ЛАБОРАТОРНА РОБОТА № 8

ДОСЛІДЖЕННЯ МЕТОДІВ КОМП'ЮТЕРНОГО ЗОРУ

Мета роботи: *використовуючи спеціалізовані бібліотеки та мову програмування Python навчитися обробляти зображення за допомогою бібліотеки OpenCV.*

1. ТЕОРЕТИЧНІ ВІДОМОСТІ

Основні теоретичні відомості подані на лекціях. Також доцільно вивчити матеріал поданий в літературі:

Додатково деякі теоретичні відомості можуть бути подані у кожному завданні окремо.

Можна використовувати Google Colab або Jupiter Notebook.

Бібліотека комп'ютерного зору OpenCV

Офіційна сторінка: http://opencv.org

Фактично найпопулярніша бібліотека комп'ютерного зору.

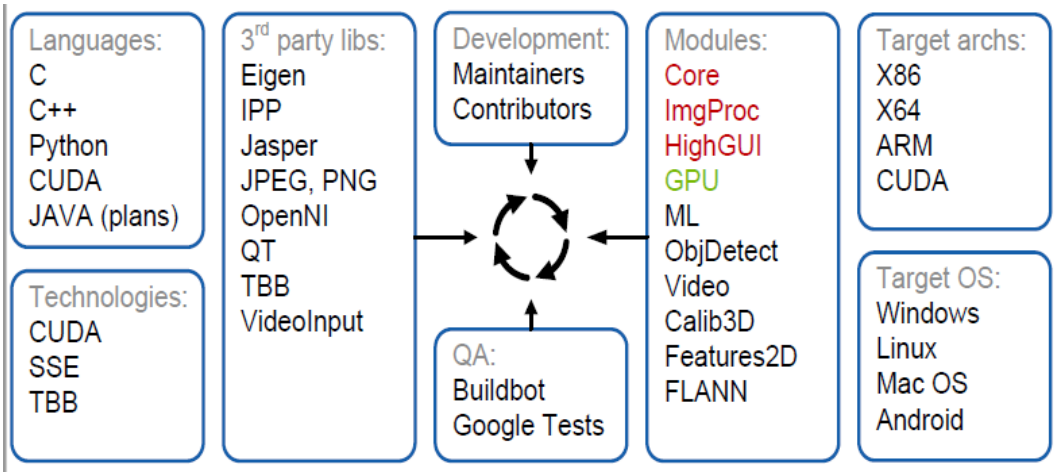
•Написана на C/C++, вихідний код відкритий, містить понад 1000 функцій та алгоритмів.

•Ліцензія BSD (дозволяється безкоштовне використання вдома, для навчання, на роботі)

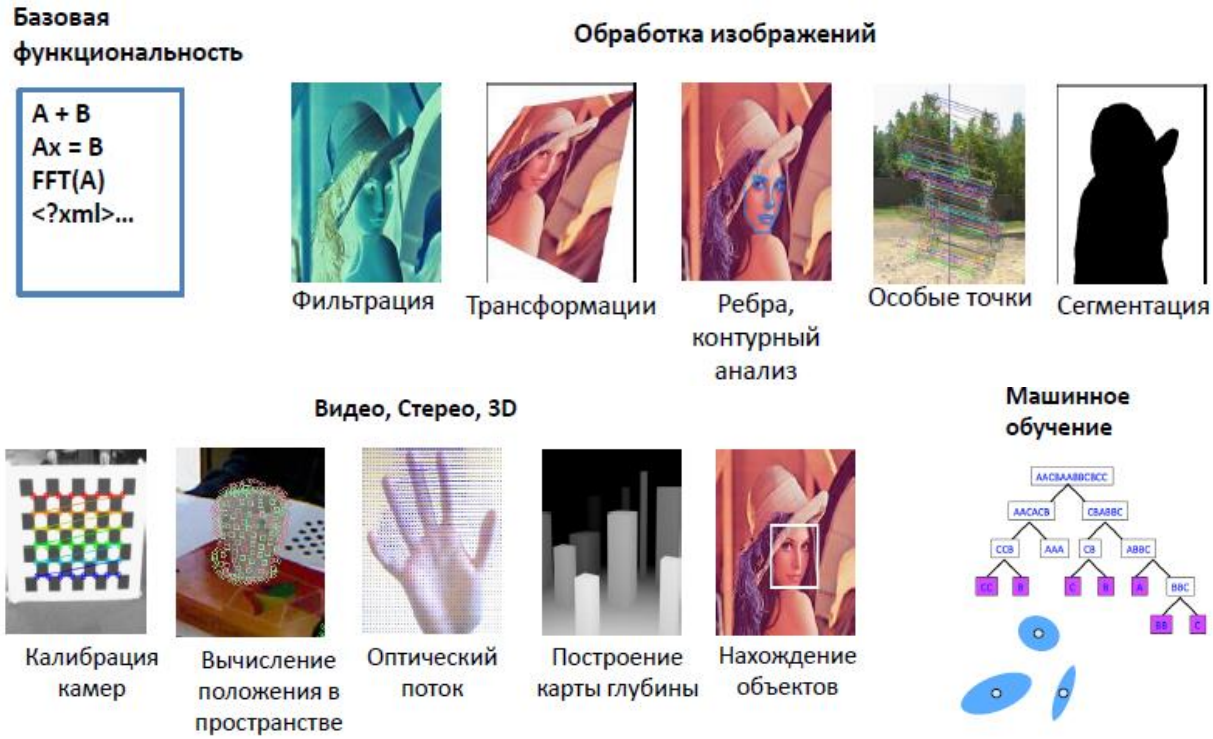
• Розробляється з 1998 року, спочатку в Інтел, тепер у компанії Itseez за активної участі спільноти.

