**Міністерство освіти і науки України**
**Національний технічний університет України**
**«Київський політехнічний інститут імені Ігоря Сікорського»**
**Факультет інформатики та обчислювальної техніки**
**Кафедра обчислювальної техніки**

**Лабораторна робота №2**
з дисципліни
«Технології Computer Vision»
на тему
«Дослідження алгоритмів формування та обробки растрових цифрових зображень»

Виконала:
Студентка групи ІМ-21
Кривохата Марія Юріївна
Номер у списку групи: 12

Перевірив: Баран Д. Р.

Київ 2024

**Мета:**

Виявити дослідити та узагальнити особливості реалізації алгоритмів растрової цифрових зображень на прикладі застосування алгоритмів растеризації, побудови складних 3D растрових об'єктів та застосування технологій корекції характеристик кольору окремих растрів цифрових зображень.

**Завдання:**

***<u>Рівень III</u>***

**Завдання III рівня – максимально 9 балів.**

Реалізувати розробку програмного скрипта, що реалізує корекцію кольору цифрового растрового зображення з переліку: зміна яскравості, відтінки сірого, негатив, серпія – в градієнтах: діагональ (будь-який напрям); від центру; до центра. Обробку реалізувати на рівні матриці растра. Зображення обрати самостійно. Рекомендується обрати реалістичне зображення, наприклад із джерел:

https://www.kaggle.com/
https://paperswithcode.com
https://www.sentinel-hub.com/
https://livingatlas2.arcgis.com/landsatexplorer/
https://www.bing.com/maps
https://mapcarta.com/Map
https://eos.com/uk/gallery/

**Синтезовані математичні моделі:**

*<u>Відтінки сірого</u>*

1. ***Математична модель***

Математична модель перетворення зображення у відтінки сірого описується звичайним знаходженням середнього арифметичного значення трьох компонентів кольору r, g, b.

$$I = \frac{R + G + B}{3}$$

В коді розрахунок кольору одного пікселя виглядатиме так:

```
gray = (r + g + b) // 3
```

Однак! Такий підхід не є найефективнішим способом застосування ефекту відтінків сірого, бо не враховує особливості сприйняття кольорів людським оком. Наприклад, є інший метод, що враховує більшу чутливість людського ока до зеленого кольору і меншу до синього. Такий метод дає більш «реальні» результати, але вимагає використання захардкоджених коефіцієнтів для кожного кольору (замість «рівноправних» коефіцієнтів середнього аифметичного). Сприйняття кольору у різних людей відрізняється, а значить кофіцієнти також можуть бути різними, тож, на мою суб'єктивну думку, середнє арифметичне значення є більш універсальним рішенням.

*<u>Сепія</u>*

1. ***Математична модель***

Перетворення зображення за допомогою ефекту сепії полягає в тому, щоб спочатку перетворити зображення у відтінки сірого, а лише після цього застосувати формулу для отримання нових коричнево-червоних «ретро» відтінків.

$I = \frac{R+G+B}{3}$ - отримали сірий колір

Червоний канал отримує найбільший зсув на основі значення depth, додаючи подвійну глибину до сірого значення:

$R_{sepia} = min(255, I + 2 \cdot depth)$

Зелений канал отримує середній зсув, додаючи просто значення depth до сірого кольору:

$G_{sepia} = min(255, I + depth)$

Синій канал не отримує жодного додаткового зсуву і залишається на рівні сірого кольору:

$B_{sepia} = min(255, I)$

Всі значення були обмежені функцією min, щоб не виходити за область допустимих значень [0, 255].

В коді загальний процес перетворення кольору для одного пікселя буде виглядати так:

```
gray = (r + g + b) // 3
r_sepia = min(255, gray + depth * 2)
g_sepia = min(255, gray + depth)
b_sepia = min(255, gray)
```

**depth** – «сила» сепії, яка впливає на те, наскільки ближчим колір пікселя буде до характерного коричнево-червоного кольору

*Негатив*

1. ***Математична модель***
   Математичну модель перетворення зображення за допомогою ефекту негативу можна описати як обчислення доповнення r, g, b до максимального значення – 255.

   R′ = 255 - R
   G′ = 255 - G
   B′ = 255 - B

   Тобто в коді розрахунок компонент кольору одного пікселя виглядатиме так:

```
r_neg = 255 - r
g_neg = 255 - g
b_neg = 255 - b
```

**r_neg, g_neg, b_neg** – нові значення кольорових каналів
**r, g, b** – старі значення

*Зміна яскравості*

1. ***Математична модель***
   У цій моделі до кожного компоненту кольору додається (для збільшення яскравості) або віднімається (для зменшення яскравості) деяке стале значення, яке називається зсувом яскравості.
   $R' = R + C$
   $G' = G + C$
   $B' = B + C$

   При цьому значення компонентів після зміни яскравості потрібно обмежити діапазоном [0, 255], щоб воно не виходило за межі допустимих значень.

