**Міністерство освіти і науки України**

**Національний технічний університет України «КПІ» імені Ігоря Сікорського**

**Кафедра обчислювальної техніки ФІОТ**

**ЗВІТ**

**з лабораторної роботи No2**

**з навчальної дисципліни «Технології Computer Vision»**

**Тема:**

**ДОСЛІДЖЕННЯ АЛГОРИТМІВ ФОРМУВАННЯ ТА ОБРОБКИ РАСТРОВИХ ЦИФРОВИХ ЗОБРАЖЕНЬ**

**Виконала:**

Студентка 3 курсу кафедри ОТ ФІОТ,

Навчальної групи ІМ-24

Іванова Дар'я

**Перевірив:**

Професор кафедри ОТ ФІОТ

Олексій Писарчук

**Київ 2024**

**І. Мета роботи:**

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.

**ІІ. Завдання:**

Завдання ІІІ рівня – максимально 9 балів.

Лабораторія провідної ІТ-компанії реалізує масштабний проект розробки універсальної платформи з цифрової обробки зображень для задач Computer Vision. Платформа передбачає розташування back-end компоненти на власному хмарному сервері з наданням повноважень користувачам заздалегідь адаптованого front-end функціоналу універсальної платформи. Цим формується унікальна для потреб замовника ERP

система з технологіями Computer Vision Замовниками ресурсів платформи є: державні та комерційні компанії, що розробляють медичне обладнання з діагностування захворювань за візуальною інформацією; автоматизації аграрного бізнесу в

аспекті обліку посівних територій за даними з БПЛА; візуального контролю безпекових заходів на об’єктах критичної інфраструктури: аеропорти, торгівельно-розважальні центри, житлові комплекси тощо.

|  |
| --- |
| Завдання ІІІ 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: |

Завдання ІІІ рівня – максимально 9 балів.

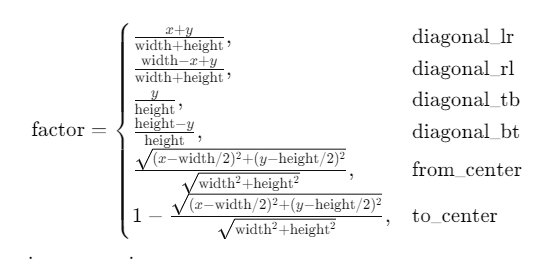
Реалізувати розробку програмного скрипта, що реалізує корекцію кольору цифрового растрового зображення з переліку: зміна яскравості, відтінки сірого, негатив, серпія – в градієнтах: діагональ (будь-який напрям); від центру; до центра. Обробку реалізувати на рівні матриці растра. Зображення обрати самостійно.

### ІІІ. Результати виконання лабораторної роботи. **5.3. Результати виконання лабораторної роботи**

**5.3.1. Синтезована математична модель**

Синтезована математична модель обробки зображень базується на наступних перетвореннях

* **Градаційне перетворення (градієнт):** Формула для обчислення коефіцієнта градієнту залежно від вибраного напрямку:



* **Корекція яскравості:**

I′(x,y)=min(max(I(x,y)+B,0),255) де B — коефіцієнт яскравості.

* **Фільтрація:**

Для ефектів типу "контур", "згладжування" застосовуються бібліотечні функції з PIL.ImageFilter.

**5.3.2. Результати архітектурного проектування та їх опис**

**Архітектурна структура проекту:**  
Проект складається з наступних основних компонентів:

1. **Модуль завантаження зображення (load\_image)**
   * Перевіряє існування файлу, завантажує зображення у вигляді масиву numpy.
2. **Модуль збереження зображення (save\_image)**
   * Зберігає оброблене зображення у вказаному форматі.
3. **Модуль обробки ефектів (apply\_grayscale, apply\_negative, apply\_sepia, apply\_brightness)**
   * Використовуються різні методи корекції кольору, включаючи градієнтні ефекти.
4. **Модуль вибору параметрів (select\_option)**
   * Реалізує інтерактивний вибір ефектів користувачем.
5. **Головний модуль (main)**
   * Об'єднує всі частини проекту, обробляє аргументи командного рядка та викликає відповідні функції.

.