•Використовується багатьма компаніями, організаціями, ВНЗ, наприклад NVidia, Willow Garage, Intel, Google, Stanford …

Архітектура та розробка OpenCV



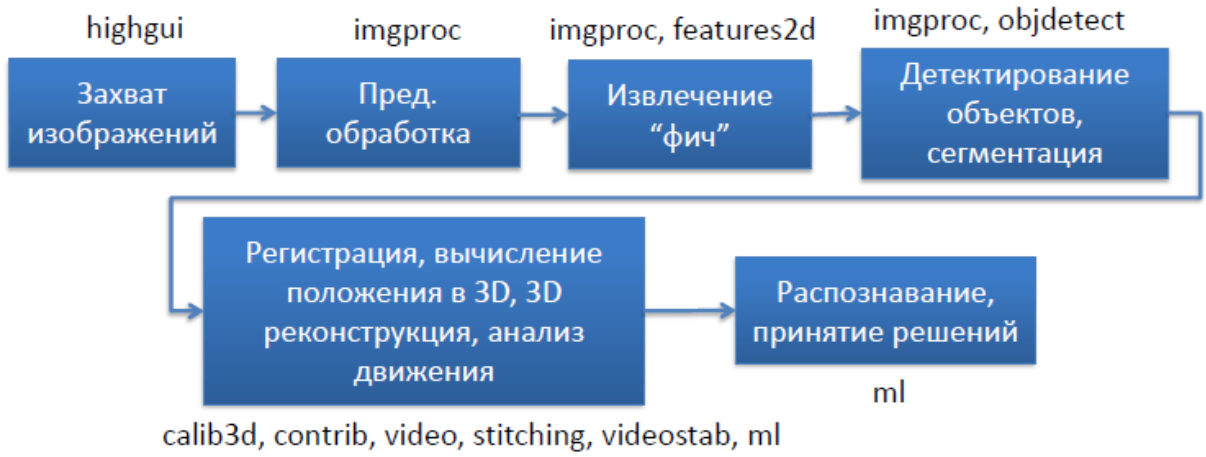
Функціональність бібліотеки



OpenCV у додатках комп'ютерного зору

OpenCV - базова, в цілому низькорівнева бібліотека. Вона надає будівельні блоки для додатків. Самі додатки розробляють користувачі.

Загальна схема типового додатка CV



2. ЗАВДАННЯ НА ЛАБОРАТОРНУ РОБОТУ ТА МЕТОДИЧНІ РЕКОМЕНДАЦІЇ ДО ЙОГО ВИКОНАННЯ

Завдання 2.1. Завантаження зображень та відео в OpenCV

З офіційного сайту імпортуйте та встановіть бібліотеку OpenCV

Прочитайте та виконайте дії згідно рекомендацій до виконання.