   $R' = \min(255, \max(0, R+C))$

   $G' = \min(255, \max(0, G+C))$

   $B' = \min(255, \max(0, B+C))$

   У кінцевому варіанті в коді розрахунки кольору одного пікселя виглядатимуть так:

```
r = min(255, max(0, r + factor))
g = min(255, max(0, g + factor))
b = min(255, max(0, b + factor))
```

   **factor** – зсув яскравості (може бути від'ємним або додатнім)

*Накладання ефектів у вигляді градієнту різних напрямків*

1. ***Математична модель***
   1) Спочатку необхідно **створити маску** для майбутнього градієнту. Для кожного з трьох напрямків градієнту, які використовуються в рамках лабораторної роботи, ця маска буде різною.

      **Діагональний градієнт**
      Математична модель цього градієнту описується так:

      $$M[i,j] = \frac{i+j}{W+H}$$

      - M[i, j] – значення пікселя (i, j) у масці. При цьому: 0 <= M[i, j] <= 1
      - W та H – ширина та висота зображення

Таким чином значення маски збільшується, коли ми рухаємося по діагоналі від верхнього лівого до нижнього правого кута зображення. При цьому значення у масці нормалізовані між 0 і 1.

**<u>До ценру:</u>**

Для того, щоб скласти математичну модель маски цього градієнту, спочатку потрібно розрахувати відстань від певного пікселя до центру. Знаючи координати центру зображення, можемо легко зробити це за формулою:

$$d(i,j) = \sqrt{(i - \frac{W}{2})^2 + (j - \frac{H}{2})^2}$$

- W та H – ширина та висота зображення
- d(i, j) – відстань від пікселя (i, j) до центру зображення

Тоді остаточне значення пікселя маски буде рахуватись за таким принципом:

$$M[i,j] = \frac{d(i,j)}{d_{max}}$$

- M[i, j] – значення пікселя (i, j) у масці. При цьому: 0 <= M[i, j] <= 1
- $d_{max}$ – максимально можлива відстань (із будь-якого з кутів до центру зображення)

**<u>Від центру:</u>**
Математична модель маски цього градієнту – протилежність маски градієнту «до центру», тому вона може бути виражена як:

$$M[i,j] = 1 - \frac{d(i,j)}{d_{max}}$$

**2)** Далі необхідно **«змішати» разом оригінальне зображення та зображення із накладеним ефектом за допомогою вже визначеної маски градієнту:**

Припустимо, що:

- O[i, j] = ($r_o$, $g_o$, $b_o$) – оригінальний колір пікселя
- E[i, j] = ($r_e$, $g_e$, $b_e$) – колір пікселя після того, як до нього був застосований один із ефектів (відтінки сірого, сепія, негатив, зміна яскравості)

Тоді остаточний колір пікселя F[i, j] = ($r_f$, $g_f$, $b_f$) може бути розрахований за таким принципом:

