

Algebra Projekt 3

June 14, 2023

Paweł Pozorski i Hubert Sobociński

Transformacje przestrzenne są kluczowym elementem grafiki komputerowej. Wykorzystuje się je do manipulacji obiektami graficznymi, takimi jak modele 3D czy obrazy, w celu zmiany ich położenia, skali i orientacji w przestrzeni. **Algebra liniowa odgrywa istotną rolę w realizacji tych transformacji.**

Podstawowymi narzędziami algebraicznymi używanymi do transformacji przestrzennych są macierze. Przykładowe operacje, takie jak translacje, skalowanie i obracanie, są reprezentowane przez odpowiednie macierze transformacji. Dzięki temu można wykonywać te operacje na obiektach graficznych za pomocą prostych operacji algebraicznych, takich jak mnożenie macierzy. Podobnie, skalowanie i obrót są reprezentowane przez odpowiednie macierze transformacji. Dzięki temu można skalować i obracać obiekty graficzne w różnych płaszczyznach. Transformacje przestrzenne są niezbędne do wizualizacji obiektów w przestrzeni trójwymiarowej oraz do tworzenia animacji, efektów specjalnych i innych zaawansowanych technik graficznych. Algebra liniowa dostarcza narzędzi, które umożliwiają programistom i grafikom komputerowym manipulowanie obiektami graficznymi w intuicyjny i efektywny sposób.

W naszej pracy przedstawiliśmy przykładowe użycia algebr liniowych w grafice komputerowej pokazując przekształcenia wykonywane na obrazakach, które wykorzystują właśnie narzędzia z algebrą liniową :)

Konwencja w pliku: 1. Kazdy wynik wypluwa - 3 rysunki - od lewej - oryginalny, implementacja używająca bibliotek jako porównanie kontrolne poprawności naszego rozwiązania, nasza implementacja from scratch - 2 - nasze rozwiązanie wtedy wypada 2. filtr_jakis - funkcja zwracająca wprzefiltrowany obraz używając bibliotek, filtr_jakis_ - nasza implementacja. Analogicznie przekształcenie_jakies, przekształcenie_jakies_. Wszelkie funkcje kończące się na "_" są częścią naszej implementacji.

```
[173]: import cv2
import numpy as np
import matplotlib.pyplot as plt
import requests
import warnings
import pywt

from skimage import color

np.random.seed(42)
warnings.simplefilter("ignore")
```

```
[13]: def fetch_image(response):
    res = cv2.imdecode(np.asarray(bytearray(response.content)), cv2.
        IMREAD_COLOR)
    assert type(res) != None, f"Failed downloading from {response.url}"
    return res

def fetch_images(urls):
    return [fetch_image(requests.get(url)) for url in urls]
```

```
[14]: IMAGES_HIGH_RES = fetch_images(
    [
        "https://ithemes.com/wp-content/uploads/2016/10/
        ↵Free-High-Quality-Images-Gratisography.jpg",
        "https://ithemes.com/wp-content/uploads/2016/10/
        ↵Free-High-Quality-Images-Unsplash.jpeg",
        "https://ithemes.com/wp-content/uploads/2016/10/Pixabay.jpg",
        "https://ithemes.com/wp-content/uploads/2016/10/
        ↵Free-High-Quality-Images-Lock-and-Stock-Photos.jpg",
        "https://ithemes.com/wp-content/uploads/2016/10/
        ↵Free-High-Quality-Images-Picography-e1477688691166.jpg",
        "https://iso.500px.com/wp-content/uploads/2014/06/W4A2827-1-3000x2000.
        ↵jpg",
    ]
)
```

```
[15]: def apply_transformation(
    transformation,
    transformation_=None,
    imgs=IMAGES_HIGH_RES,
    cmap="viridis",
    fig_size=8,
    **kwargs
):
    assert 2 <= len(imgs), "Provide at least 2 images"

    if transformation_ is not None:
        fig, axs = plt.subplots(len(imgs), 3, figsize=(fig_size, fig_size))
    else:
        fig, axs = plt.subplots(len(imgs), 2, figsize=(fig_size, fig_size))

    i = 0
    for img in imgs:
        if transformation_ is not None:
            transformed_img_ = transformation_(img, **kwargs)
            transformed_img = transformation(img, **kwargs)
```

```

        axs[i, 0].imshow(img, cmap=cmap)
        axs[i, 0].axis("off")
        axs[i, 1].imshow(transformed_img, cmap=cmap)
        axs[i, 1].axis("off")
        if transformation_ is not None:
            axs[i, 2].imshow(transformed_img_, cmap=cmap)
            axs[i, 2].axis("off")
        i += 1

plt.subplots_adjust(wspace=0.0, hspace=0.0)
plt.show()

def apply_transformation_on_random(
    transformation, transformation_=None, imgs=IMAGES_HIGH_RES, imgs_num=2, □
    **kwargs
):
    assert (
        2 <= imgs_num <= len(imgs)
    ), "Images num must be between 2 and number of provided images"
    idxs = np.random.choice(np.arange(len(imgs)), imgs_num, replace=False)
    choosen_imgs = [imgs[i] for i in idxs]
    return apply_transformation(transformation, transformation_, choosen_imgs, □
    **kwargs)

def show_core_matrix(matrix, title, axis="off"):
    plt.figure(figsize=(6, 6), dpi=150)
    plt.imshow(matrix, interpolation="none", cmap="gray")
    plt.title(title)
    plt.axis(axis)
    plt.colorbar()
    plt.show()

