

Міністерство освіти і науки України
Одеський національний університет імені І. І. Мечникова
кафедра загальної фізики і фізики теплоенергетичних та хімічних процесів

Методичні вказівки до лабораторних робіт з курсу

«Комп'ютерні методи обробки зображень»

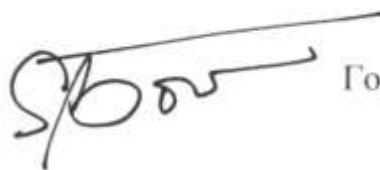
Одеса 2020

Автор та укладач:

Доцент Дараків Денис Сергійович

Розглянуто та рекомендовано на засіданні кафедри загальної фізики і фізики теплоенергетичних та хімічних процесів протокол № 2 від 28.09.2020р.

Завідуючий кафедри загальної фізики і фізики теплоенергетичних та хімічних процесів



Гоцувський В.Я.

Автор та укладач:

Доцент Дараків Денис Сергійович

Розглянуто та рекомендовано на засіданні кафедри загальної фізики і фізики теплоенергетичних та хімічних процесів протокол № 2 від 28.09.2020р.

Завідуючий кафедри загальної фізики і фізики теплоенергетичних та хімічних процесів

Гоцувський В.Я.

ВСТУП

Програма навчальної дисципліни «Комп'ютерна обробка зображень» складена відповідно до освітньо-наукової програми підготовки першого (освітньо-наукового) рівня вищої освіти (бакалавр). Галузь знань: 10 – «Природничі науки». Спеціальність: 105 - Прикладна фізика та наноматеріали. Освітньо-наукова програма: “ Прикладна фізика та наноматеріали ”.

Метою викладання навчальної дисципліни є надати майбутнім бакалаврам необхідного мінімуму попередніх відомостей щодо методів та алгоритмів комп'ютерної обробки зображень, засвоєння фундаментальних фізичних складових та отримання практичних навичок, що здобуваються в межах дисципліни «Комп'ютерна обробка зображень». Це є умовою для подальшого засвоєння дисциплін за вибором з циклу професійної підготовки, успішного виконання теоретичної та експериментальної наукової роботи.

Основними завданнями вивчення дисципліни є засвоєння студентами основ методів та алгоритмів обробки зображень та успішне використання їх на практиці. Ця розробка є описом лабораторних робіт, які включені до складу курсу.

Лабораторна робота 1.

Обробка зображень із використанням мови програмування Python.

Огляд необхідних модулів: SciPy, OpenCv, Matplotlib, SciKit-Image.

OpenCV

OpenCV був започаткований в Intel в 1999 році Гері Бредським, а перший реліз вийшов у 2000 році. Вадим Писаревський приєднався до Гері Брадського, щоб керувати російською командою програмного забезпечення IntelOpenCV. У 2005 році OpenCV був використаний на Stanley, транспортному засобі, який виграв 2005 DARPA GrandChallenge. Пізніше його активний розвиток продовжувався за підтримки WillowGarage, а керівником проекту були Гері Брадський та Вадим Писаревський. Зараз OpenCV підтримує багато алгоритмів, пов'язаних із комп'ютерним зором та машинним навчанням, і він розширюється з кожним днем.

В даний час OpenCV підтримує широкий спектр мов програмування, таких як C ++, Python, Java тощо, і доступний на різних платформах, включаючи Linux, Windows, OS X, Android, iOS тощо. Крім того, інтерфейси на основі CUDA та OpenCL також активно розробляються для високих - швидкісні операції графічного процесора.

OpenCV-Python - це Python API OpenCV. Він поєднує в собі найкращі якості OpenCV C ++ API та мови Python.

OpenCV-Python

Python - це мова програмування загального призначення, започаткована ГвідованРоссумом, яка за короткий час стала дуже популярною в основному завдяки своїй простоті та читабельності коду. Це дозволяє програмісту

висловлювати свої ідеї меншою кількістю рядків коду, не зменшуючи при цьому читабельності.

У порівнянні з іншими мовами, такими як C / C ++, Python працює повільніше. Але ще однією важливою особливістю Python є те, що її можна легко розширити за допомогою C / C ++. Ця функція допомагає нам писати обчислювально обчислювальні коди на C / C ++ і створювати для неї обгортку Python, щоб ми могли використовувати ці обгортки як модулі Python. Це дає нам дві переваги: по-перше, наш код такий же швидкий, як оригінальний код C / C ++ (оскільки це власне код C ++, що працює у фоновому режимі), а по-друге, дуже легко кодувати на Python. Ось як працює OpenCV-Python, це обгортка Python навколо оригінальної реалізації C ++.

А підтримка NumPy полегшує завдання. NumPy - це високооптимізована бібліотека для числових операцій. Він надає синтаксис у стилі MATLAB. Усі структури масивів OpenCV перетворюються в масиви NumPy та з них. Тож, які б операції ви не робили в NumPy, ви можете поєднувати їх з OpenCV, що збільшує кількість зброї у вашому арсеналі. Крім того, кілька інших бібліотек, таких як SciPy, Matplotlib, які підтримують NumPy, можуть бути використані з цим.

Отже, OpenCV-Python є відповідним інструментом для швидкого прототипування проблем із комп'ютерним зором.

Початок роботи із зображеннями

Цілі:

- Тут ви дізнаєтесь, як читати зображення, як його відобразити та як зберегти назад.
- Ви вивчите ці функції: `cv2.imread()`, `cv2.imshow()`, `cv2.imwrite()`

- За бажанням ви дізнаєтесь, як відобразити зображення за допомогою Matplotlib

Використання OpenCV

Зчитування зображення

Використовуйте функцію `cv2.imread()` для читання зображення. Зображення повинно бути в робочому каталозі або вказати повний шлях до зображення.

Другий аргумент - це прапор (`flag`), який визначає спосіб читання зображення.

`cv2.IMREAD_COLOR`: завантажує кольорове зображення. Будь-яка прозорість зображення буде нехтуватися. Це прапор за замовчуванням.

`cv2.IMREAD_GRAYSCALE`: завантажує зображення в режимі відтінків сірого.

`cv2.IMREAD_UNCHANGED`: Завантажує зображення як таке, включаючи альфа-канал.

Подивіться на код нижче:

```
import numpy as np
import cv2

# Load an color image in grayscale
img = cv2.imread('messi5.jpg', 0)
```

Відображення зображення

Використовуйте функцію `cv2.imshow()` для відображення зображення у вікні. Вікно автоматично підходить до розміру зображення.

