylabel("Frequency")
    plt.title("Histogram of Intensity Frequencies")

    # Show the plot
    plt.figure()
```

classification.py

```
import numpy as np
from PIL import Image
import math
import matplotlib.pyplot as plt
PREP_LEFT=180
PREP_RIGHT=255
def preparation(im):
    l=PREP_LEFT
    r=PREP_RIGHT
    d=r-l
    def func(x):
        if x<l or x>r:
            return 0
        return int((x-l)/d*255)

    im2=np.array([[func(i) for i in j] for j in im])

    return im2
def extract_channels(image):

    arr = np.array(image)

    red_channel = np.array(arr[:, :, 0], dtype=np.float32)
    green_channel = np.array(arr[:, :, 1], dtype=np.float32)
    blue_channel = np.array(arr[:, :, 2], dtype=np.float32)

    return red_channel, green_channel, blue_channel

def classify_images(image_array,flag):
```

```
    def read_im(p):
        im=np.array(Image.open(p).convert('L'))

        return  preparation(im)

    if flag=='planes':
        type_1 = read_im('./pictures/planes/type_1.png')
        type_2 = read_im('./pictures/planes/type_2.png')
        type_3 = read_im('./pictures/planes/type_3.png')
        type_4 = read_im('./pictures/planes/type_4.png')
        not_plane_1 = read_im('./pictures/planes/not_plane_1.png')

        not_plane_2 = read_im('./pictures/planes/not_plane_2.png')

        objects = {1: type_1, 2: type_2, 3: type_3, 4: type_4,-1: not_plane_1, -2:
not_plane_2}


    def m_pq(im, p, q):
        i, j = np.indices(im.shape)
        return np.sum(im * np.power(i, q) * np.power(j, p))

    def mu_pq(im, p, q):
        x_shap = m_pq(im, 0, 1) / m_pq(im, 0, 0)
        y_shap = m_pq(im, 1, 0) / m_pq(im, 0, 0)
        i, j = np.indices(im.shape)
        return np.sum(im * np.power(i - x_shap, q) * np.power(j - y_shap, p))

    def etha_pq(im, p, q):
        mupq = mu_pq(im, p, q)
        mu00 = mu_pq(im, 0, 0) ** ((p + q) / 2 + 1)
        return mupq / mu00

    def F_1(im):
        return -np.log(etha_pq(im, 2, 0) + etha_pq(im, 0, 2))

    def F_2(im):
        return -np.log((etha_pq(im, 2, 0) - etha_pq(im, 0, 2)) ** 2 + 4 *
etha_pq(im, 1, 1) ** 2)
```

```
    def F_3(im):
        return -np.log((etha_pq(im, 3, 0) - 3*etha_pq(im, 1, 2)) ** 2 +
(3*etha_pq(im, 2, 1) - etha_pq(im, 0, 3)) ** 2)

    def apply_func(v,i):
        a=[]
        ar_of_func=[F_1,F_2,F_3]
        for j in range(i):
            a.append(ar_of_func[j](v))
        return np.array(a)
    def get_res():
        n_funcs=1
        objects_funcs = {k: apply_func(v,n_funcs) for k, v in objects.items()}
        image_funcs={v[1]: apply_func(v[0],n_funcs) for v in image_array}
        print(objects_funcs)
        print()
        print(image_funcs)
        print()
        classific = {k: find_nearest(v, objects_funcs) for k, v in
image_funcs.items()}
        print(classific)
        results=[(k,v[0]) for k,v in classific.items() if v[0]>0 ]
        return results
    def dist(f,v1):
        return math.sqrt(np.sum(np.abs(f-v1)**2))

    def find_nearest(f, funcs):
        d= np.sum(f)
        n1= -1
        d_sec=np.sum(f)
        n_sec=0
        for k, v1 in funcs.items():
            l=dist(f,v1)
            if d > l:
                n_sec=n1
                d_sec=d
                d=l
                n1 = k
            elif d_sec>l:
                n_sec=k
                d_sec=l

        if n1==4:
```

```
        print(d,d_sec)
    return [n1,d/d_sec<0.8]

return get_res()
```