**5.3.3. Опис структури проекту програми**

Структура проекту складається з таких компонентів:

* **Основний файл transformation.py**:

Lab\_work\_2/

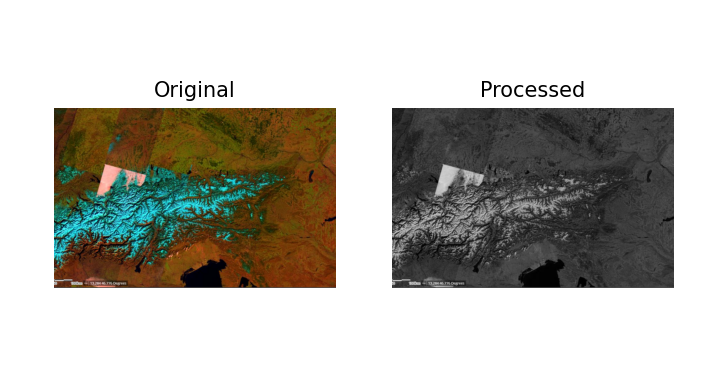
│-- lab2.py # Основний скрипт програми

│-- image.png # Вхідне зображення

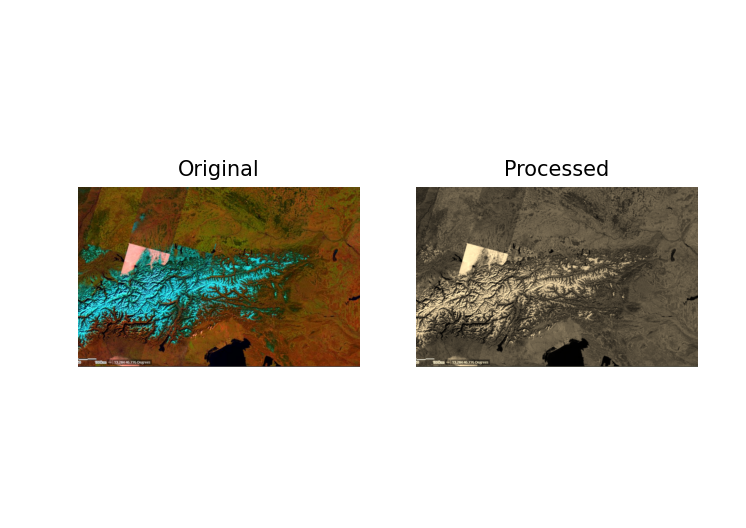
│-- result.png # Оброблене зображення

**5.3.4. Результати роботи програми відповідно до завдання**

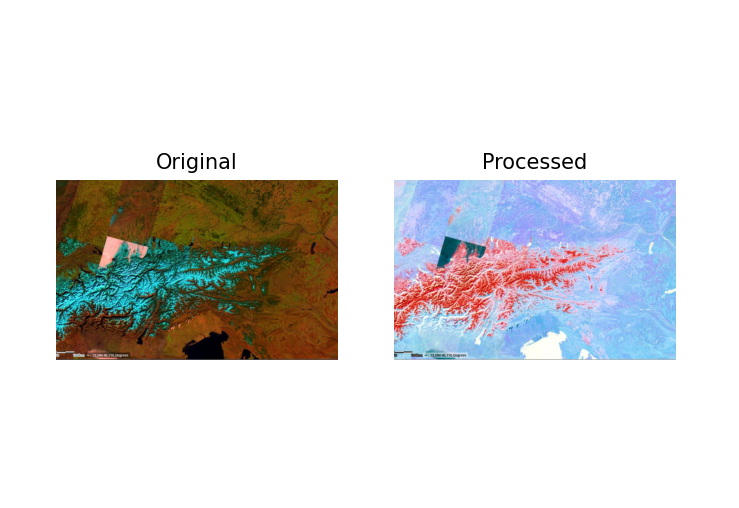
0 - grayscale



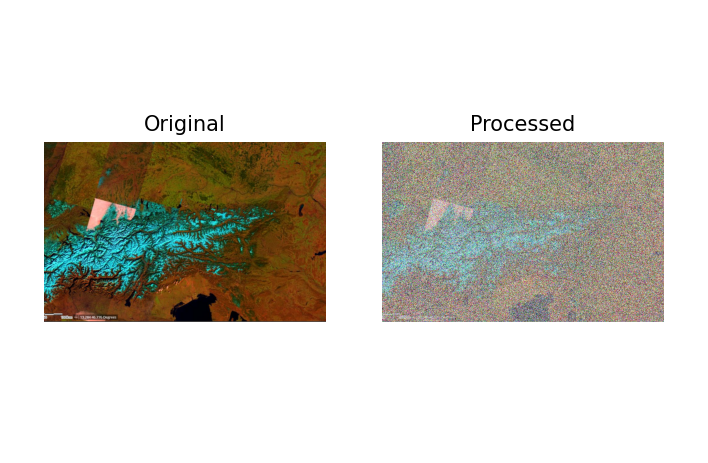