Перший аргумент - це ім'я вікна, яке належить до типу даних `string`. Другий аргумент - це наше зображення. Ви можете створити скільки завгодно вікон, але з різними назвами.

```
cv2.imshow('image',img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Знімок екрана вікна буде виглядати на зразок цього:



`cv2.waitKey()` - це функція прив'язки клавіатури. Його аргументом є час у мілісекундах. Функція чекає вказаних мілісекунд для будь-якої події клавіатури. Якщо ви натиснете будь-яку клавішу протягом цього часу, програма продовжиться. Якщо передано 0, він нескінченно чекає натискання клавіші. Він також може бути встановлений для виявлення конкретних натискань клавіш, наприклад, якщо натиснута клавіша а тощо, про що ми поговоримо нижче.

`cv2.destroyAllWindows()` просто знищує всі створені нами вікна. Якщо ви хочете знищити якийсь конкретне вікно, використовуйте функцію `cv2.destroyWindow()`, де ви передаєте точну назву вікна як аргумент.

```
cv2.namedWindow('image', cv2.WINDOW_NORMAL)
cv2.imshow('image',img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Запис зображення

Використовуйте функцію `cv2.imwrite()`, щоб зберегти зображення. Перший аргумент - це назва файлу, другий аргумент - зображення, яке потрібно зберегти.

```
cv2.imwrite('messigray.png',img)
```

Це збереже зображення у форматі PNG у робочому каталозі.

Підсумок

Нижче програма завантажує зображення у відтінках сірого, відображає його, зберігає зображення, якщо натискаєте «s» і виходить, або просто виходить без збереження, якщо натискаєш клавішу ESC.

```
import numpy as np
import cv2

img = cv2.imread('messi5.jpg',0)
```



```
cv2.imshow('image',img)
k = cv2.waitKey(0)
if k == 27:          # waitfor ESC keytoexit
    cv2.destroyAllWindows()
elif k == ord('s'): # waitfor's'keytosaveandexit
    cv2.imwrite('messigray.png',img)
    cv2.destroyAllWindows()
```

Якщо ви використовуєте 64-розрядну машину, вам доведеться змінити рядок `k = cv2.waitKey(0)` таким чином: `k = cv2.waitKey(0) & 0xFF`.

Використання Matplotlib

Matplotlib - це бібліотека для побудови графіків для Python, яка надає широкий спектр методів побудови графіків. Ви побачите їх у наступних статтях. Тут ви дізнаєтесь, як відображати зображення за допомогою Matplotlib. Ви можете збільшувати зображення, зберігати їх тощо за допомогою Matplotlib.

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

img = cv2.imread('messi5.jpg', 0)
plt.imshow(img, cmap='gray', interpolation='bicubic')
plt.xticks([], plt.yticks([])) # to hide tick values on X and Y axis
plt.show()
```

Знімок екрана вікна буде виглядати на зразок цього:



Кольорове зображення, завантажене OpenCV, знаходиться в режимі BGR. Але Matplotlib відображається в режимі RGB. Отже, кольорові зображення не відобразатимуться належним чином у Matplotlib, якщо зображення читатиметься за допомогою OpenCV.

Завдання:

На прикладі будь-якого зображення, виконати вищеописані операції зчитування, відображення та запису зображення. Для відображення використовуйте як модуль OpenCV, так і Matplotlib.

Лабораторна робота 2.

Основи обробки зображень. Позитивне та негативне зображення.

Зображення також є набір пікселів. Коли ми зберігаємо зображення в комп'ютерах або в цифровій формі, зберігаються відповідні значення пікселів. Отже, коли ми читаємо зображення у змінну за допомогою OpenCV у Python, змінна зберігає значення пікселів зображення. Коли ми намагаємось негативно перетворити зображення, найсвітліші ділянки перетворюються на найтемніші, а найтемніші - на найсвітліші.

Як ми знаємо, кольорове зображення зберігає 3 різні канали. Вони бувають червоними, зеленими та синіми. Ось чому кольорові зображення також відомі як RGB-зображення. Отже, якщо нам потрібне негативне перетворення зображення, тоді нам потрібно інвертувати ці 3 канали.

Давайте побачимо 3 канали кольорового зображення, побудувавши його в гістограмі.

Зображення на вході:



```
# Weneed cv2 moduleforimage  
# readingandmatplotlibmodule  
# forplotting
```

```

import cv2
import matplotlib.pyplot as plt

img_bgr = cv2.imread('scenary.jpg', 1)

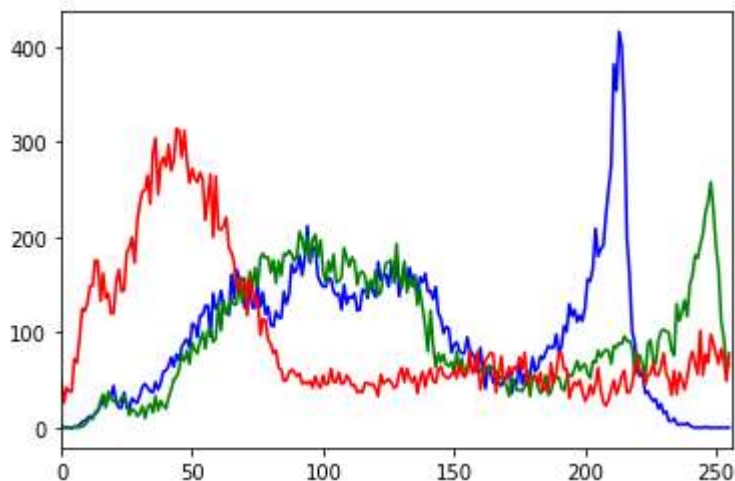
color = ('b', 'g', 'r')

for i, col in enumerate(color):
    histr = cv2.calcHist([img_bgr], [i], None, [256], [0,
256])
    plt.plot(histr, color = col)
    plt.xlim([0, 256])

plt.show()

```

На виході ми отримаємо такий результат:



Тут 3 канали (червоний, зелений, синій) перекриваються і створюють єдину гістограму. Якщо ви раніше вивчали пікселі та RGB, ви можете знати, що кожен колір містить 256 значень. Якщо значення RGB для кольору дорівнює (255, 255, 255), тоді цей колір відображається як білий, а якщо

значення RGB для кольору дорівнює (0, 0, 0), то цей колір відображається як чорний. Так само, зазначені вище 3 канали також містять 256 номерів пікселів.

Отже, вісь X показує загалом 256 значень (0 - 255), а вісь Y - загальні частоти кожного каналу. Як ви можете бачити на гістограмі, синій канал має найвищу частоту, і ви можете легко позначити кількість синього кольору, присутнього на зображенні, подивившись на нього.

Негативна трансформація зображення

Давайте створимо негативну трансформацію зображення. Існує 2 різних способи перетворення зображення в негатив за допомогою модуля OpenCV. Перший метод пояснює негативне перетворення поетапно, а другий - негативне перетворення зображення в один рядок.

Перший метод: Кроки для негативного перетворення

- Прочитайте зображення
- Отримайте висоту та ширину зображення
- Кожен піксель містить 3 канали. Отже, візьміть значення пікселя та зберіть 3 канали у 3 різних змінних.
- Перенесіть значення 3 пікселів від 255 і збережіть їх знову у пікселях, що використовувались раніше.
- Зробіть це для всіх значень пікселів, наявних на зображенні.