```

1 Filtracja obrazów

1.0.1 Filtracja dolnoprzepustowa

Redukuje wysokie częstotliwości, wygładzając obraz i redukując szumy

```
[16]: def conv2D_(img, kernel):
    kernel_h = kernel.shape[0]
    kernel_w = kernel.shape[1]
    h = kernel_h // 2
    w = kernel_w // 2

    image_pad = np.pad(
```

```

        img, pad_width=((h, h), (w, w)), mode="constant", constant_values=0
    ).astype(np.float32)

    image_conv = np.zeros(image_pad.shape)

    for i in range(h, image_pad.shape[0] - h):
        for j in range(w, image_pad.shape[1] - w):
            x = image_pad[i - h : i - h + kernel_h, j - w : j - w + kernel_w]
            x = x.flatten() * kernel.flatten()
            image_conv[i][j] = x.sum()

    h_end = -h
    w_end = -w

    if h == 0:
        return image_conv[h:, w:w_end]
    if w == 0:
        return image_conv[h:h_end, w:]
    return image_conv[h:h_end, w:w_end]

def map_conv2D_(img, kernel):
    if len(img.shape) == 2:
        return conv2D_(img, kernel)

    img_filtered = np.zeros_like(img, dtype=np.float32)
    for c in range(3):
        img_filtered[:, :, c] = conv2D_(img[:, :, c], kernel)
    return img_filtered.astype(np.uint8)

def gaussian_fiter_(kernel_size, sigma):
    ax = np.linspace(-(kernel_size - 1) / 2.0, (kernel_size - 1) / 2.0, kernel_size)
    xx, yy = np.meshgrid(ax, ax)

    kernel = np.exp(-0.5 * (np.square(xx) + np.square(yy)) / np.square(sigma))

    return kernel / np.sum(kernel)

```

```
[17]: def filtr_dolnoprzepustowy_srednia(img, kernel_size=5):
    kernel = np.ones((kernel_size, kernel_size), np.float32) / (
        kernel_size * kernel_size
    )
    return cv2.filter2D(img, -1, kernel)
```

```
def filtr_dolnoprzepustowy_gauss(img, kernel_size=5, sigma=2.0):
```

```

    return cv2.GaussianBlur(img, (kernel_size, kernel_size), sigma)

def filtr_dolnoprzepustowy_gauss_(img, kernel_size=5, sigma=2.0):
    global gaussian_kernel
    gaussian_kernel = gaussian_fiter_(kernel_size, sigma)
    return map_conv2D_(img, gaussian_kernel)

def filtr_dolnoprzepustowy_srednia_(img, kernel_size=5):
    global average_kernel
    average_kernel = np.ones((kernel_size, kernel_size), np.float32) / (
        kernel_size * kernel_size
    )
    return map_conv2D_(img, average_kernel)

```

[18]:

```

noisy_imgs = fetch_images(
    [
        "https://media.cheggcdn.com/media/8c7/
        ↪8c7f3063-bccd-49ef-a483-c160557d6ef4/phpAtYk4M.png",
        "https://people.math.sc.edu/Burkardt/c_src/image_denoise/balloons_noisy.
        ↪png",
    ]
)

```

[19]:

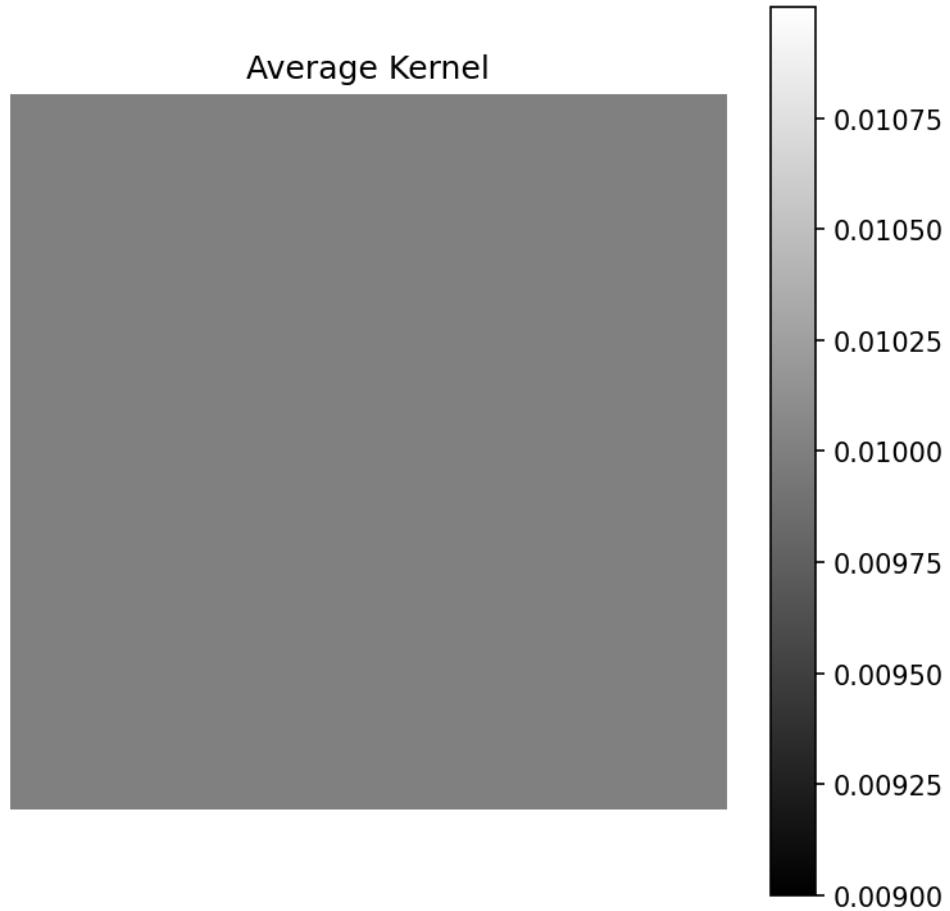
```

# uzywajqc sredniej wartosci pikseli
apply_transformation(
    filtr_dolnoprzepustowy_srednia,
    transformation_=filtr_dolnoprzepustowy_srednia_,
    imgs=noisy_imgs,
    kernel_size=10,
)