1 - sepia



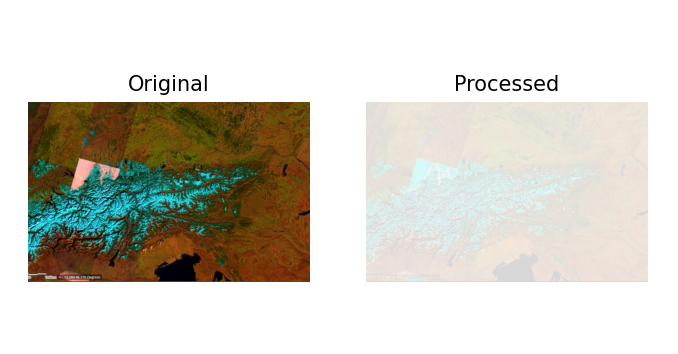
2 - negative



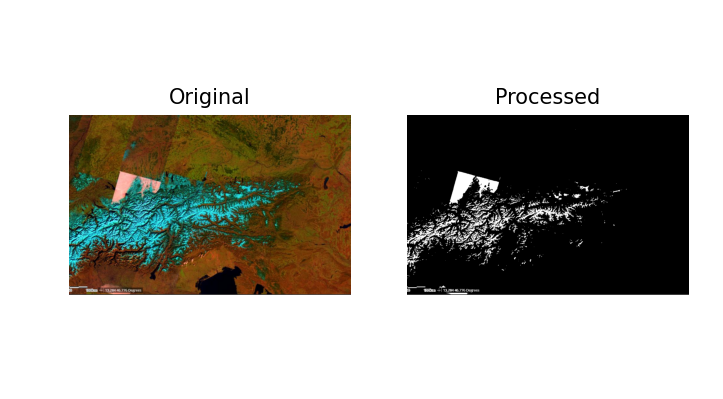
3 - noise



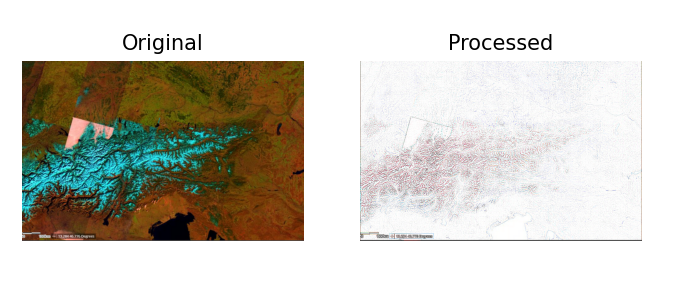
4 - brightness



5 - monochrome

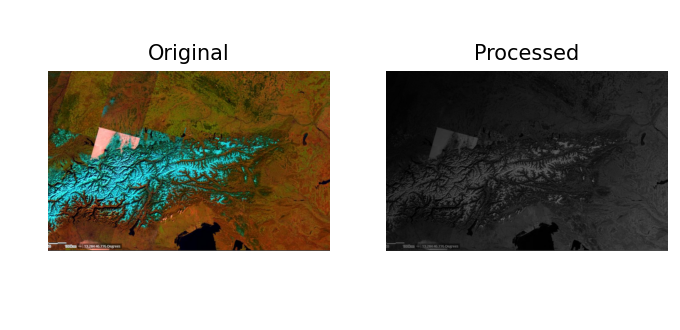


6 - contour

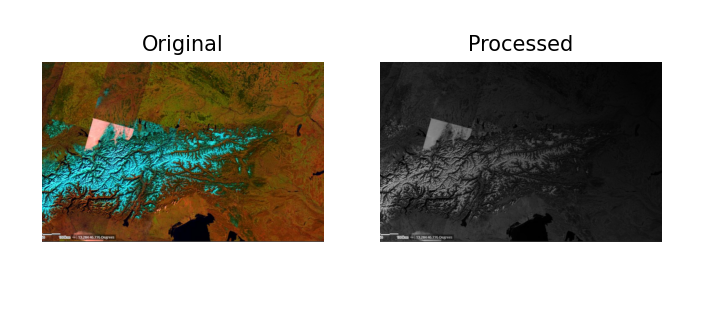


7 - gradient

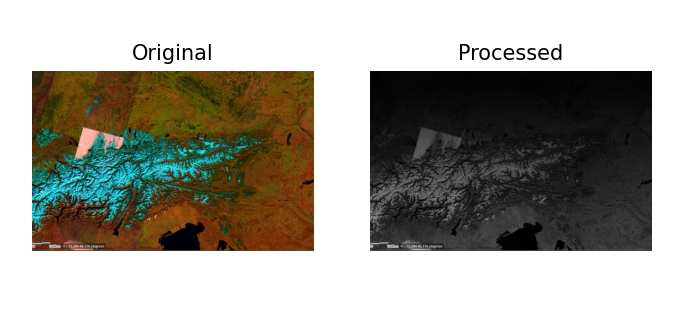
0 - diagonal\_lr



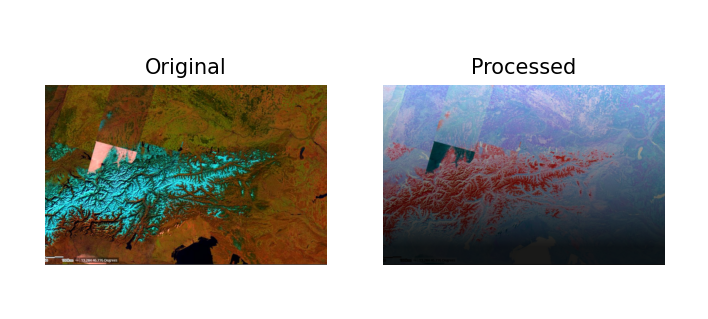
1 - diagonal\_rl



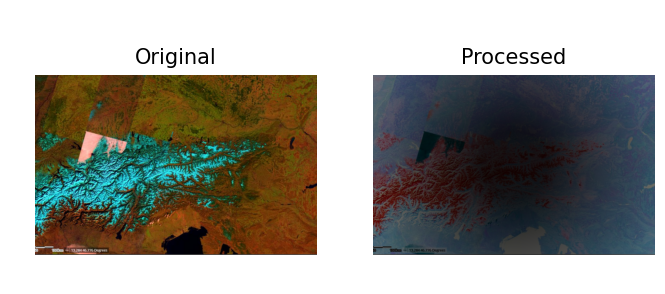