$$r_f = M[i,j] \cdot r_e + (1 - M[i,j]) \cdot r_o$$

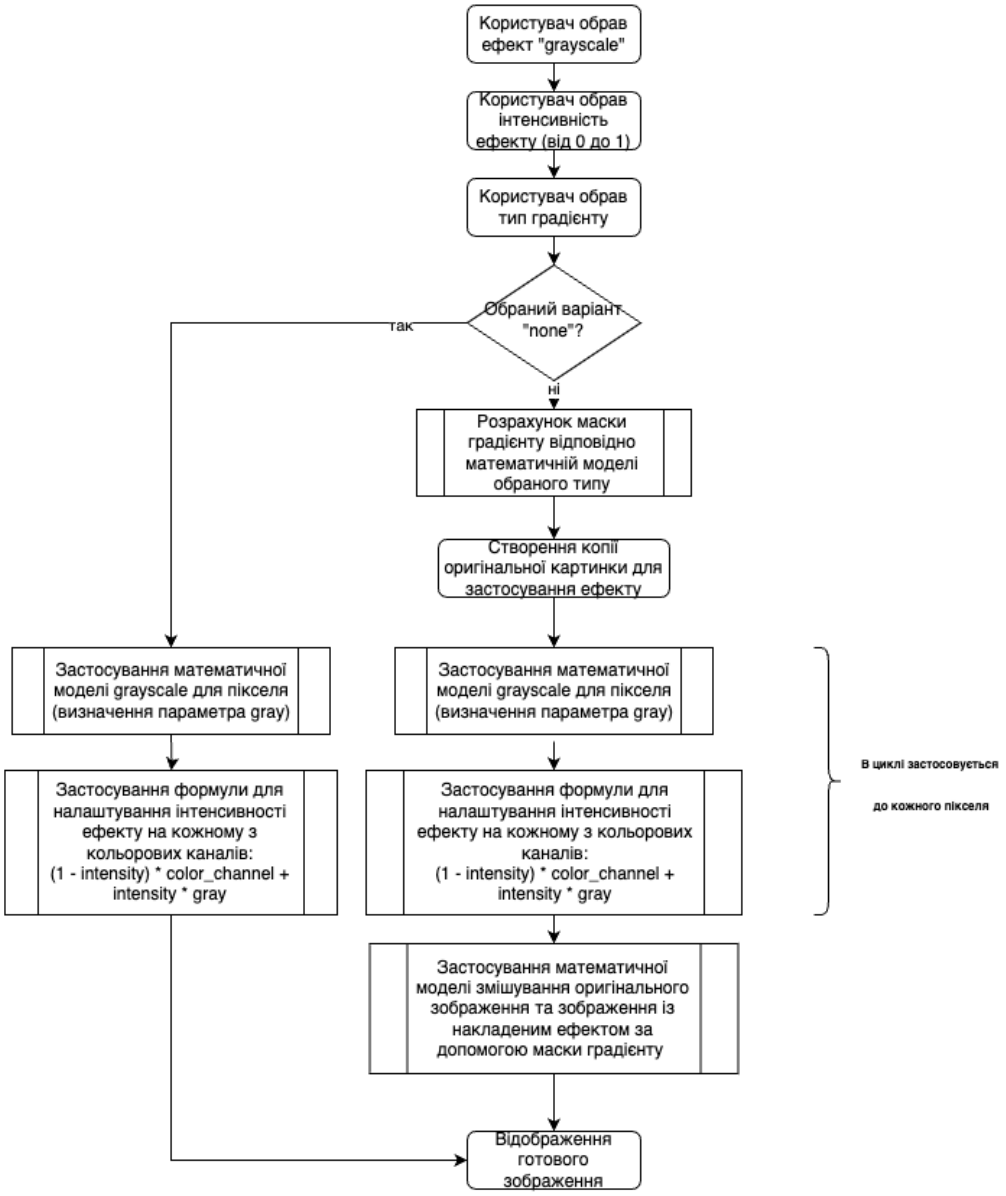
$$g_f = M[i,j] \cdot g_e + (1 - M[i,j]) \cdot g_o$$

$$b_f = M[i,j] \cdot b_e + (1 - M[i,j]) \cdot b_o$$

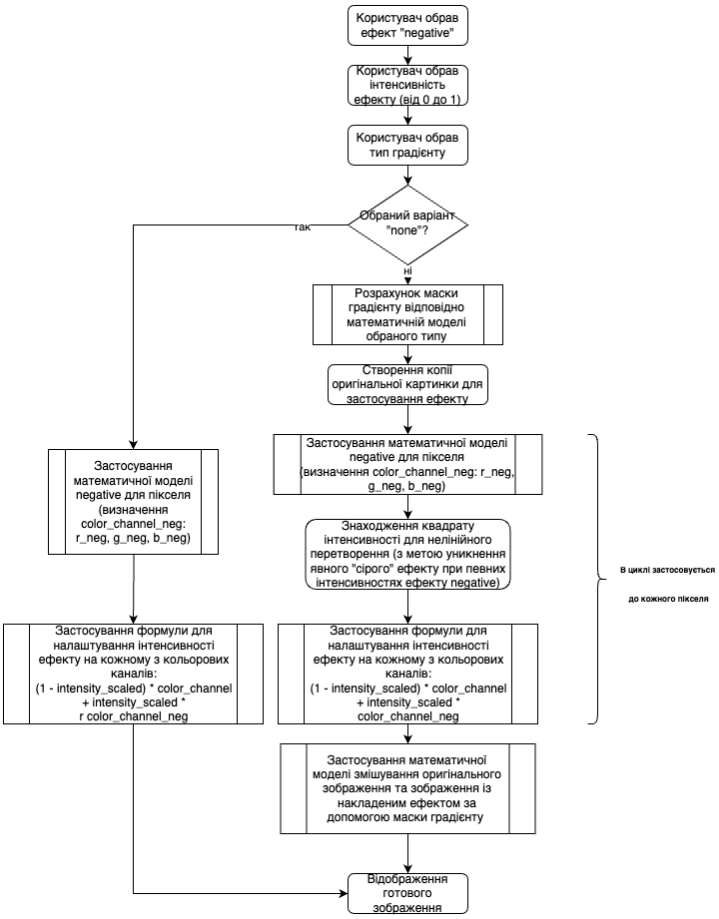
**Архітектурне проектування**

Для зручності у блок-схемах випадки були розбиті для кожного ефекту окремо (щоб не нагромаджувати розгалудження). Блок-схеми представлені не у класичному стилі – на певних етапах наявні детальніші пояснення причин певного рішення.