```



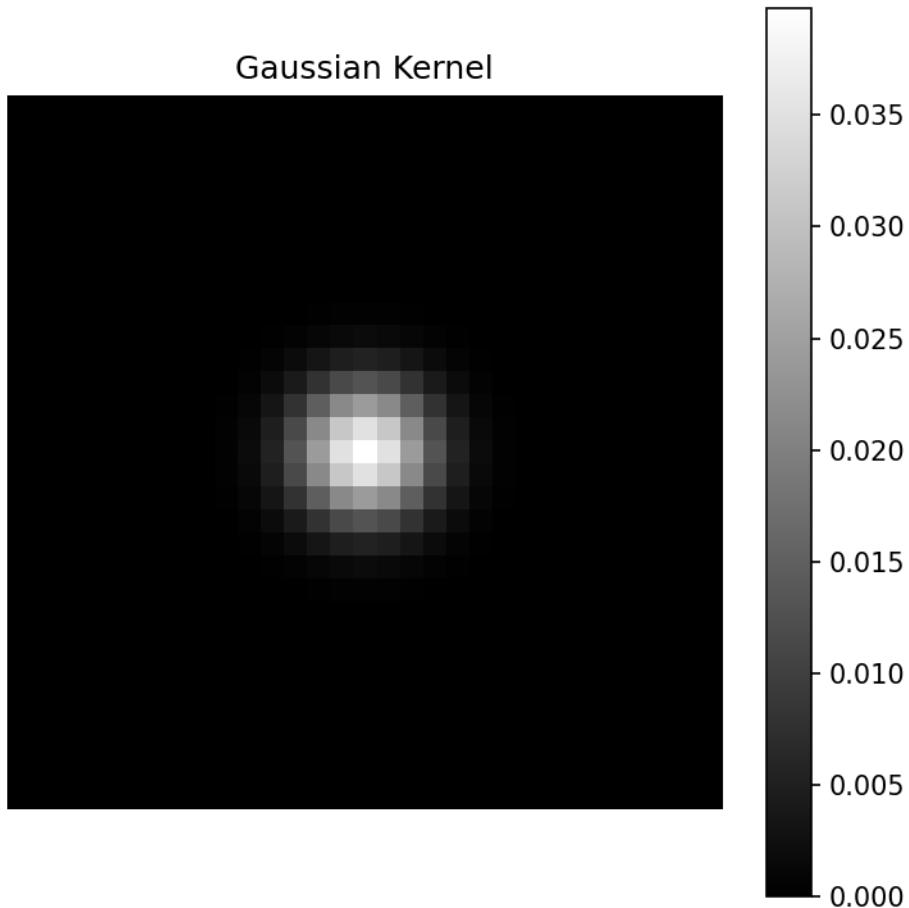
```
[20]: show_core_matrix(average_kernel, "Average Kernel")
```



```
[21]: # uzywajqc gaussian filter
apply_transformation(
    filtr_dolnoprzepustowy_gauss,
    transformation_=filtr_dolnoprzepustowy_gauss_,
    imgs=noisy_imgs,
    kernel_size=31,
)
```



```
[22]: show_core_matrix(gaussian_kernel, "Gaussian Kernel")
```