```
import cv2
import matplotlib.pyplot as plt

# Read an image
```

```

img_bgr = cv2.imread('scenary.jpg', 1)
plt.imshow(img_bgr)
plt.show()

# Histogramplottingoftheimage
color = ('b', 'g', 'r')

for i, col in enumerate(color):
    histr = cv2.calcHist([img_bgr],
                        [i], None,
                        [256],
                        [0, 256])
    plt.plot(histr, color = col)

    # Limit X - axisto 256
    plt.xlim([0, 256])

plt.show()

# getheightandwidthoftheimage
height, width, _ = img_bgr.shape

for i in range(0, height - 1):
    for j in range(0, width - 1):
        # Getthepixelvalue
        pixel = img_bgr[i, j]

        # Negateeachchannelby
        # subtractingitfrom 255

```

```

    # 1st indexcontainsredpixel
    pixel[0] = 255 - pixel[0]

    # 2nd indexcontainsgreenpixel
    pixel[1] = 255 - pixel[1]

    # 3rd indexcontainsbluepixel
    pixel[2] = 255 - pixel[2]

    # Storenewvaluesinthepixel
    img_bgr[i, j] = pixel

# Displaythenegativetransformedimage
plt.imshow(img_bgr)
plt.show()

# Histogramplottingofthe
# negativetransformedimage
color = ('b', 'g', 'r')

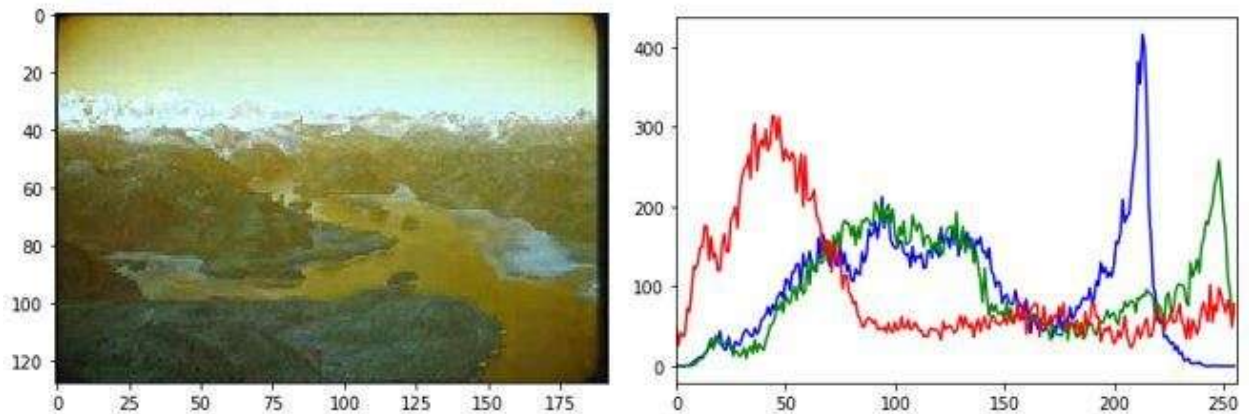
for i, col in enumerate(color):
    histr = cv2.calcHist([img_bgr],
                        [i], None,
                        [256],
                        [0, 256])

    plt.plot(histr, color = col)
    plt.xlim([0, 256])

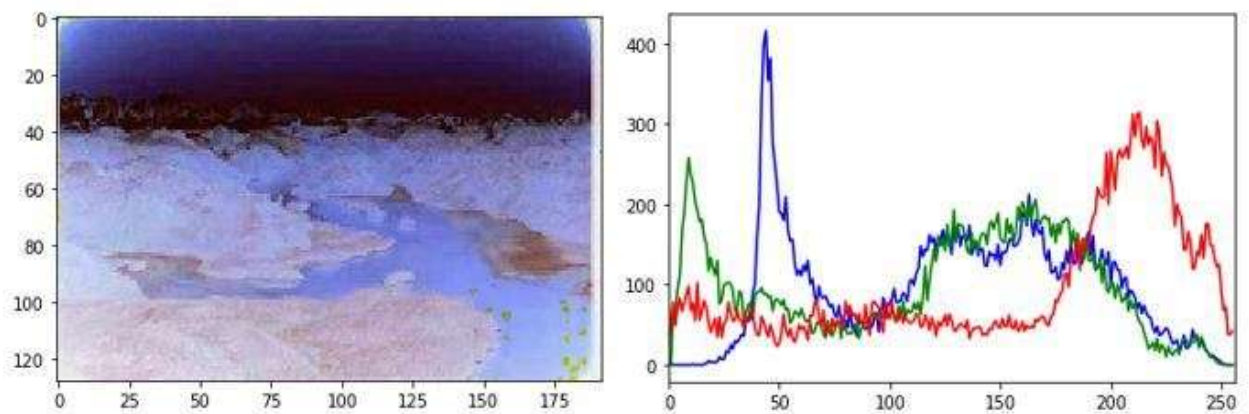
```

```
plt.show()
```

На виході ми отримаємо такий результат:



(Оригінальне зображення та його гістограма)



(Негативне зображення і його гістограма)

2-й метод: Кроки для негативного перетворення

- Прочитайте зображення та збережіть його у змінній.
- Відніміть змінну від 1 і збережіть значення в іншій змінній.
- Готово. Ви успішно зробили негативну трансформацію.