РЕКОМЕНДАЦІЇ ДО ВИКОНАННЯ

Підключіть вашу веб-камеру та виведіть зображення:

```
import cv2
frameWidth = 640
frameHeight = 480
cap = cv2.VideoCapture(0)
cap.set(3, frameWidth)
cap.set(4, frameHeight)
cap.set(10,150)
while True:
    success, img = cap.read()
    cv2.imshow("Result", img)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
```

За допомогою інструмента Ножиці робіть вирізку ВАШОГО зображення із вікна відображення вашої веб-камери та збережіть його під назвою xxxx.jpg, де xxxx – ваше прізвище англійськими літерами.

Закрийте вікно веб-камери натисканням клавіші «q».

На зображенні повинно бути ваше обличчя, так щоб воно займало приблизно 5-6 частину всього зображення і вас можна було б ідентифікувати.

У разі якщо за якихось причин веб-камера у вас не працює, то візьміть своє електронне фото на документи.

Перенесіть ваше зображення у ваш проект, щоб не прописувати шлях до нього.

Завантажте ваше зображення

```
import cv2
# LOAD AN IMAGE USING 'IMREAD'
img = cv2.imread("xxxx.jpg")
# DISPLAY
cv2.imshow("xxxx ",img)
cv2.waitKey(0)
```

Отримане зображення занесіть у звіт!

Збережіть код робочої програми під назвою LR_8_task_1.py

Зробіть висновок

Завдання 2.2. Дослідження перетворень зображення

Для вашого зображення отриманого у попередньому завданні дослідіть як впливають на зображення методи: cvtColor, GaussianBlur, Canny, dilate, erode.

РЕКОМЕНДАЦІЇ ДО ВИКОНАННЯ

Створіть новий файл Python та виконайте такі дії.

```
import cv2
import numpy as np

img = cv2.imread("xxxx.jpg")
kernel = np.ones((5,5),np.uint8)

imgGray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
imgBlur = cv2.GaussianBlur(imgGray,(7,7),0)
imgCanny = cv2.Canny(img,150,200)
imgDialation = cv2.dilate(imgCanny,kernel,iterations=1)
imgEroded = cv2.erode(imgDialation,kernel,iterations=1)

cv2.imshow("Gray Image",imgGray)
cv2.imshow("Blur Image",imgBlur)
cv2.imshow("Canny Image",imgCanny)
cv2.imshow("Dialation Image",imgDialation)
cv2.imshow("Eroded Image",imgEroded)
cv2.waitKey(0)
```

Отримані зображення занесіть у звіт.

Проаналізуйте отримані зображення. Зробіть висновки. У висновках дайте відповіді на такі питання:

Для чого застосовується метод cvtColor, та що ми отримали у результаті його застосування?

Для чого застосовується метод GaussianBlur, та що ми отримали у результаті його застосування?

Для чого застосовується метод Canny, та що ми отримали у результаті його застосування?

Для чого застосовується метод dilate, та що ми отримали у результаті його застосування?

Для чого застосовується метод erode, та що ми отримали у результаті його застосування?

Збережіть код робочої програми з обов'язковими коментарям під назвою LR_8_task_2.py

Завдання 2.3. Вирізання частини зображення

По аналогії з прикладом наведеним у рекомендаціях розробіть програмку та виріжте із вашого зображення лише ваше обличчя.

РЕКОМЕНДАЦІЇ ДО ВИКОНАННЯ

Створіть новий файл Python та виріжте частину зображення.

```
import cv2
import numpy as np

img = cv2.imread("shapes.png")
print(img.shape)

imgResize = cv2.resize(img,(1000,500))
print(imgResize.shape)

imgCropped = img[46:119,352:495]

cv2.imshow("Image",img)
#cv2.imshow("Image Resize",imgResize)
cv2.imshow("Image Cropped",imgCropped)

cv2.waitKey(0)
```

Код програми та вирізане зображення занесіть у звіт.

Програмний код збережіть під назвою LR_8_task_3.py

Завдання 2.4. Розпізнавання обличчя на зображенні

По аналогії з кодом, що наданий у рекомендаціях, виконайте розпізнавання обличчя на вашому зображенні.