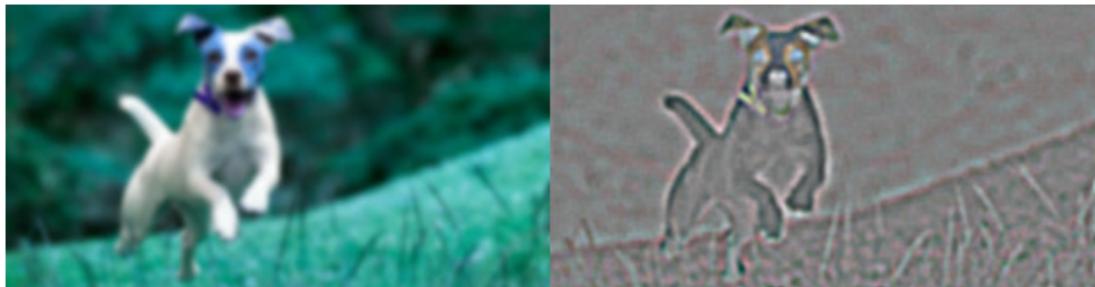
1.0.2 Filtracja górnoprzepustowa

Redukuje niskie częstotliwości, wydobywając krawędzie i detale obrazu.

```
[23]: def high_pass_filter_laplacian(img, kernel_size):
    filtered_img = cv2.Laplacian(img, cv2.CV_64F, ksize=kernel_size)
    return cv2.normalize(filtered_img, None, 0, 255, cv2.NORM_MINMAX, cv2.CV_8U)

[24]: blurred_images = fetch_images(
    [
        "https://images.ctfassets.net/u4vv676b8z52/aLsx9SoEj0jI9nkHuEiB6/
         ↪b128cc239af84fd9b969fc037693df7b/jack-russell-blurred-1200x630.jpg_h_250?
         ↪fm=jpg&q=80",
        "https://encrypted-tbn0.gstatic.com/images?q=tbn:
         ↪ANd9GcTSjNh0l7lhalkb9B1VXL1kfPDKRK5p7Vh6iMTao_wjTHPuSl-mvuqe3AFxLqcinSMkRE4&usqp=CAU",
    ]
)
```

```
[25]: # Laplacian filter
apply_transformation(high_pass_filter_laplacian, imgs=blured_images,
                     kernel_size=11)
```



1.0.3 Filtracja medianowa

Redukuje szумy impulsowe, zastępując wartość piksela medianą w jego otoczeniu.

```
[26]: def median_filter(img, kernel_size):
        return cv2.medianBlur(img, kernel_size)

def median_filter2D_(img, kernel_size):
    for i in range(0, img.shape[0] - kernel_size + 1, kernel_size):
        for j in range(0, img.shape[1] - kernel_size + 1, kernel_size):
            img[i : i + kernel_size, j : j + kernel_size] = np.full(
                [kernel_size, kernel_size],
```

```
        np.median(img[i : i + kernel_size, j : j + kernel_size]),
    )
return img

def median_filter_(img, kernel_size):
    img = img.copy()
    if len(img.shape) == 3:
        for i in range(3):
            img[:, :, i] = median_filter2D_(img[:, :, i], kernel_size)
    else:
        img = median_filter2D_(img, kernel_size)
    return img
```

```
[27]: apply_transformation(
    median_filter,
    transformation_=median_filter_,
    imgs=noisy_imgs,
    kernel_size=5,
)

# cv2 robi to jakos lepiej u nas widac kwadraty wynikajace z wielkosci kernela
```



```
[28]: # cv2 naklada 2 na siebie (troszke lepiej co prawda)
def median_average_filter_(img, kernel_size):
    return filtr_dolnoprzepustowy_gauss_(median_filter_(img, kernel_size), 6)

apply_transformation(
    median_filter,
    transformation_=median_average_filter_,
    imgs=noisy_imgs,
    kernel_size=5,
)
```

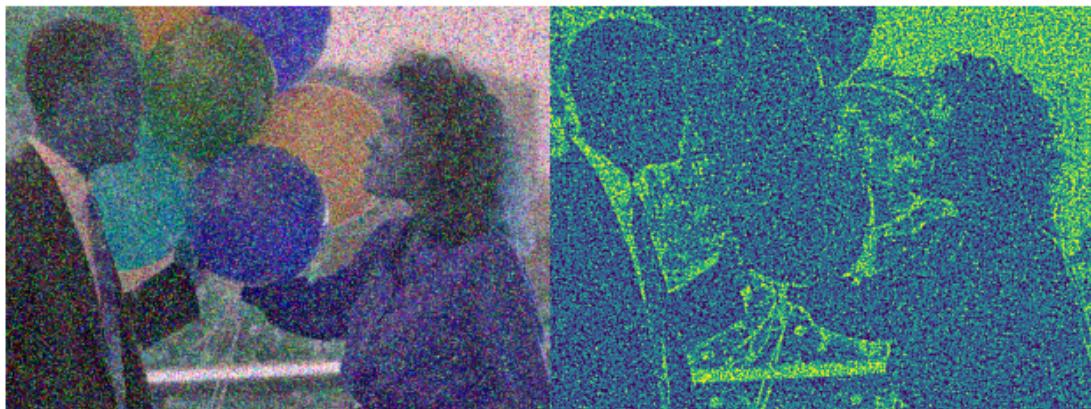


1.0.4 Filtracja adaptacyjna

Dostosowuje parametry filtra do lokalnych cech obrazu, poprawiając jakość w obszarach o zróżnicowanej jasności lub kontraście.

```
[41]: def adaptive_filtration(img, block_size, kernel_size):
    gray_image= cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    return cv2.adaptiveThreshold(gray_image, 255, cv2.ADAPTIVE_THRESH_MEAN_C,
                                 cv2.THRESH_BINARY, block_size, ↴
                                 kernel_size)
```

```
[47]: apply_transformation(
    adaptive_filtration,
    imgs=noisy_imgs,
    block_size = 11,
    kernel_size=2,
)
```