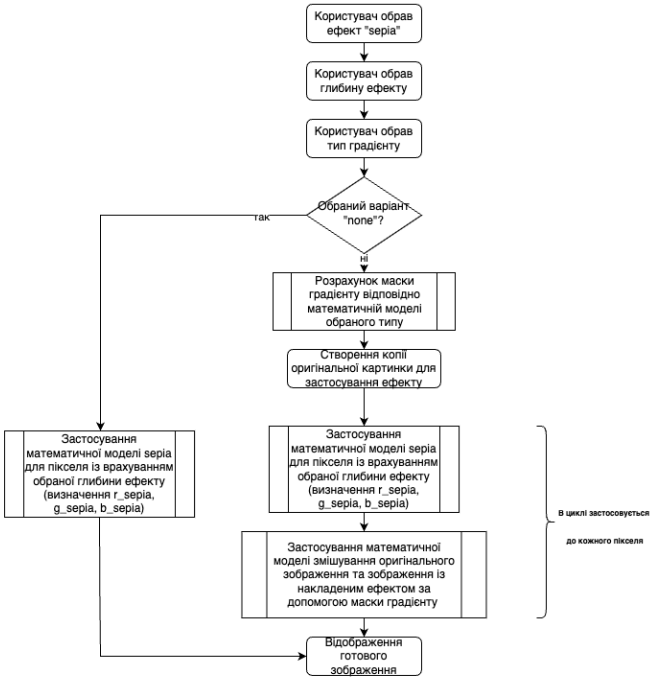
*Відтінки сірого*



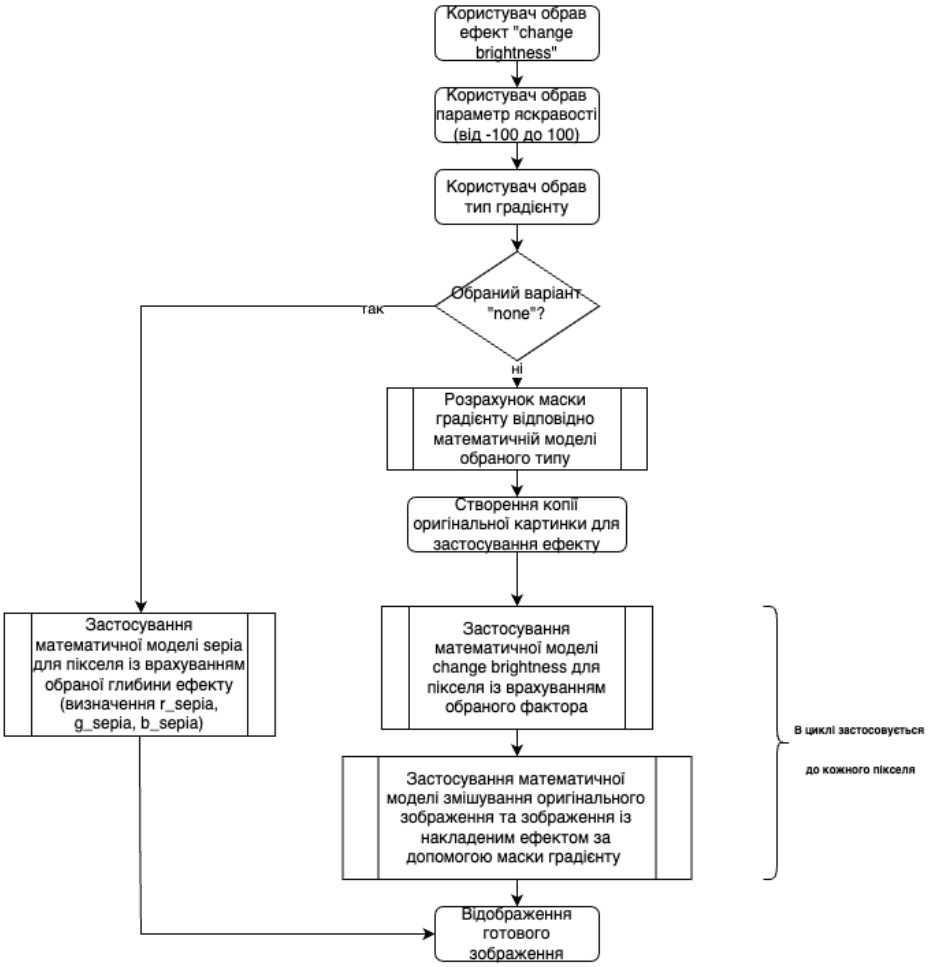

*Негатив*




## *Сепія*

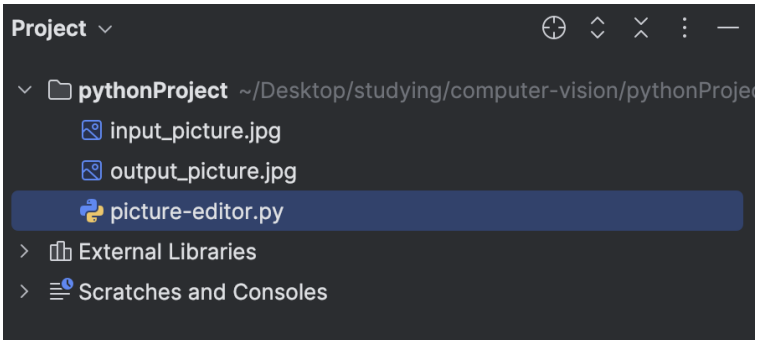



## *Зміна яскравості*




**Опис структури проєкту програми:**

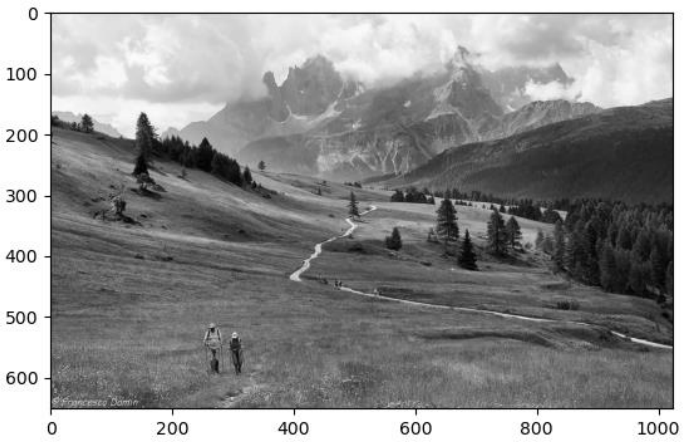
Структура проєкту є максимально простою: весь код програми знаходиться у файлі picture-editor.py, вхідне зображення у файлі input_picture.jpg, а результуюче зображення зберігається у файл output_picture.jpg у тій самій папці.




**Результати роботи програми відповідно до завдання:**

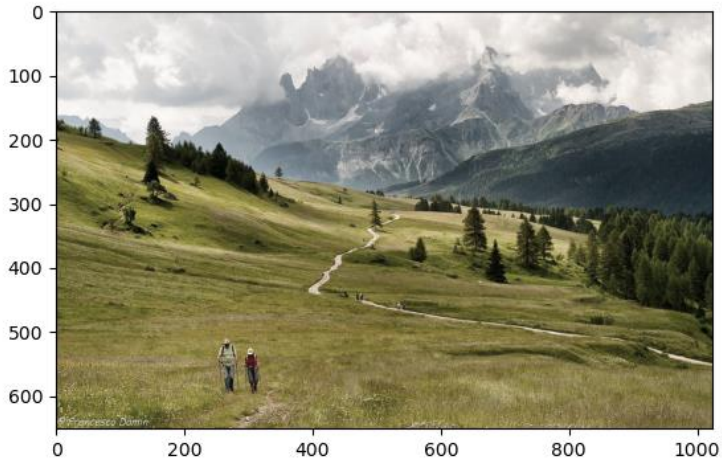
*Оригінал зображення:*

*Відтінки сірого; без градієнту; інтенсивність максимальна:*




```
Choose an effect:
0 - grayscale
1 - sepia
2 - negative
3 - change brightness
effect: 0
Choose a gradient direction:
0 - no gradient
1 - from center
2 - to center
3 - diagonal
gradient direction: 0
Enter effect intensity (0.0 to 1.0): 1
Image saved as output_picture.jpg
```

*Відтінки сірого; без градієнту; інтенсивність 0.5:*




```
Choose an effect:
0 - grayscale
1 - sepia
2 - negative
3 - change brightness
effect: 0
Choose a gradient direction:
0 - no gradient
1 - from center
2 - to center
3 - diagonal
gradient direction: 0
Enter effect intensity (0.0 to 1.0): 0.5
Image saved as output_picture.jpg
```

*Відтінки сірого; градієнт від центру; інтенсивність максимальна:*

```
Choose an effect:
0 - grayscale
1 - sepia
2 - negative
3 - change brightness
effect: 0
Choose a gradient direction:
0 - no gradient
1 - from center
2 - to center
3 - diagonal
gradient direction: 1
Enter effect intensity (0.0 to 1.0): 1
Image saved as output_picture.jpg
```