2 - diagonal\_tb



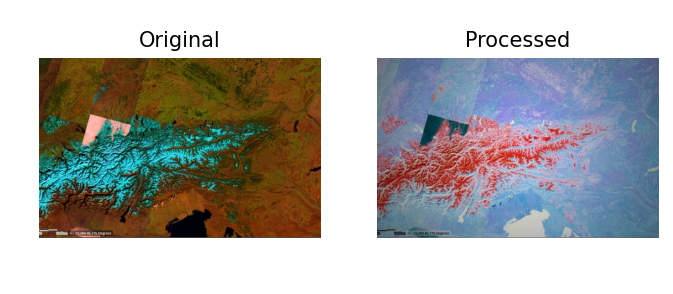
3 - diagonal\_bt



4 - from\_center



5 - to\_center



**5.3.5. Програмний код, що забезпечує отримання результату**

*import* os  
*import* numpy *as* np  
*from* PIL *import* Image, ImageDraw, ImageFilter  
*import* argparse  
*import* matplotlib.pyplot *as* plt  
*from* PIL.ImageFilter *import* (  
 BLUR, CONTOUR, DETAIL, EDGE\_ENHANCE, EDGE\_ENHANCE\_MORE,  
 EMBOSS, FIND\_EDGES, SMOOTH, SMOOTH\_MORE, SHARPEN)  
  