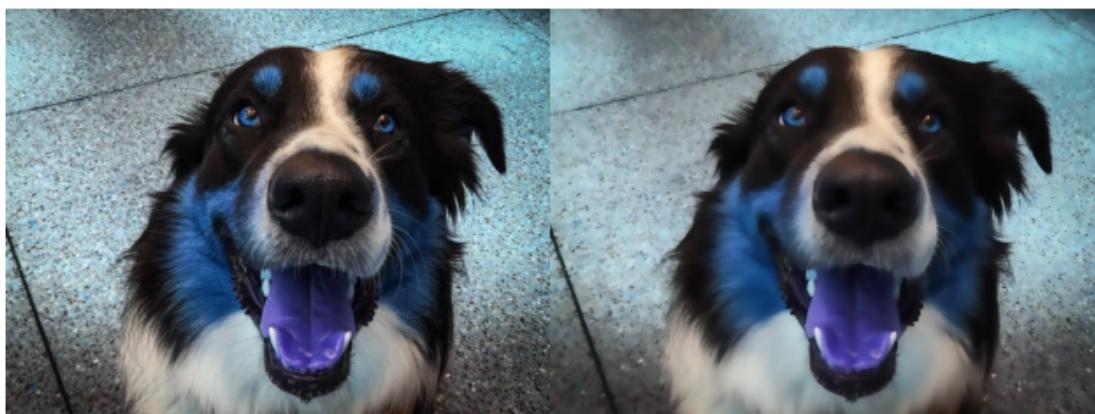
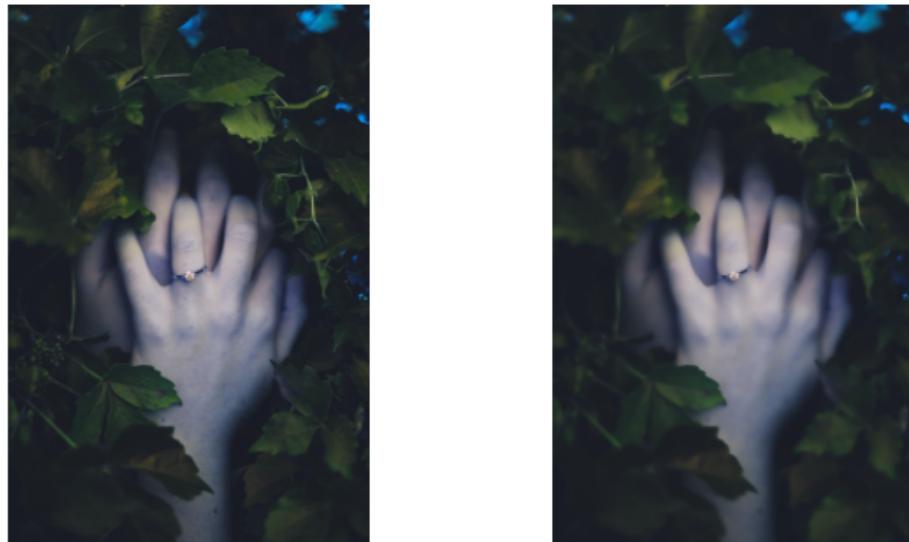
1.0.5 Filtracja bilateralna

Wygładza obraz, zachowując jednocześnie krawędzie i szczegóły poprzez uwzględnienie informacji o intensywności piksela i jego otoczeniu.

- **d:** Średnica obszaru w pikselach, na podstawie którego liczona jest różnica intensywności. Większe wartości dają większe rozmycie.
- **sigma_color:** Odchylenie standardowe w przestrzeni kolorów. Wyższe wartości uwzględniają większą różnicę w kolorach.
- **sigma_space:** Odchylenie standardowe w przestrzeni pikseli. Wyższe wartości uwzględniają większą różnicę przestrzenną.

```
[55]: def bilateral_filter(img, d, sigma_color, sigma_space):  
    filtered_image = cv2.bilateralFilter(img, d, sigma_color, sigma_space)  
    return filtered_image
```

```
[60]: apply_transformation(  
    bilateral_filter,  
    imgs=[IMAGES_HIGH_RES[1], IMAGES_HIGH_RES[3]],  
    d = 50,  
    sigma_color = 100,  
    sigma_space = 5,  
)
```