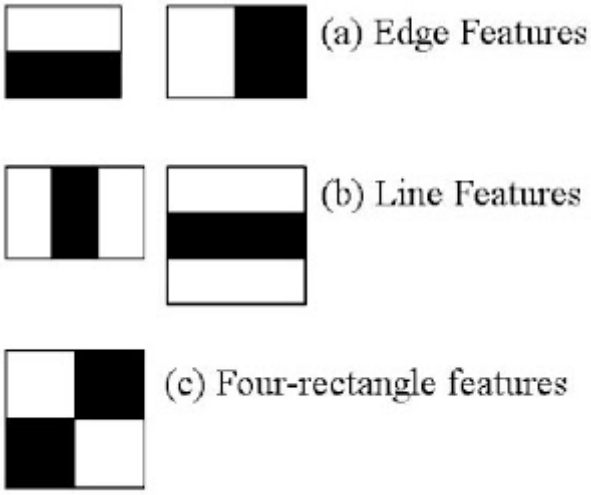
РЕКОМЕНДАЦІЇ ДО ВИКОНАННЯ

Теорія

Виявлення об’єктів за допомогою каскадних класифікаторів на основі функцій Хаара є ефективним методом виявлення об’єктів, запропонованим Полом Віолою та Майклом Джонсом у їхній статті «Швидке виявлення об’єктів за допомогою посиленого каскаду простих функцій» у 2001 році. Це підхід, заснований на машинному навчанні, де каскадна функція тренується на великій кількості позитивних і негативних образів. Потім він використовується для виявлення об’єктів на інших зображеннях.

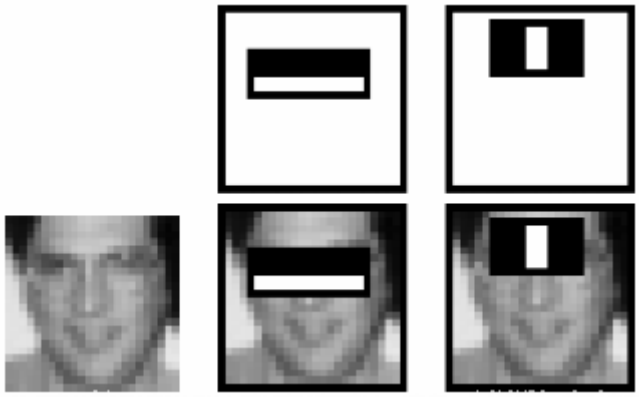
Тут ми будемо працювати з розпізнаванням обличчя. Спочатку алгоритм потребує багато позитивних зображень (зображення облич) і негативних зображень (зображення без облич), щоб навчити класифікатор. Потім нам

потрібно витягти з нього ознаки (функції). Для цього використовуються функції Хаара, показані на зображенні нижче. Вони подібні до нашого згорткового ядра. Кожна функція є окремим значенням, отриманим шляхом віднімання суми пікселів під білим прямокутником із суми пікселів під чорним прямокутником.



Тепер усі можливі розміри та розташування кожного ядра використовуються для обчислення багатьох функцій. (Тільки уявіть, скільки обчислень для цього потрібно? Навіть вікно 24x24 призводить до понад 160 000 функцій). Для кожного обчислення функції нам потрібно знайти суму пікселів під білим і чорним прямокутниками. Щоб вирішити цю проблему, вони ввели цілісне зображення. Яким би великим не було ваше зображення, воно зводить обчислення для певного пікселя до операції, що включає лише чотири пікселі. Гарно, чи не так? Це робить роботу надзвичайно швидкою.

Але серед усіх цих функцій, які ми підрахували, більшість із них несуттєві. Для прикладу розглянемо зображення нижче. Верхній рядок показує дві хороші характеристики. Перша вибрана ознака, здається, зосереджена на тому, що область очей часто темніша за область носа та щік. Друга вибрана ознака базується на тому, що очі темніші за перенісся. Але ті ж вікна, нанесені на щоки або будь-яке інше місце, не мають значення. Отже, як вибрати найкращі функції з 160 000+ функцій? Це досягається Adaboost.



Для цього ми застосовуємо кожну функцію на всіх навчальних зображеннях. Для кожної функції він знаходить найкращий поріг, який класифікуватиме обличчя на позитивні та негативні. Очевидно, будуть помилки або неправильна класифікація. Ми вибираємо ознаки з мінімальною частотою помилок, що означає, що вони найточніше класифікують зображення обличчя та зображення без нього. (Цей процес не такий простий. Спочатку кожному зображенню надається однакова вага. Після кожної класифікації ваги неправильно класифікованих зображень збільшуються. Потім виконується той самий процес. Обчислюються нові частоти помилок. Також нові ваги. процес триває, доки не буде досягнуто необхідної точності чи частоти помилок або не знайдено необхідну кількість функцій).