  
*#input\_path = r"C:\Users\Даша\OneDrive\Документы\GitHub\computer\_vision\_1\Lab\_work\_2\image.png"  
#output\_path = r"C:\Users\Даша\OneDrive\Документы\GitHub\computer\_vision\_1\Lab\_work\_2\result.png"  
  
def* load\_image(image\_path):  
 *if not* os.path.exists(image\_path):  
 print(f"Помилка: Файл {image\_path} не знайдено!")  
 exit()  
 *return* np.array(Image.open(image\_path))  
  
  
  
*def* save\_image(image\_array, output\_path, format="PNG"):  
 output\_image = Image.fromarray(image\_array.clip(0, 255).astype(np.uint8))  
 output\_image.save(output\_path, format=format)  
  
  
*def* get\_gradient\_factor(x, y, width, height, gradient\_type):  
 factor = 1.0  
 *if* gradient\_type == 'diagonal\_lr':  
 factor = (x + y) / (width + height)  
 *elif* gradient\_type == 'diagonal\_rl':  
 factor = (width - x + y) / (width + height)  
 *elif* gradient\_type == 'diagonal\_tb':  
 factor = y / height  
 *elif* gradient\_type == 'diagonal\_bt':  
 factor = (height - y) / height  
 *elif* gradient\_type == 'from\_center':  
 factor = ((x - width // 2) \*\* 2 + (y - height // 2) \*\* 2) \*\* 0.5 / ((width \*\* 2 + height \*\* 2) \*\* 0.5)  
 *elif* gradient\_type == 'to\_center':  
 factor = 1 - ((x - width // 2) \*\* 2 + (y - height // 2) \*\* 2) \*\* 0.5 / ((width \*\* 2 + height \*\* 2) \*\* 0.5)  
 *return* max(0.1, min(1.0, factor))  
  
  
  
*def* select\_option(options, prompt):  
 print(prompt)  
 *for* key, value *in* options.items():  
 print(f"{key} - {value}")  
 choice = input("Введіть номер: ").strip()  
 *while* choice *not in* options:  
 print("Невірний вибір, спробуйте ще раз.")  
 choice = input("Введіть номер: ").strip()  
 *return* options[choice]  
  
  
*def* apply\_grayscale(image, gradient\_type):  
 image\_pil = Image.fromarray(image).convert('RGB') *# Конвертація для редагування* pix = image\_pil.load()  
 width, height = image\_pil.size  
 gray\_image = Image.new("RGB", (width, height))  
  
 *for* i *in* range(width):  
 *for* j *in* range(height):  
 r, g, b = pix[i, j]  
 gray = (r + g + b) // 3  
 factor = get\_gradient\_factor(i, j, width, height, gradient\_type)  
 new\_gray = int(gray \* factor)  
 gray\_image.putpixel((i, j), (new\_gray, new\_gray, new\_gray))  
  
 *return* np.array(gray\_image)  
  
  
  
*def* apply\_negative(image, gradient\_type):  
 image\_pil = Image.fromarray(image).convert('RGB') *# Конвертація для можливості редагування* pix = image\_pil.load()  
 width, height = image\_pil.size  
 *for* i *in* range(width):  
 *for* j *in* range(height):  
 r, g, b = pix[i, j]  
 factor = get\_gradient\_factor(i, j, width, height, gradient\_type)  
 pix[i, j] = (  
 int((255 - r) \* factor),  
 int((255 - g) \* factor),  
 int((255 - b) \* factor)  
 )  
 *return* np.array(image\_pil)  
  
  
*def* apply\_monochrome(image, gradient\_type):  
 image\_pil = Image.fromarray(image).convert('RGB')  
 pix = image\_pil.load()  
 width, height = image\_pil.size  
 monochrome\_image = Image.new("RGB", (width, height))  
  
 *for* i *in* range(width):  
 *for* j *in* range(height):  
 r, g, b = pix[i, j]  
 gray = (r + g + b) // 3  
 threshold = 128 *# Фіксований поріг для чорно-білого зображення* factor = get\_gradient\_factor(i, j, width, height, gradient\_type)  
 *if* gray \* factor > threshold:  
 monochrome\_image.putpixel((i, j), (255, 255, 255))  
 *else*:  
 monochrome\_image.putpixel((i, j), (0, 0, 0))  
  
 *return* np.array(monochrome\_image)  
  
  
*def* apply\_noise(image):  
 noise\_level = 300 *# Регульований рівень шуму* noisy\_image = image.copy()  
 height, width, channels = image.shape  
 noise = np.random.randint(-noise\_level, noise\_level, (height, width, channels))  
 noisy\_image = np.clip(image + noise, 0, 255)  
 *return* noisy\_image.astype(np.uint8)  
  