1.0.6 Filtracja krawędziowa

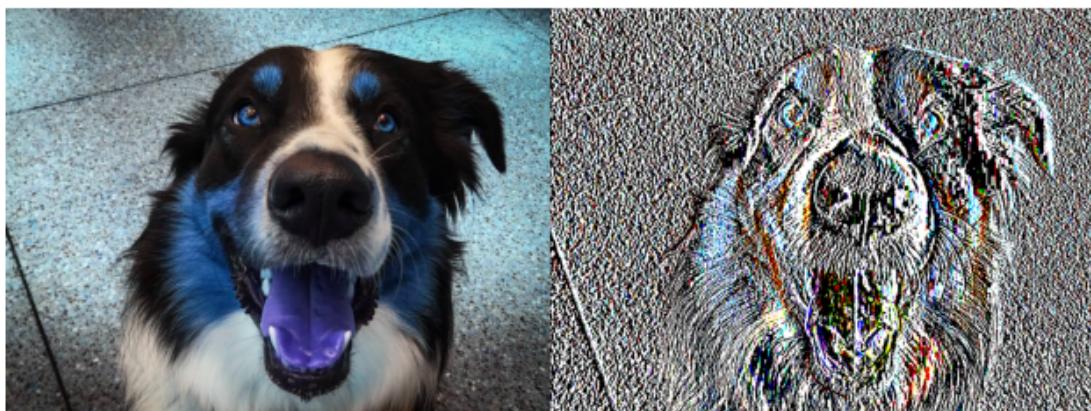
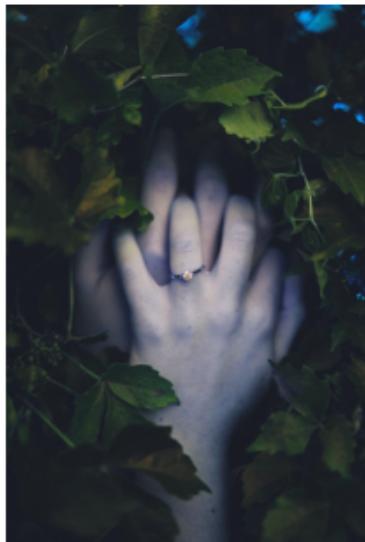
Wyodrębnia krawędzie i detale obrazu poprzez podkreślenie zmian intensywności pikseli wzdłuż krawędzi. Przykłady to operator Sobela i operator Laplace'a.

```
[18]: def edge_filter(img, kernel_size):  
    return cv2.Sobel(img, cv2.CV_64F, 1, 0, kernel_size)
```

```
[19]: apply_transformation(  
    edge_filter,  
    imgs=[IMAGES_HIGH_RES[1], IMAGES_HIGH_RES[3]],  
    kernel_size=5,  
)
```

Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).



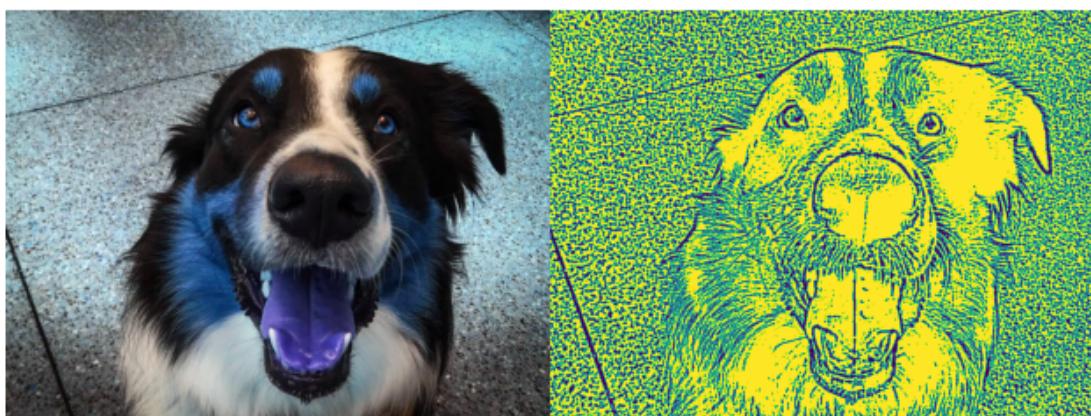
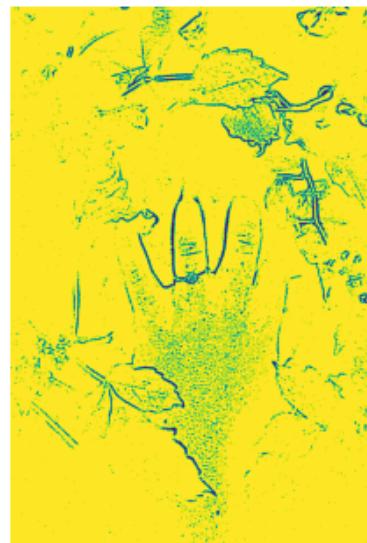
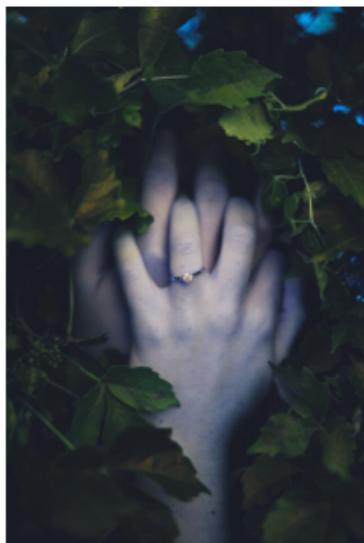
1.0.7 Filtracja adaptacyjna lokalna

Dostosowuje parametry filtra do lokalnych cech obrazu, co pozwala na bardziej precyzyjne filtrowanie w obszarach o zróżnicowanej jasności lub kontraste. Jest skuteczna w redukcji szumów i poprawie jakości obrazu.