```
import cv2
import matplotlib.pyplot as plt

# Read an image
img_bgr = cv2.imread('scenary.jpg', 1)

plt.imshow(img_bgr)
plt.show()

# Histogram plotting of original image
color = ('b', 'g', 'r')

for i, col in enumerate(color):
    histr = cv2.calcHist([img_bgr],
                        [i], None,
                        [256],
                        [0, 256])

    plt.plot(histr, color = col)

    # Limit X - axis to 256
    plt.xlim([0, 256])

plt.show()

# Negate the original image
img_neg = 1 - img_bgr

plt.imshow(img_neg)
```

```
plt.show()

# Histogramplottingof
# negativetransformedimage
color = ('b', 'g', 'r')

for i, col in enumerate(color):
    histr = cv2.calcHist([img_neg],
                        [i], None,
                        [256],
                        [0, 256])

    plt.plot(histr, color = col)
    plt.xlim([0, 256])

plt.show()
```

Завдання:

На прикладі будь-якого кольорового зображення, виконати вищеописані методи перетворення позитив-негатив. Проаналізувати гістограми зображень.

Лабораторна робота 3.

Основи обробки зображень. Виділення та робота із каналами кольорів в системі RGB.

Мета:

- Ви дізнаєтесь, як конвертувати зображення з одного кольорового простору в інший, наприклад BGR ↔ Gray, BGR ↔ HSV тощо.
- Окрім цього, ми створимо додаток, який виділяє кольоровий об'єкт з відео.
- Ви дізнаєтесь такі функції: cv2.cvtColor (), cv2.inRange () тощо.
-

Зміна колірної простору

У OpenCV доступно понад 150 методів перетворення кольорового простору. Але ми розглянемо лише дві найбільш широко використовувані, BGR ↔ Grey та BGR ↔ HSV.

Для перетворення кольорів ми використовуємо функцію cv2.cvtColor(input_image, flag), де прапор визначає тип перетворення.

Для перетворення BGR ↔ Gray ми використовуємо прапори cv2.COLOR_BGR2GREY. Подібно до BGR ↔ HSV ми використовуємо прапор cv2.COLOR_BGR2HSV. Щоб отримати інші прапори, просто запустіть наступні команди у своєму терміналі Python:

```
>>>importcv2
>>>flags=[iforiindir(cv2)ifi.startswith('COLOR_')]
>>>printflags
```

Для HSV діапазон відтінків дорівнює [0,179], діапазон насиченості [0,255] і діапазон значень [0,255]. Різні програмні засоби використовують

різні масштаби. Отже, якщо ви порівнюєте з ними значення OpenCV, вам потрібно нормалізувати ці діапазони.

Відстеження об'єктів (ObjectTracking)

Тепер ми знаємо, як перетворити зображення BGR у HSV, ми можемо використовувати це для вилучення кольорового об'єкта. У HSV легше представити колір, ніж кольоровий простір RGB. У нашому додатку ми спробуємо витягти об'єкт синього кольору. Отже, ось метод:

- Вилучити кожен кадр з відео
- Перетворити з BGR на кольоровий простір HSV
- Ми обмежуємо зображення HSV для діапазону синього кольору

Нижче наведено код, який докладно коментується:

```
import cv2
import numpy as np

cap = cv2.VideoCapture(0)

while(1):

    # Take each frame
    _, frame = cap.read()

    # Convert BGR to HSV
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

    # define range of blue color in HSV
    lower_blue = np.array([110,50,50])
```

```

upper_blue = np.array([130,255,255])

# Threshold the HSV image to get only blue colors
mask = cv2.inRange(hsv, lower_blue, upper_blue)

# Bitwise-AND mask and original image
res = cv2.bitwise_and(frame,frame, mask=mask)

cv2.imshow('frame',frame)
cv2.imshow('mask',mask)
cv2.imshow('res',res)
k = cv2.waitKey(5) & 0xFF
if k == 27:
break

cv2.destroyAllWindows()

```

Нижче на зображенні показано відстеження синього об'єкта:



Як знайти значення HSV для відстеження?

Це поширене запитання, яке можна знайти на stackoverflow.com. Це дуже просто, і ви можете використовувати ту саму функцію, `cv2.cvtColor()`. Замість того, щоб передати зображення, ви просто передаєте потрібні значення BGR. Наприклад, щоб знайти значення HSV зеленого, спробуйте наступні команди в терміналі Python:

```
>>>green=np.uint8([[[0,255,0]]])
>>>hsv_green=cv2.cvtColor(green,cv2.COLOR_BGR2HSV)
>>>printhsv_green
[[[ 60 255 255]]]
```

Тепер ви берете [H-10, 100,100] та [H + 10, 255, 255] як нижню межу та верхню межу відповідно. Окрім цього методу, ви можете використовувати будь-які інструменти редагування зображень, такі як GIMP або будь-які онлайн-перетворювачі, щоб знайти ці значення, але не забудьте налаштувати діапазони HSV.

Завдання:

На прикладі будь-якого кольорового зображення спробуйте знайти спосіб вилучити більше одного кольорового об'єкта, наприклад, одночасно витягти червоний, синій, зелений об'єкти.

Лабораторна робота 4.

Просторова фільтрація зображень. Імплементація технології EdgeDetection.

Цілі:

- Навчіться застосовувати різні геометричні перетворення до таких зображень, як переклад, обертання, афінне перетворення тощо.
- Познакомитись із функціями: `cv2.getPerspectiveTransform`.

Трансформації

OpenCV надає дві функції перетворення, `cv2.warpAffine` та `cv2.warpPerspective`, за допомогою яких ви можете здійснювати всі види перетворень. `cv2.warpAffine` бере матрицю перетворення 2×3 , тоді як `cv2.warpPerspective` бере матрицю перетворення 3×3 як вхід.

Масштабування

Масштабування - це просто зміна розміру зображення. Для цього OpenCV постачається з функцією `cv2.resize()`. Розмір зображення можна вказати вручну, а можна вказати коефіцієнт масштабування. Використовуються різні методи інтерполяції. Переважними методами інтерполяції є `cv2.INTER_AREA` для зменшення та `cv2.INTER_CUBIC` (повільний) та `cv2.INTER_LINEAR` для збільшення. За замовчуванням використовується метод інтерполяції `cv2.INTER_LINEAR` для всіх цілей зміни розміру. Ви можете змінити розмір вхідного зображення одним із наступних способів:

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

```
img=cv2.imread('messi5.jpg')

res=cv2.resize(img,None,fx=2,fy=2,interpolation=cv2.INTER_CUBIC)

#OR

height,width=img.shape[:2]
res=cv2.resize(img,(2*width,2*height),interpolation=cv2.INTER_CUBIC)
```

Трансляція (Translation)

Трансляція- це зміна місця розташування об'єкта. Якщо ви знаєте зсув у напрямку (x, y), нехай це буде (t_x, t_y), ви можете створити матрицю перетворення M наступним чином:

$$M = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \end{bmatrix}$$

Її можна представити у вигляді масиву Numpy типу np.float32 і передати його у функцію cv2.warpAffine(). Дивіться код нижче для зміщення (100,50):

```
importcv2
importnumpyasnp

img=cv2.imread('messi5.jpg',0)
rows,cols=img.shape
```



```
M=np.float32([[1,0,100],[0,1,50]])
dst=cv2.warpAffine(img,M,(cols,rows))

cv2.imshow('img',dst)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Третім аргументом функції `cv2.warpAffine()` є розмір вихідного зображення, який повинен бути у формі (ширина, висота). Запам'ятайте ширину = кількість стовпців і висоту = кількість рядків.