*def* apply\_sepia(image, gradient\_type):  
 image\_pil = Image.fromarray(image).convert('RGB')  
 pix = image\_pil.load()  
 width, height = image\_pil.size  
 sepia\_image = Image.new("RGB", (width, height))  
  
 *for* i *in* range(width):  
 *for* j *in* range(height):  
 r, g, b = pix[i, j]  
 factor = get\_gradient\_factor(i, j, width, height, gradient\_type)  
 tr = int((0.393 \* r + 0.769 \* g + 0.189 \* b) \* factor)  
 tg = int((0.349 \* r + 0.686 \* g + 0.168 \* b) \* factor)  
 tb = int((0.272 \* r + 0.534 \* g + 0.131 \* b) \* factor)  
 sepia\_image.putpixel((i, j), (min(255, tr), min(255, tg), min(255, tb)))  
  
 *return* np.array(sepia\_image)  
  
  
  
*def* apply\_brightness(image, gradient\_type, factor):  
 *return* np.clip(image + factor, 0, 255).astype(np.uint8)  
  
  
*def* apply\_filter(image, filter\_type):  
 image\_pil = Image.fromarray(image)  
 *if* filter\_type == "contour":  
 image\_pil = image\_pil.filter(ImageFilter.CONTOUR)  
 *elif* filter\_type == "sharpen":  
 image\_pil = image\_pil.filter(ImageFilter.SHARPEN)  
 *return* np.array(image\_pil)  
  
*def* main():  
 parser = argparse.ArgumentParser(description='Image color correction script.')  
 parser.add\_argument('--input', required=*True*, help='Path to input image')  
 parser.add\_argument('--output', required=*True*, help='Path to save processed image')  
 args = parser.parse\_args()  
  
 image = load\_image(args.input)  
  
 effects = select\_option({  
 "0": "grayscale",  
 "1": "sepia",  
 "2": "negative",  
 "3": "noise",  
 "4": "brightness",  
 "5": "monochrome",  
 "6": "contour",  
 "7": "gradient"  
 }, "Оберіть тип перетворення!")  
  
 brightness\_factor = 0  
 gradient\_type = ""  
  
 *if* effects == "brightness":  
 brightness\_factor = int(input("Введіть фактор яскравості (-100 до 100): "))  
  
 *if* effects == "gradient":  
 gradient\_type = select\_option({  
 "0": "diagonal\_lr",  
 "1": "diagonal\_rl",  
 "2": "diagonal\_tb",  
 "3": "diagonal\_bt",  
 "4": "from\_center",  
 "5": "to\_center"  
 }, "Оберіть тип градієнта!")  
 effects = select\_option({  
 "0": "grayscale",  
 "1": "sepia",  
 "2": "negative"  
 }, "Оберіть ефект для застосування з градієнтом:")  
  
 *if* effects == "grayscale":  
 result = apply\_grayscale(image, gradient\_type)  
 *elif* effects == "negative":  
 result = apply\_negative(image, gradient\_type)  
 *elif* effects == "sepia":  
 result = apply\_sepia(image, gradient\_type)  
 *elif* effects == "brightness":  
 result = apply\_brightness(image, gradient\_type, brightness\_factor)  
 *elif* effects == "contour":  
 result = apply\_filter(image, "contour")  
 *elif* effects == "monochrome":  
 result = apply\_monochrome(image, gradient\_type)  
 *elif* effects == "noise":  
 result = apply\_noise(image)  
 *else*:  
 print("Невідомий ефект!")  
 *return* save\_image(result, args.output)  
 print(f"Processed image saved at {args.output}")  
  
 plt.subplot(1, 2, 1)  
 plt.title('Original')  
 plt.imshow(Image.open(args.input))  
 plt.axis('off')  
  
 plt.subplot(1, 2, 2)  
 plt.title('Processed')  
 plt.imshow(result)  
 plt.axis('off')  
  
 plt.show()  
  
*if* \_\_name\_\_ == '\_\_main\_\_':  
 main()

**5.4. Висновки**

У ході виконання лабораторної роботи було реалізовано програмний скрипт для корекції кольору цифрового растрового зображення.