- **block_size**: określa rozmiar bloku używanego do obliczania progów adaptacyjnych
- **constant**: określa stałą odejmowaną od średniej wartości w bloku.

```
[79]: def local_adaptive_filtration(img, block_size, kernel_size):
    gray_image = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    return cv2.adaptiveThreshold(gray_image, 255, cv2.ADAPTIVE_THRESH_MEAN_C, cv2.THRESH_BINARY, block_size, kernel_size)
```

```
[80]: apply_transformation(
    local_adaptive_filtration,
    imgs=[IMAGES_HIGH_RES[1], IMAGES_HIGH_RES[3]],
    block_size = 11,
    kernel_size = 5,
)
```

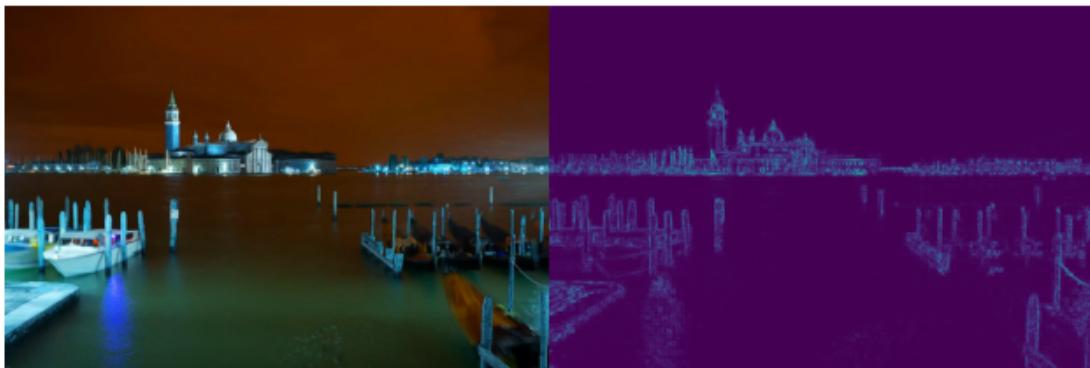


1.0.8 Filtracja tekstur

Wzmocnienie lub redukcja tekstur w obrazie dla podkreślenia lub wygładzenia cech teksturalnych.

```
[69]: def texture_filtration(img):
    gray_image = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    image_filter = cv2.Laplacian(gray_image, cv2.CV_64F)
    return np.uint8(np.absolute(image_filter))
```

```
[76]: apply_transformation(
    texture_filtration,
    imgs=[IMAGES_HIGH_RES[4], IMAGES_HIGH_RES[5]],
)
```



1.0.9 Filtracja anizotropowa

Wygładza obraz, uwzględniając kierunkowość struktur.

```
[82]: def anisotropic_filter(img, iterations=5, k=30, sigma=15):
    # Konwertowanie obrazka do skali szarości
    gray_image = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    # Inicjalizacja macierzy wynikowej
    filtered = np.copy(gray_image).astype(np.float64)

    # Wykonanie filtracji anizotropowej
    for _ in range(iterations):
        gradient_x = cv2.Sobel(filtered, cv2.CV_64F, 1, 0, ksize=3)
        gradient_y = cv2.Sobel(filtered, cv2.CV_64F, 0, 1, ksize=3)

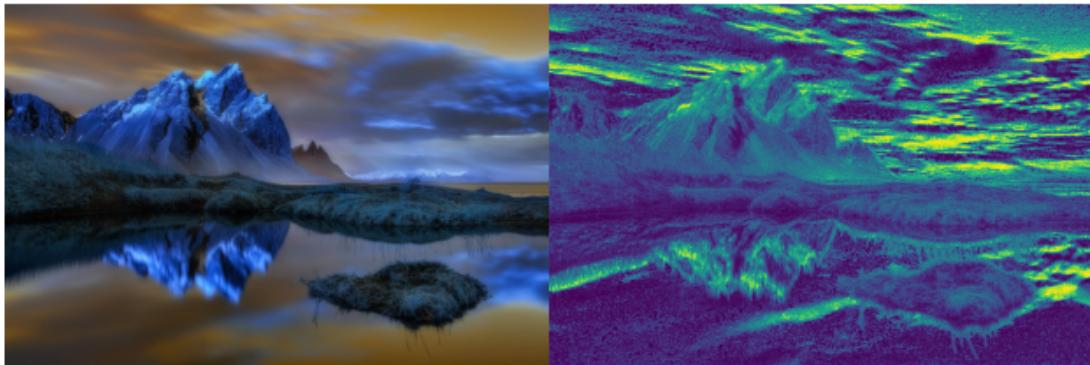
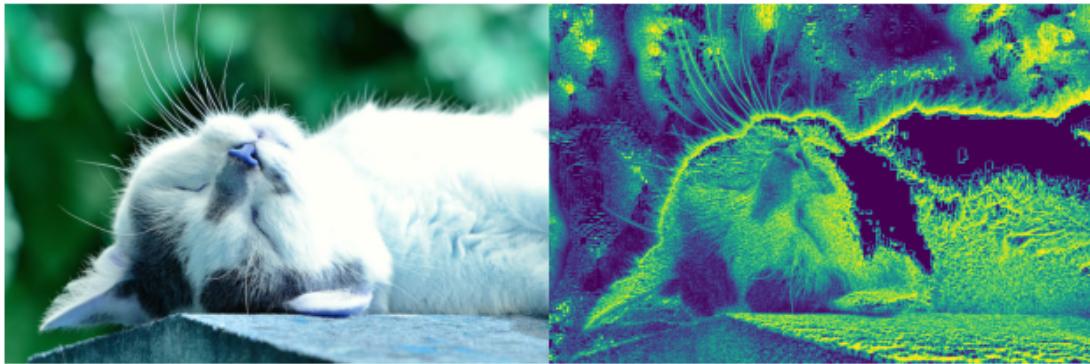
        c = 1.0 / (1.0 + np.square(gradient_x) + np.square(gradient_y) / np.
        ↪square(sigma))

        filtered = (1 - c) * filtered + c * (
            gradient_x * cv2.GaussianBlur(filtered, (3, 3), sigma)) +
            (gradient_y * cv2.GaussianBlur(filtered, (3, 3), sigma))
    )

    filtered = np.clip(filtered, 0, 255).astype(np.uint8)

    return filtered
```

```
[87]: apply_transformation(
    anisotropic_filter,
    imgs=[IMAGES_HIGH_RES[2], IMAGES_HIGH_RES[5]],
)
```