Остаточний класифікатор є зваженою сумою цих слабких класифікаторів. Він називається слабким, оскільки сам по собі не може класифікувати зображення, але разом з іншими утворює сильний класифікатор. У документі йдеться, що навіть 200 функцій забезпечують виявлення з точністю 95%. Їхнє остаточне налаштування мало близько 6000 функцій. (Уявіть собі скорочення з 160 000+ функцій до 6000 функцій. Це великий приріст).

Отже, тепер ви зробите зображення. Візьміть кожне вікно 24x24. Застосуйте до нього 6000 функцій. Перевірте, обличчя це чи ні. Вау.. Хіба це не трохи неефективно та забирає багато часу? Так. Для цього автори знайшли хороше рішення.

На зображенні більша частина зображення не є областю обличчя. Тому краще мати простий спосіб перевірити, чи вікно не є областю обличчя. Якщо це не так, викиньте його за один раз і не обробляйте знову. Натомість зосередьтеся на регіонах, де може бути обличчя. Таким чином ми витрачаємо більше часу на перевірку можливих областей обличчя.

Для цього вони представили концепцію каскаду класифікаторів. Замість застосування всіх 6000 функцій у вікні, функції групуються в різні етапи класифікаторів і застосовуються одна за одною. (Зазвичай перші кілька етапів містять набагато менше функцій). Якщо вікно не проходить перший етап, відкиньте його. Ми не розглядаємо інші функції на ньому. Якщо він пройде, застосуйте другий етап функцій і продовжіть процес. Вікно, яке проходить усі етапи, є лицьовою областю.

Детектор авторів мав понад 6000 функцій із 38 етапами з 1, 10, 25, 25 та 50 ознаками на перших п’яти етапах. (Дві функції на зображенні вище насправді отримані як дві найкращі функції від Adaboost). За словами авторів, в середньому на підвікні оцінюються 10 функцій з 6000+.

Це просте інтуїтивне пояснення того, як працює розпізнавання обличчя за методом Віоли-Джонса.

OpenCV надає метод для м (Cascade Classifier Training) або попередньо навчені моделі, які можна читати за допомогою методу cv::CascadeClassifier::load. Попередньо підготовлені моделі знаходяться в папці даних у встановленому OpenCV або їх можна знайти окремо.

У наступному прикладі коду використовуватимуться попередньо підготовлені каскадні моделі Хаара для виявлення облич і очей на зображенні. Спочатку створюється cv::CascadeClassifier і завантажується необхідний файл XML за допомогою методу cv::CascadeClassifier::load. Після цього виявлення виконується за допомогою методу cv::CascadeClassifier::detectMultiScale, який повертає прямокутники меж для виявлених облич.

Створіть новий файл Python та виконайте дії для свого зображення.

```
import cv2

faceCascade=
cv2.CascadeClassifier("Resources/haarcascade_frontalface_default.xml")
img = cv2.imread('XXXX.jpg')
imgGray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)

faces = faceCascade.detectMultiScale(imgGray,1.1,4)

for (x,y,w,h) in faces:
    cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)


cv2.imshow("Result", img)
cv2.waitKey(0)
```

Збережіть код робочої програми з обов'язковими коментарям під назвою LR_8_task_4.py

Код програми та зображення з виявленим обличчям занесіть у звіт. Зробіть висновок. Висновок занесіть у звіт.

Завдання 2.5. Розпізнавання об'єктів на зображенні за допомогою методів зіставлення шаблонів (Template Matching)

По аналогії з кодом, що наданий у рекомендаціях, виконайте розпізнавання об'єктів на зображенні за допомогою методів зіставлення шаблонів.

РЕКОМЕНДАЦІЇ ДО ВИКОНАННЯ

Теорія

Зіставлення шаблонів (Template Matching) — це метод пошуку та знаходження розташування зображення шаблону на більшому зображенні. OpenCV постачається з функцією cv.matchTemplate() для цієї мети. Він просто накладає зображення шаблону на вхідне зображення (як у 2D згортці) і порівнює шаблон і фрагмент вхідного зображення зі зображенням шаблону.

У OpenCV реалізовано кілька методів порівняння. (Детальнішу інформацію можна переглянути в документах). Він повертає зображення у відтінках сірого, де кожен піксель означає, наскільки околиці цього пікселя відповідають шаблону.