Результат роботи коду:



Ротація (Rotation)

Поворот зображення для кута θ досягається матрицею перетворення форми:

$$M = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

Але OpenCV забезпечує масштабоване обертання з регульованим центром обертання, так що ви можете обертати в будь-якому місці, яке вам більше подобається. Модифікована матриця перетворення задається формулою:

$$\begin{bmatrix} \alpha & \beta & (1 - \alpha) \cdot center.x - \beta \cdot center.y \\ -\beta & \alpha & \beta \cdot center.x + (1 - \alpha) \cdot center.y \end{bmatrix}$$

де:

$$\alpha = scale \cdot \cos \theta,$$

$$\beta = scale \cdot \sin \theta$$

Щоб знайти цю матрицю перетворень, OpenCV надає функцію `cv2.getRotationMatrix2D`. Подивіться нижче приклад, який повертає зображення на 90 градусів відносно центру без масштабування.

```
img=cv2.imread('messi5.jpg',0)
rows,cols=img.shape

M=cv2.getRotationMatrix2D((cols/2,rows/2),90,1)
dst=cv2.warpAffine(img,M,(cols,rows))
```

Результат роботи коду:



Афінна трансформація (AffineTransformation)

При афінному перетворенні всі паралельні лінії на вихідному зображенні все одно будуть паралельними на вихідному зображенні. Щоб знайти матрицю перетворення, нам потрібні три точки від вхідного зображення та їх відповідних місць у вихідному зображенні. Тоді

`cv2.getAffineTransform` створить матрицю 2×3 , яку потрібно передати `cv2.warpAffine`.

Перегляньте приклад нижче, а також перегляньте обрані мною точки (які позначені зеленим кольором):

```
img=cv2.imread('drawing.png')
rows,cols,ch=img.shape

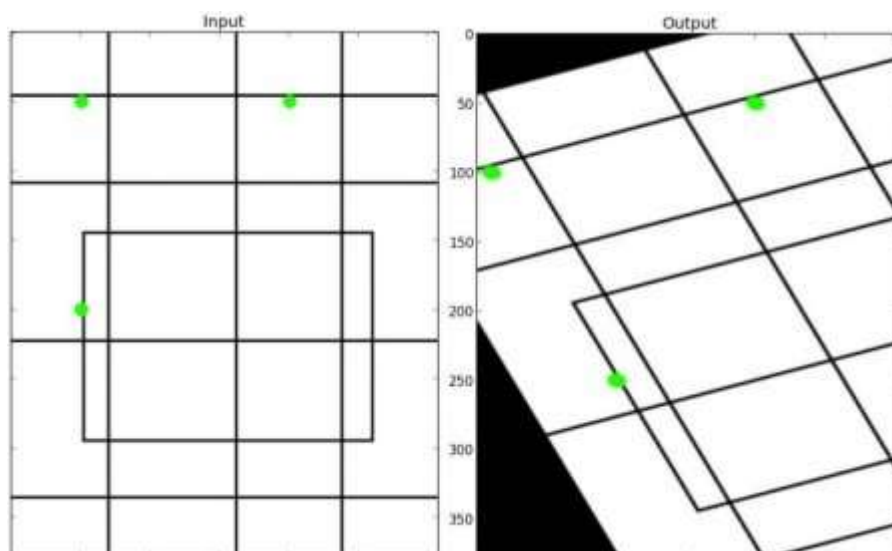
pts1=np.float32([[50,50],[200,50],[50,200]])
pts2=np.float32([[10,100],[200,50],[100,250]])

M=cv2.getAffineTransform(pts1,pts2)

dst=cv2.warpAffine(img,M,(cols,rows))

plt.subplot(121),plt.imshow(img),plt.title('Input')
plt.subplot(122),plt.imshow(dst),plt.title('Output')
plt.show()
```

Результат роботи коду:



Трансформація перспективи (PerspectiveTransformation)

Для перспективного перетворення вам потрібна матриця перетворення 3x3. Прямі лінії залишаються прямими навіть після перетворення. Щоб знайти цю матрицю перетворення, вам потрібно 4 точки на вхідному зображенні та відповідні точки на вихідному зображенні. Серед цих 4 пунктів 3 з них не повинні бути колінеарними. Тоді матрицю перетворення можна знайти за допомогою функції `cv2.getPerspectiveTransform`. Потім застосуйте `cv2.warpPerspective` з цією матрицею перетворення 3x3.

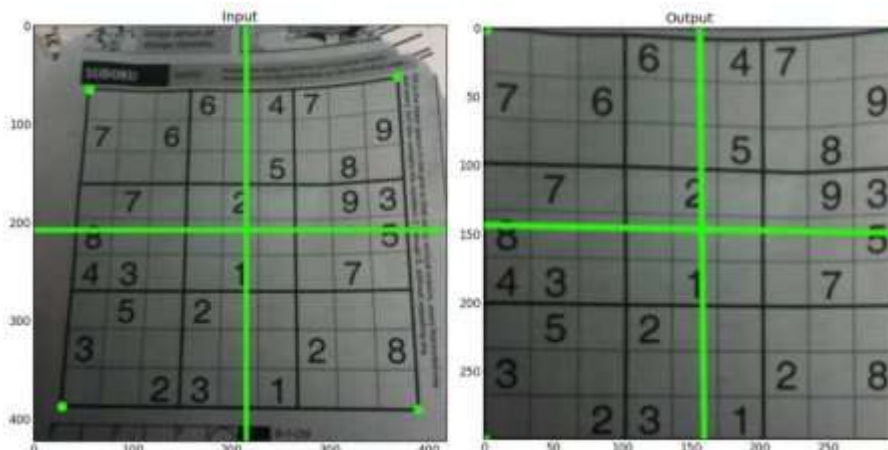
```
img=cv2.imread('sudokusmall.png')
rows,cols,ch=img.shape

pts1=np.float32([[56,65],[368,52],[28,387],[389,390]])
pts2=np.float32([[0,0],[300,0],[0,300],[300,300]])

M=cv2.getPerspectiveTransform(pts1,pts2)
dst=cv2.warpPerspective(img,M,(300,300))

plt.subplot(121),plt.imshow(img),plt.title('Input')
plt.subplot(122),plt.imshow(dst),plt.title('Output')
plt.show()
```

Результат роботи коду:



CannyEdgeDetection

CannyEdgeDetection - популярний алгоритм виявлення країв. Він був розроблений Джоном Ф. Кенні в 1986 році. Це багатоступеневий алгоритм, і ми пройдемо кожен етап.