Сепія; градієнт діагональ; depth = 80:

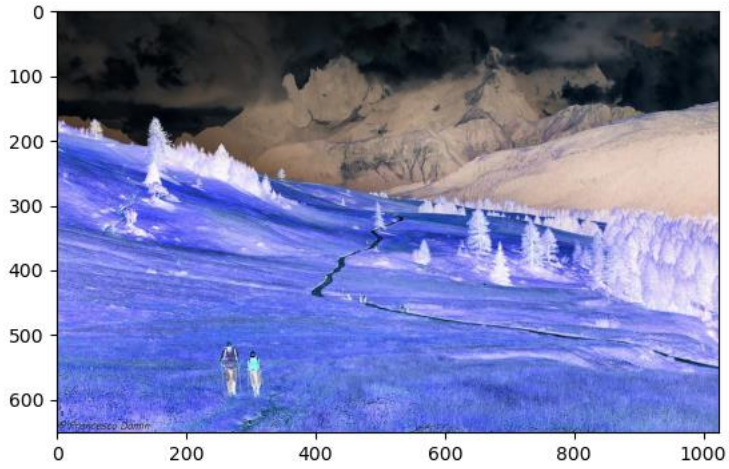
```
Choose an effect:
0 - grayscale
1 - sepia
2 - negative
3 - change brightness
effect: 1
Choose a gradient direction:
0 - no gradient
1 - from center
2 - to center
3 - diagonal
gradient direction: 3
Enter sepia depth (recommended range 10-60): 80
Image saved as output_picture.jpg
```

*Сепія; градієнт діагональ; depth = 30:*

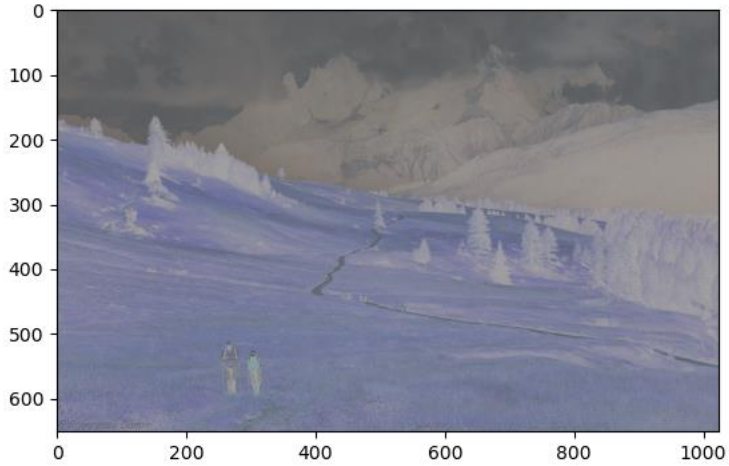
```
Choose an effect:
0 - grayscale
1 - sepia
2 - negative
3 - change brightness
effect: 1
Choose a gradient direction:
0 - no gradient
1 - from center
2 - to center
3 - diagonal
gradient direction: 3
Enter sepia depth (recommended range 10-60): 30
Image saved as output_picture.jpg
```

*Негатив; без градієнту; ітенсивність максимальна:*

```
Choose an effect:
0 - grayscale
1 - sepia
2 - negative
3 - change brightness
```

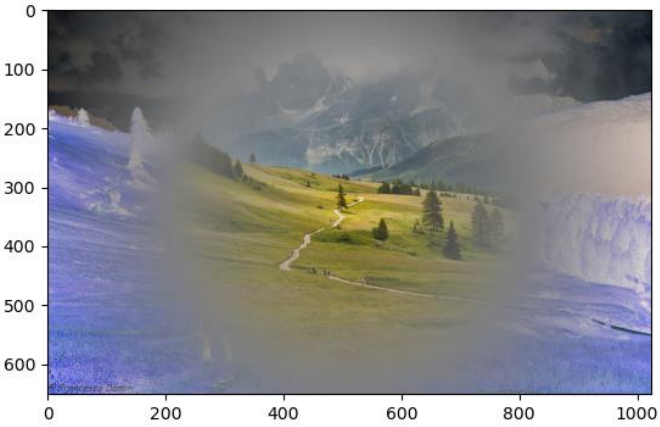


*Негатив; без градієнту; інтенсивність 0.8:*



```
Choose an effect:
0 - grayscale
1 - sepia
2 - negative
3 - change brightness
effect: 2
Choose a gradient direction:
0 - no gradient
1 - from center
2 - to center
3 - diagonal
gradient direction: 0
Enter effect intensity (0.0 to 1.0): 0.8
Image saved as output_picture.jpg
```