Якщо вхідне зображення має розмір *ширина* x *висота* (W x H), а зображення шаблону — розмір (w x h), вихідне зображення матиме розмір (W-w+1, H-h+1). Отримавши результат, ви можете використовувати функцію cv.minMaxLoc(), щоб знайти максимальне/мінімальне значення. Візьміть його за верхній лівий кут прямокутника, а (w, h) за ширину та висоту прямокутника. Цей прямокутник є вашою областю шаблону.

Примітка: Якщо ви використовуєте cv.TM_SQDIFF як метод порівняння, мінімальне значення дає найкращий збіг.

Спочатку, ми будемо шукати обличчя Мессі на його фотографії. Тому із зображення messi_full.JPG було створено шаблон , messi_face.JPG:

Введіть код та спробуйте всі методи порівняння, щоб побачити, як виглядають їхні результати для цих зображень:

```
import cv2 as cv
import numpy as np
from matplotlib import pyplot as plt

img = cv.imread('messi_full.JPG',0)
img2 = img.copy()
template = cv.imread('messi_face.JPG',0)

w, h = template.shape[::-1]
# All the 6 methods for comparison in a list
methods = ['cv.TM_CCOEFF', 'cv.TM_CCOEFF_NORMED', 'cv.TM_CCORR',
            'cv.TM_CCORR_NORMED', 'cv.TM_SQDIFF', 'cv.TM_SQDIFF_NORMED']
for meth in methods:
    img = img2.copy()
    method = eval(meth)
    # Apply template Matching
    res = cv.matchTemplate(img,template,method)
    min_val, max_val, min_loc, max_loc = cv.minMaxLoc(res)
    # If the method is TM_SQDIFF or TM_SQDIFF_NORMED, take minimum
    if method in [cv.TM_SQDIFF, cv.TM_SQDIFF_NORMED]:
        top_left = min_loc
    else:
        top_left = max_loc
    bottom_right = (top_left[0] + w, top_left[1] + h)

    cv.rectangle(img,top_left, bottom_right, 255, 2)

    plt.subplot(121),plt.imshow(res,cmap = 'gray')
    plt.title('Matching Result'), plt.xticks([]), plt.yticks([])
    plt.subplot(122),plt.imshow(img,cmap = 'gray')
    plt.title('Detected Point'), plt.xticks([]), plt.yticks([])
    plt.suptitle(meth)

    plt.show()
```

Перегляньте отримані результати.

cv.TM_CCOEFF
cv.TM_CCOEFF_NORMED
cv.TM_CCORR
cv.TM_CCORR_NORMED
cv.TM_SQDIFF
cv.TM_SQDIFF_NORMED

Ви побачите, що результат використання cv.TM_CCORR не такий добрий, як ми очікували.

Відкрийте ваше зображення xxxx.jpg та за допомогою інструмента Ножиці виріжте своє обличчя. Збережіть його в окремий файл xxxx_face.jpg. (Можна скористатися результатами завдання 2.3)
Проведіть досліди для своїх файлів. Занесіть у звіт ваше зображення та шаблон та результати пошуку (6x2 зображень).

Збережіть код робочої програми з обов'язковими коментарям під назвою LR_8_task_5.py
Код програми та зображення з виявленим обличчям занесіть у звіт. Зробіть висновок. Висновок занесіть у звіт.
У висновках поясніть що означає кожен метод (cv.TM_CCOEFF, cv.TM_CCOEFF_NORMED, cv.TM_CCORR, cv.TM_CCORR_NORMED, cv.TM_SQDIFF, cv.TM_SQDIFF_NORMED) та який метод ви вважаєте найкращим для вашої задачі.

Завдання 2.6. Сегментація зображення алгоритмом водорозподілу

За наведеними рекомендаціями вивчіть сегментацію зображення алгоритмом водо розподілу.

РЕКОМЕНДАЦІЇ ДО ВИКОНАННЯ

Теорія

Будь-яке зображення у відтінках сірого можна розглядати як топографічну поверхню, де висока інтенсивність позначає вершини та пагорби, а низька інтенсивність позначає долини. Ви починаєте заповнювати кожну ізольовану долину (місцеві мінімуми) різнокольоровою водою (мітки). Коли вода піднімається, залежно від піків (градієнтів) поблизу, вода з різних долин, очевидно, різного кольору, почне зливатися. Щоб уникнути цього, ви будуєте бар'єри в місцях злиття води. Ви продовжуєте роботу по наповненню водою та будівництву перешкод, доки всі вершини не опиниться під водою.