1. Зменшення шуму

Оскільки виявлення країв сприйнятливим до шуму на зображенні, першим кроком є усунення шуму на зображенні за допомогою Гаусового фільтра 5x5.

2. Пошук градієнта інтенсивності зображення

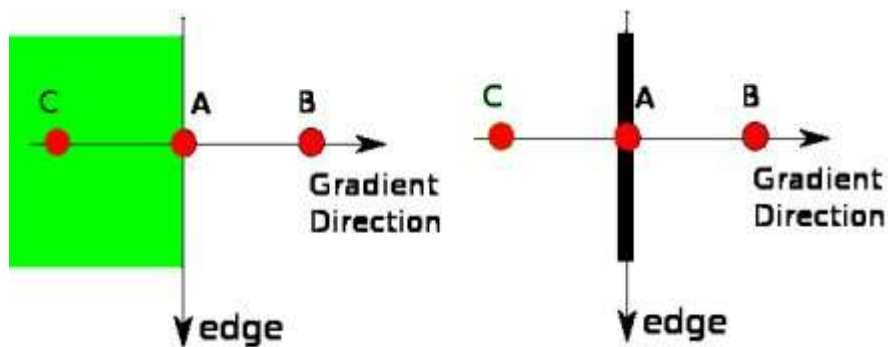
Потім згладжене зображення фільтрується ядром Собеля як у горизонтальному, так і у вертикальному напрямку, щоб отримати першу похідну в горизонтальному напрямку (G_x) і вертикальному напрямку (G_y). З цих двох зображень ми можемо знайти градієнт краю та напрямок для кожного пікселя наступним чином:

$$\begin{aligned} \text{Edge_Gradient } (G) &= \sqrt{G_x^2 + G_y^2} \\ \text{Angle } (\theta) &= \tan^{-1} \left(\frac{G_y}{G_x} \right) \end{aligned}$$

Напрямок градієнта завжди перпендикулярний до країв. Він округлений до одного з чотирьох кутів, що представляють вертикальний, горизонтальний та два діагональні напрямки.

3. Немаксимальнешумоподавлення

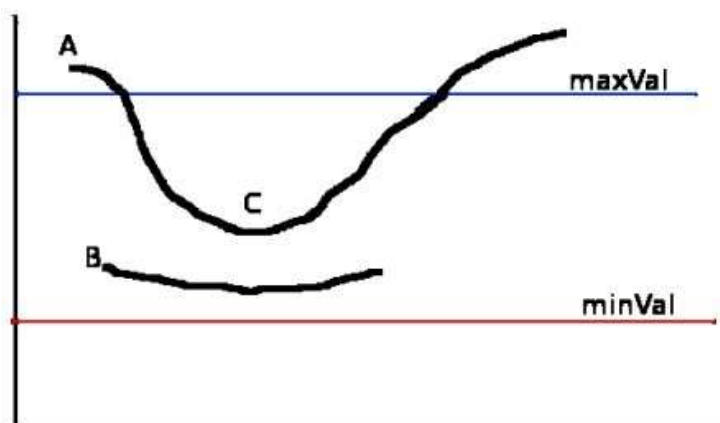
Після отримання величини та напрямку градієнта виконується повне сканування зображення, щоб видалити всі небажані пікселі, які не можуть становити край. Для цього на кожному пікселі перевіряється піксель, якщо він є локальним максимумом у його сусідстві у напрямку градієнта. Подивіться на зображення нижче:



Точка А знаходиться на краю (у вертикальному напрямку). Напрямок градієнта нормальний до краю. Точки В і С знаходяться в градієнтних напрямках. Отже, точка А перевіряється за допомогою точок В і С, чи не утворює вона локальний максимум. Якщо так, це розглядається для наступного етапу, інакше воно придушується (ставиться до нуля). В результаті ви отримуєте бінарне зображення з «тонкими краями».

4. Порогове обмеження гістерезису

На цьому етапі вирішується, які всі краї насправді є ребрами (краями), а які ні. Для цього нам потрібні два порогові значення, $minVal$ і $maxVal$. Будь-які ребра з градієнтом інтенсивності, що перевищують $maxVal$, обов'язково будуть ребрами, а ті, що нижче $minVal$, обов'язково будуть неребрами, тому відкидаються. Ті, хто лежить між цими двома порогамі, класифікуються як ребра, так і неребра на основі їх зв'язку. Якщо вони підключені до пікселів із «достовірним краєм», вони вважаються частиною країв. В іншому випадку їх також відкидають. Дивіться зображення нижче:



Край А знаходиться вище *maxVal*, тому його розглядають як “вірний край”. Хоча ребро С нижче *maxVal*, воно з'єднане з ребром А, так що це також вважається дійсним ребром, і ми отримуємо повну криву. Але край В, хоча він і вище *minVal*, і знаходиться в тій же області, що і край С, він не підключений до жодного “надійного краю”, тому його відкидають. Тож дуже важливо, щоб ми мали вибрати *minVal* та *maxVal* відповідно, щоб отримати правильний результат.

На цьому етапі також видаляються шуми малих пікселів при припущенні, що краї є довгими лініями.

Отже, нарешті ми отримуємо сильні краї зображення.

CannyEdgeDetectioninOpenCV

OpenCV розміщує все вищезазначене в одній функції, `cv2.Canny()`. Ми побачимо, як ним користуватися. Перший аргумент - це наше вхідне зображення. Другий та третій аргументи - це наші *minVal* та *maxVal* відповідно. Третій аргумент - *aperture_size*. Це розмір ядра Собеля, який використовується для пошуку градієнтів зображення. За замовчуванням це 3. Останній аргумент - *L2gradient*, який задає рівняння для пошуку величини градієнта. Якщо це істина, вона використовує вищевказане рівняння, яке є більш точним, інакше використовує цю функцію:
$$\text{Edge_Gradient } (G) = |G_x| + |G_y|$$

За замовчуванням це False.