*Негатив; градієнт до центру; інтенсивність максимальна:*

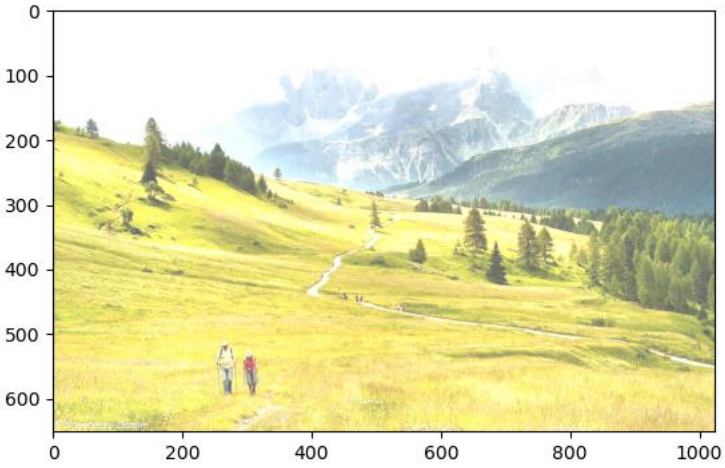


```
Choose an effect:
0 - grayscale
1 - sepia
2 - negative
3 - change brightness
effect: 2
Choose a gradient direction:
0 - no gradient
1 - from center
2 - to center
3 - diagonal
gradient direction: 2
Enter effect intensity (0.0 to 1.0): 1
Image saved as output_picture.jpg
```

*Зміна яскравості (-100); без градієнту:*


```
Choose an effect:
0 - grayscale
1 - sepia
2 - negative
3 - change brightness
effect: 3
Choose a gradient direction:
0 - no gradient
1 - from center
2 - to center
3 - diagonal
gradient direction: 0
Enter brightness factor (-100 to 100): -100
Image saved as output_picture.jpg
```

*Зміна яскравості (+100); без градієнту:*




```
Choose an effect:
0 - grayscale
1 - sepia
2 - negative
3 - change brightness
effect: 3
Choose a gradient direction:
0 - no gradient
1 - from center
2 - to center
3 - diagonal
gradient direction: 0
Enter brightness factor (-100 to 100): 100
Image saved as output_picture.jpg
```