Тоді бар’єри, які ви створили, дають вам результат сегментації. Це «філософія» вододілу.

Тобто, конвеєр алгоритму водо розподілу можна подати як:

1. Створення бінарного зображення
2. Обчислення відстані перетворення.
3. Знаходження точок локальних максимумів.
4. Маркування позначок

Ви можете відвідати веб-сторінку, щоб зрозуміти це за допомогою деяких анімацій https://people.cmm.minesparis.psl.eu/users/beucher/wtshed.html

Але цей підхід дає результат із надмірною сегментацією через шум або будь-які інші нерівності в зображенні. Отже, OpenCV реалізував алгоритм вододілу на основі маркерів, де ви вказуєте, які всі точки долини потрібно об’єднати, а які ні. Це інтерактивна сегментація зображення. Що ми робимо, це даємо різні мітки для нашого об’єкта, який ми знаємо. Позначте область, яка, як ми впевнені, є переднім планом або об’єктом, одним кольором (або інтенсивністю), позначте область, яка, як ми впевнені, є фоном або необ’єктом, іншим кольором і, нарешті, область, у якій ми ні в чому не впевнені, позначте його 0. Це наш маркер. Потім застосуйте алгоритм вододілу. Тоді наш маркер буде оновлено наданими мітками, а межі об’єктів матимуть значення -1.

Нижче ми побачимо приклад того, як використовувати перетворення відстані разом із вододілом для сегментації об’єктів, що дотикаються один до одного.

Розгляньте зображення монет нижче, де монети торкаються одна одної.

Створіть новий файл та завантажте зображення coins.jpg .

```
import numpy as np
import cv2
from matplotlib import pyplot as plt

img = cv2.imread('coins.jpg')
cv2.imshow("coins",img)
cv2.waitKey(0)
```

Отримане зображення занесіть у звіт.

Почнемо з пошуку приблизної оцінки монет. Для цього ми можемо використати бінаризацію.

```
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
ret, thresh = cv2.threshold(gray,0,255,cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU)
cv2.imshow("coins bin ",thresh)
cv2.waitKey(0)
```

Отримане зображення занесіть у звіт.

Тепер нам потрібно видалити всі маленькі шуми на зображенні. Для цього ми можемо використати фільтр морфологічне відкриття. Щоб видалити будь-які маленькі отвори в об'єкті, ми можемо використати морфологічне закриття. Отже, тепер ми точно знаємо, що область, розташована ближче до центру об'єктів, є переднім планом, а область, розташована набагато далі від об'єкта, є фоном. Єдиний регіон, який ми не впевнені, це граничний регіон монет.

Отже, нам потрібно виділити область, яка, як ми впевнені, є монетами. Ерозія видаляє граничні пікселі. Отже, що б не залишилося, ми можемо бути впевнені, що це монети. Це спрацювало б, якби предмети не торкалися один одного. Але оскільки вони торкаються один одного, іншим хорошим варіантом було б знайти перетворення відстані та застосувати відповідний поріг. Далі нам потрібно знайти область, яка, як ми впевнені, не є монетами. Для цього розширюємо результат. Розширення збільшує межу об'єкта до фону. Таким чином ми можемо переконатися, що будь-яка область фону в результаті дійсно є фоном, оскільки прикордонна область видаляється. Дивіться зображення нижче.

Решта регіонів - це ті, про які ми не маємо жодного уявлення, будь то монети чи фон. Алгоритм вододілу повинен знайти його. Зазвичай ці області розташовані навколо кордонів монет, де зустрічаються передній план і фон (або навіть дві різні монети). Ми називаємо це кордоном. Його можна отримати шляхом віднімання площі sure_fg від площі sure_bg.

```
# видалення шуму
kernel = np.ones((3,3),np.uint8)
opening = cv2.morphologyEx(thresh,cv2.MORPH_OPEN,kernel, iterations = 2)
# певна фонова область
sure_bg = cv2.dilate(opening,kernel,iterations=3)
# Пошук впевненої області переднього плану
dist_transform = cv2.distanceTransform(opening,cv2.DIST_L2,5)
ret, sure_fg = cv2.threshold(dist_transform,0.7*dist_transform.max(),255,0)
# Пошук невідомого регіону
sure_fg = np.uint8(sure_fg)
unknown = cv2.subtract(sure_bg,sure_fg)
cv2.imshow("coins ",opening)
cv2.waitKey(0)
```

Отримане зображення занесіть у звіт.