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

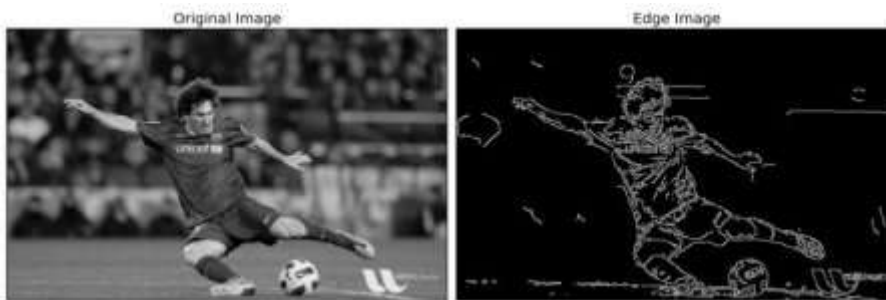
img = cv2.imread('messi5.jpg', 0)
```

```
edges=cv2.Canny(img,100,200)

plt.subplot(121),plt.imshow(img,cmap='gray')
plt.title('Original Image'),plt.xticks([]),plt.yticks([])
plt.subplot(122),plt.imshow(edges,cmap='gray')
plt.title('Edge Image'),plt.xticks([]),plt.yticks([])

plt.show()
```

Результат роботи коду:



Завдання:

1. На прикладі будь-якого кольорового зображення реалізуйте всі вищенаведені види геометричних перетворені (scaling, translation, rotation, affinetransformation, perspectivetransformation).
2. Імплементуйте алгоритм Canny edge detection. Проаналізуйте його роботу на двох типах зображень – зображення з чіткими виділеними прямими лініями (наприклад, зображення будинку), та зображення без таких (наприклад пейзаж). Поясніть відмінності.

Лабораторна робота 5.

Частотна фільтрація зображень. Низькочастотні, високочастотні та інші фільтри.

Перетворення Фур'є

Перетворення Фур'є використовується для аналізу частотних характеристик різних фільтрів. Для зображень використовується 2D дискретне перетворення Фур'є (DFT) для пошуку частотної області. Для розрахунку DFT використовується швидкий алгоритм, який називається швидким перетворенням Фур'є (ШПФ). Детальну інформацію про них можна знайти в будь-яких підручниках з обробки зображень або обробки сигналів.

Для синусоїдального сигналу

$$x(t) = A \sin(2\pi ft)$$

ми можемо сказати, що f - частота сигналу, і якщо взяти його частотну область, ми можемо побачити сплеск у f . Якщо сигнал семплується для формування дискретного сигналу, ми отримуємо ту саму частотну область, але періодичну в діапазоні $[-\pi, \pi]$ або $[0, 2\pi]$ (або $[0, N]$ для N -точкового DFT). Ви можете розглядати зображення як сигнал, який дискретизується у двох напрямках. Отже, реалізуючи перетворення Фур'є в обох напрямках X та Y , ви отримуєте частотне представлення зображення.

Більш інтуїтивно для синусоїдального сигналу, амплітуда якого змінюється швидко за короткий час, можна сказати, що це високочастотний сигнал. Якщо він змінюється повільно, це сигнал низької частоти. Ви можете поширити ту саму ідею на зображення. Де амплітуда різко змінюється на зображеннях? У крайових точках, або у зашумлених точках. Тож можна сказати, що краї та шуми - це високочастотний вміст зображення. Якщо амплітуда змінюється не різко, це низькочастотна складова.

Тепер ми побачимо, як реалізувати перетворення Фур'є.

Перетворення Фур'є в NumPy

Функція `np.fft.fft2()` забезпечує перетворення частоти, яке буде комплексним масивом. Першим його аргументом є вхідне зображення, яке має відтінки сірого. Другий аргумент необов'язковий - він визначає розмір вихідного масиву. Якщо воно більше за розмір вхідного зображення, перед обчисленням ШПФ вхідне зображення заповнюється нулями. Якщо воно менше вхідного зображення, вхідне зображення буде обрізане. Якщо аргументів не передано, розмір вихідного масиву буде однаковим із вхідним.

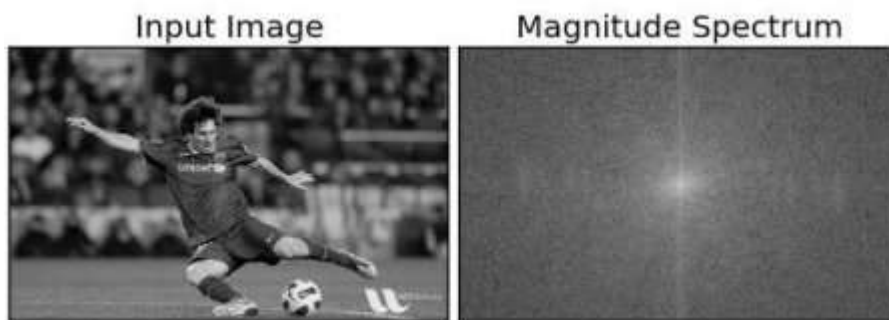
Тепер, як тільки ви отримаєте результат, нульова частотна складова (компонент постійного струму) буде у верхньому лівому куті. Якщо ви хочете піднести його до центру, вам потрібно зсунути результат на $N/2$ в обох напрямках. Це просто робить функція `np.fft.fftshift()`. Тепер можна побудувати спектр зображення (magnitude spectrum).

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

img = cv2.imread('messi5.jpg', 0)
f = np.fft.fft2(img)
fshift = np.fft.fftshift(f)
magnitude_spectrum = 20 * np.log(np.abs(fshift))

plt.subplot(121), plt.imshow(img, cmap='gray')
plt.title('Input Image'), plt.xticks([]), plt.yticks([])
plt.subplot(122), plt.imshow(magnitude_spectrum, cmap='gray')
plt.title('Magnitude Spectrum'), plt.xticks([]), plt.yticks([])
plt.show()
```

Результат роботи коду:



Можна побачити більше білого регіону в центрі, де вміст низьких частот більше.