**Програмний код, що забезпечує отримання результату:**

```python
import numpy as np
from PIL import Image, ImageDraw
from matplotlib import pyplot as plt

def image_read(file_name: str):
    image = Image.open(file_name)
```

```
    draw = ImageDraw.Draw(image)
    width, height = image.size
    pix = image.load()
    return {"image_file": image, "image_draw": draw, "image_width": width,
"image_height": height, "image_pix": pix}

def change_brightness(image_info, factor=100):
    image, draw, width, height, pix = (image_info["image_file"],
image_info["image_draw"],
                                       image_info["image_width"],
image_info["image_height"], image_info["image_pix"])

    for i in range(width):
        for j in range(height):
            r, g, b = pix[i, j][:3]
            r = min(255, max(0, r + factor))
            g = min(255, max(0, g + factor))
            b = min(255, max(0, b + factor))
            draw.point((i, j), (r, g, b))

def apply_sepia(image_info, depth=30):
    image, draw, width, height, pix = (image_info["image_file"],
image_info["image_draw"],
                                       image_info["image_width"],
image_info["image_height"], image_info["image_pix"])
    for i in range(width):
        for j in range(height):
            r, g, b = pix[i, j][:3]
            gray = (r + g + b) // 3
            r_sepia = min(255, gray + depth * 2)
            g_sepia = min(255, gray + depth)
            b_sepia = min(255, gray)
            draw.point((i, j), (r_sepia, g_sepia, b_sepia))

def apply_grayscale(image_info, intensity=1.0):
    image, draw, width, height, pix = (image_info["image_file"],
image_info["image_draw"],
                                       image_info["image_width"],
image_info["image_height"], image_info["image_pix"])
    for i in range(width):
        for j in range(height):
            r, g, b = pix[i, j][:3]
            gray = (r + g + b) // 3
            r = int((1 - intensity) * r + intensity * gray)
            g = int((1 - intensity) * g + intensity * gray)
            b = int((1 - intensity) * b + intensity * gray)
            draw.point((i, j), (r, g, b))

def apply_negative(image_info, intensity=1.0):
    image, draw, width, height, pix = (image_info["image_file"],
image_info["image_draw"],
                                       image_info["image_width"],
image_info["image_height"], image_info["image_pix"])

    for i in range(width):
        for j in range(height):
            r, g, b = pix[i, j][:3]
            intensity_scaled = intensity ** 2  # Squaring the intensity for non-
linear blending
```

```
            r_neg = 255 - r
            g_neg = 255 - g
            b_neg = 255 - b
            r = int((1 - intensity_scaled) * r + intensity_scaled * r_neg)
            g = int((1 - intensity_scaled) * g + intensity_scaled * g_neg)
            b = int((1 - intensity_scaled) * b + intensity_scaled * b_neg)
            draw.point((i, j), (r, g, b))

def apply_gradient(image_info, effect_func, direction):
    image = image_info["image_file"]
    width, height = image_info["image_width"], image_info["image_height"]

    if direction == "none":
        effect_func(image_info)
        return

    mask = np.zeros((width, height), dtype=np.float32)

    if direction == "diagonal":
        for i in range(width):
            for j in range(height):
                mask[i, j] = (i + j) / (width + height)
    elif direction == "to_center":
        center_x, center_y = width // 2, height // 2
        for i in range(width):
            for j in range(height):
                distance = np.sqrt((i - center_x) ** 2 + (j - center_y) ** 2)
                max_distance = np.sqrt(center_x ** 2 + center_y ** 2)
                mask[i, j] = distance / max_distance
    elif direction == "from_center":
        center_x, center_y = width // 2, height // 2
        for i in range(width):
            for j in range(height):
                distance = np.sqrt((i - center_x) ** 2 + (j - center_y) ** 2)
                max_distance = np.sqrt(center_x ** 2 + center_y ** 2)
                mask[i, j] = 1 - (distance / max_distance)

    mask = np.clip(mask, 0, 1)

    effect_image = image.copy()
    effect_info = {"image_file": effect_image, "image_draw":
ImageDraw.Draw(effect_image),
                   "image_width": width, "image_height": height, "image_pix":
effect_image.load()}

    effect_func(effect_info)

    for i in range(width):
        for j in range(height):
            original_pixel = image_info["image_pix"][i, j][:3]
            effect_pixel = effect_info["image_pix"][i, j][:3]
            blended_pixel = tuple(
                int(mask[i, j] * e + (1 - mask[i, j]) * o) for e, o in
zip(effect_pixel, original_pixel))
            image_info["image_draw"].point((i, j), blended_pixel)

if __name__ == "__main__":
    file_name_start = 'input_picture.jpg'
    file_name_stop = 'output_picture.jpg'
```

```
    print('Choose an effect:')
    print('0 - grayscale')
    print('1 - sepia')
    print('2 - negative')
    print('3 - change brightness')
    mode = int(input('effect: '))

    print('Choose a gradient direction:')
    print('0 - no gradient')
    print('1 - from center')
    print('2 - to center')
    print('3 - diagonal')
    gradient_direction = int(input('gradient direction: '))

    direction_mapping = {0: "none", 1: "from_center", 2: "to_center", 3:
"diagonal"}
    direction = direction_mapping.get(gradient_direction, "diagonal")

    image_info = image_read(file_name_start)

    if mode == 1:  # Sepia
        depth = int(input('Enter sepia depth (recommended range 10-60): '))
        apply_gradient(image_info, lambda img: apply_sepia(img, depth),
direction=direction)
    elif mode == 0 or mode == 2: # Grayscale or Negative
        intensity = float(input('Enter effect intensity (0.0 to 1.0): '))
        if mode == 0:
            apply_gradient(image_info, lambda img: apply_grayscale(img,
intensity), direction=direction)
        else:
            apply_gradient(image_info, lambda img: apply_negative(img,
intensity), direction=direction)
    elif mode == 3: # Brightness
        brightness_factor = int(input('Enter brightness factor (-100 to 100):
'))
        apply_gradient(image_info, lambda img: change_brightness(img,
brightness_factor), direction=direction)

    plt.imshow(image_info["image_file"])
    plt.show()
    image_info["image_file"].save(file_name_stop)
    print(f'Image saved as {file_name_stop}')
```

**Висновки:** Виконавши цю лабораторну роботу, я ознайомилась із математичними моделями різних ефектів для корекції характеристик кольору окремих растрів цифрових зображень.

Також я ознайомилась із принципом застосування цих ефектів не рівномірно, а в різних напрямках за допомогою масок градієнтів та технології змішування обробленого та необробленого зображення - для цього також було проведене ознайомлення із математичними моделями таких градієнтів.

В процесі роботи також знайшлись декілька альтернативних моделей, але вони були відкинуті за суб'єктивним сприйняттям їх як менш зручних.

***В результаті програма працює коректно та дає змогу не тільки обирати ефекти та напрямки градієнту, але й регулювати параметри ефектів***

***(інтенсивність/глибина/фактор яскравості). Всі картинки відображаються після редагування за допомогою matplotlib, а також зберігаються в окремий файл.***