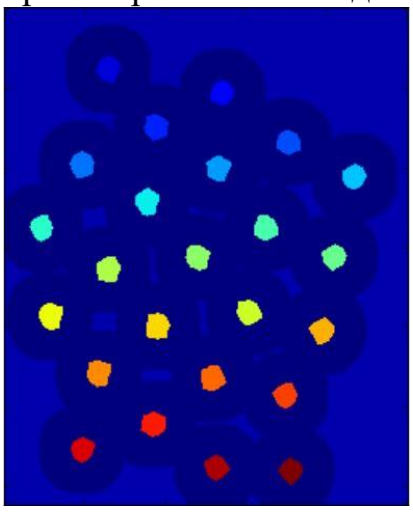
Подивіться результат. На пороговому зображенні ми отримуємо кілька областей монет, які, як ми впевнені, є монетами, і зараз вони від'єднані. (У деяких випадках вас може зацікавити лише сегментація переднього плану, а не розділення об'єктів, які торкаються один одного. У такому випадку вам не потрібно використовувати перетворення відстані, достатньо просто розмивання. Розмивання — це лише ще один метод виділення певної області переднього плану.)

Тепер ми точно знаємо, які регіони монет, які є фоном і все інше. Отже, ми створюємо маркер (це масив такого ж розміру, як і оригінальне зображення, але з типом даних int32) і позначаємо області всередині нього. Області, які ми знаємо напевно (передній план чи фон), позначаються будь-якими додатними цілими числами, але різними цілими числами, а область, яку ми не знаємо напевно, просто залишаємо нулем. Для цього ми використовуємо cv.connectedComponents(). Він позначає фон зображення 0, потім інші об'єкти позначаються цілими числами, починаючи з 1.

Але ми знаємо, що якщо фон позначено 0, вододіл вважатиме його невідомою областю. Тому ми хочемо позначити його іншим цілим числом. Замість цього ми позначимо невідомий регіон, визначений невідомим, 0.

```
# Маркування міток
ret, markers = cv2.connectedComponents(sure_fg)
# Додайте один до всіх міток, щоб впевнений фон був не 0, а 1
markers = markers+1
# Тепер позначте область невідомого нулем
markers[unknown==255] = 0
```

Перегляньте результат, показаний у кольоровій карті JET. Темно-синя область показує невідому область. Безумовно, монети мають різні номінали. Решта області, яка є впевненим фоном, показана більш світло-блакитним кольором порівняно з невідомою областю.



Тепер наш маркер готовий. Настав час останнього кроку, застосувати вододіл. Тоді зображення маркера буде змінено. Гранична область буде позначена -1.

```
markers = cv2.watershed(img,markers)
img[markers == -1] = [255,0,0]

cv2.imshow("coins_markers",img)
cv2.waitKey(0)
```

Проаналізуйте результат. Для деяких монет область, де вони торкаються, правильно сегментована, а для деяких ні.

Отримане зображення занесіть у звіт.

Збережіть код робочої програми з обов'язковими коментарям під назвою LR_8_task_6.py
Код програми та зображення з виявленим обличчям занесіть у звіт.
Зробіть висновок. Висновок занесіть у звіт.

Додаткове завдання
2.7. Сегментація зображення

Проведіть дослідження та сегментуйте монети на зображенні coins_2.JPG таким чином, щоб монети однієї вартості виділялися одним кольором.

РЕКОМЕНДАЦІЇ ДО ВИКОНАННЯ

Збережіть код робочої програми з обов'язковими коментарям під назвою LR_8_task_7.py
Код програми, початкове зображення та зображення з виявленими сегментами занесіть у звіт.
Зробіть висновок. Висновок занесіть у звіт.

Коди комітити на GitHab. У кожному звіті повинно бути посилання на GitHab.

Назвіть бланк звіту СШІ-ЛР-8-NNN-XXXXX.doc
де NNN – позначення групи
XXXXX – позначення прізвища студента.

Переконвертуйте файл звіту в СШІ-ЛР-8-NNN-XXXXX.pdf
Надішліть чи представте звіт викладачу.