Отже, ви знайшли перетворення частоти. Тепер ви можете виконувати деякі операції в частотній області, наприклад, фільтрацію високих частот та реконструювати зображення, тобто знаходити зворотний DFT. Для цього ви просто видаляєте низькі частоти, маскуючи їх прямокутним вікном розміром 60x60. Потім застосуйте зворотний зсув, використовуючи `np.fft.ifftshift()`, щоб компонент постійного струму знову потрапив у верхній лівий кут. Потім знайдіть зворотний ШПФ за допомогою функції `np.ifft2()`. Результат, знову ж таки, буде комплексним числом. Ви можете взяти його абсолютне значення.

```

rows,cols=img.shape
crow,ccol=rows/2,cols/2
fshift[crow-30:crow+30,ccol-30:ccol+30]=0
f_ishift=np.fft.ifftshift(fshift)
img_back=np.fft.ifft2(f_ishift)
img_back=np.abs(img_back)

plt.subplot(131),plt.imshow(img,cmap='gray')
plt.title('Input Image'),plt.xticks([]),plt.yticks([])
plt.subplot(132),plt.imshow(img_back,cmap='gray')
plt.title('Image after HPF'),plt.xticks([]),plt.yticks([])
plt.subplot(133),plt.imshow(img_back)

```

```
plt.title('Result in JET'),plt.xticks([]),plt.yticks([])  
  
plt.show()
```

Результат роботи коду:



Якщо ви уважно спостерігаєте за результатом, особливо за останнім зображенням у кольорі JET, ви можете побачити деякі артефакти (один приклад позначено червоною стрілкою). Він показує там деякі структури, схожі на брижі (ripple-like structures), і це називається ефектами дзвону (ringing effects). Це викликано прямокутним вікном, яке ми використовували для маскування. Ця маска перетворюється на форму sinc, що спричиняє цю проблему. Тож прямокутні вікна не використовуються для фільтрації. Кращий варіант – Гауссове вікно.

Завдання:

1. На прикладі будь-якого кольорового зображення реалізуйте ШПФ. Проаналізуйте результат у вигляді magnitude spectrum та у формі JET.
2. Додайте статистичного шуму до зображення. Повторно виконайте завдання пункту 1.

Лабораторна робота 6.

Робота із контрастністю та різкістю зображень.

Контраст - це різниця у яскравості або кольорі, що робить об'єкт (або його представлення на зображенні чи дисплеї) помітним. При візуальному сприйнятті реального світу контраст визначається різницею кольорів та яскравості предмета та інших предметів у тому самому полі зору. Зорова система людини більш чутлива до контрасту, ніж до абсолютної яскравості; ми можемо сприймати світ так само, незалежно від величезних змін освітленості протягом дня.

Піксельні перетворення

При такому перетворенні обробки зображень значення кожного вихідного пікселя залежить лише від відповідного значення вхідного пікселя (плюс, можливо, якась зібрана глобальна інформація або параметри).

Прикладами таких операторів є регулювання яскравості та контрасту, а також корекція кольору та трансформація.

Регулювання яскравості та контрасту

Два часто використовувані точкові процеси - це множення та додавання з константою:

$$g(x)=\alpha f(x)+\beta$$

Параметри $\alpha > 0$ і β часто називають параметрами коефіцієнта підсилення та зміщення; іноді кажуть, що ці параметри контролюють контраст і яскравість відповідно.

Ви можете вважати $f(x)$ джерелом пікселей вхідного зображення, а $g(x)$ - пікселей вихідного зображення. Тоді ми можемо написати вираз як:

$$g(i,j)=\alpha f(i,j)+\beta$$

де i та j вказують, що піксель знаходиться в i -му рядку та j -му стовпці.

Наступний код виконує операцію $g(i,j)=\alpha f(i,j)+\beta$:

```
from __future__ import print_function
from builtins import input
import cv2 as cv
import numpy as np
import argparse

# Read image given by user
## [basic-linear-transform-load]
parser = argparse.ArgumentParser(description='Code for
Changing the contrast and brightness of an image!
tutorial.')
parser.add_argument('--input', help='Path to input image.',
default='lena.jpg')
args = parser.parse_args()

image = cv.imread(cv.samples.findFile(args.input))
if image is None:
    print('Could not open or find the image: ', args.input)
    exit(0)
## [basic-linear-transform-load]

## [basic-linear-transform-output]
new_image = np.zeros(image.shape, image.dtype)
## [basic-linear-transform-output]

## [basic-linear-transform-parameters]
alpha = 1.0 # Simple contrast control
beta = 0    # Simple brightness control
```

```

# Initialize values
print(' Basic Linear Transforms ')
print('-----')
try:
    alpha = float(input('* Enter the alpha value [1.0-3.0]: '))
    beta = int(input('* Enter the beta value [0-100]: '))
except ValueError:
    print('Error, not a number')
## [basic-linear-transform-parameters]

# Do the operation new_image(i,j) = alpha*image(i,j) + beta
# Instead of these 'for' loops we could have used simply:
# new_image = cv.convertScaleAbs(image, alpha=alpha,
beta=beta)
# but we wanted to show you how to access the pixels :)
## [basic-linear-transform-operation]
for y in range(image.shape[0]):
    for x in range(image.shape[1]):
        for c in range(image.shape[2]):
new_image[y,x,c] = np.clip(alpha*image[y,x,c] + beta, 0,
255)
## [basic-linear-transform-operation]

## [basic-linear-transform-display]
# Show stuff
cv.imshow('Original Image', image)
cv.imshow('New Image', new_image)

```

```
# Wait until user press some key
cv.waitKey()
## [basic-linear-transform-display]
```

Результат роботи коду та зі значеннями $\alpha = 2,2$ та $\beta = 50$:



Завдання:

На прикладі будь-якого кольорового зображення реалізуйте алгоритм регулювання яскравості та контрасту. Проаналізуйте вплив параметрів α та β на результат обробки.

Перелік навчально-методичної літератури

1. Базова література

1. Гонсалес Р., Вудс Р. Цифровая обработка изображений. -. М.: Техносфера, 2005.
2. Фисенко В.Т., Фисенко Т.Ю. Компьютерная обработка и распознавание изображений. ИТМО, Санкт-Петербург, 2008.
3. Оппенгейм Э. Применение цифровой обработки сигналов. М.Мир, 1980
4. Марк Лутц – Изучаем Python. – М.: Диалектика, 2020.

Посилання на інформаційні ресурси в Інтернеті, відео-лекції, інше методичне забезпечення

1. <https://www.youtube.com/watch?v=YYXdXT2l-Gg&list=PL-osiE80TeTskrapNbzxhwoFUilCjGgY7>
2. <https://www.youtube.com/watch?v=YYXdXT2l-Gg&list=PL-osiE80TeTt2d9bfVyTiXJA-UTHn6WwU>
3. <https://www.python.org/doc/>
4. https://docs.opencv.org/master/d6/d00/tutorial_py_root.html