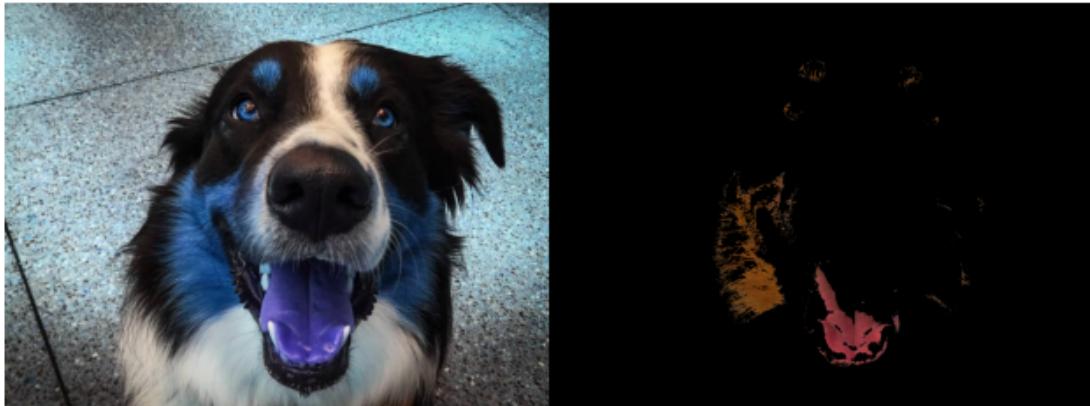
1.0.10 Filtracja kolorowa

Uwzględnia składniki kolorowe obrazu podczas filtracji, wpływając na wygląd kolorów.

```
[93]: def color_filtration(img, color):
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    if color == 'red':
        lower = np.array([0, 0, 100], dtype=np.uint8)
        upper = np.array([80, 80, 255], dtype=np.uint8)
    elif color == 'green':
        lower = np.array([0, 100, 0], dtype=np.uint8)
        upper = np.array([80, 255, 80], dtype=np.uint8)
    elif color == 'blue':
        lower = np.array([100, 0, 0], dtype=np.uint8)
        upper = np.array([255, 80, 80], dtype=np.uint8)
    else:
        print('Unknown color')
        return
```

```
mask = cv2.inRange(img, lower, upper)
return cv2.bitwise_and(img, img, mask=mask)
```

```
[102]: apply_transformation(
    color_filtration,
    imgs=[IMAGES_HIGH_RES[5], IMAGES_HIGH_RES[3]],
    color = "blue"
)
```



2 Przekształcenia geometryczne obrazów

2.0.1 Skalowanie

Zmienia rozmiar obrazu, przy czym metoda interpolacji może wpływać na jakość i dokładność transformacji.

- **scale:** współczynnik skalowania

- **interpolation_method:** wybrana metoda interpolacji, np. cv2.INTER_NEAREST, cv2.INTER_LINEAR, cv2.INTER_CUBIC, cv2.INTER_LANCZOS4

```
[112]: def image_scaling(img, scale, interpolation_method):
    height = int(img.shape[0] * scale)
    width = int(img.shape[1] * scale)
    size = (width, height)
    return cv2.resize(img, size, interpolation = interpolation_method)
```

```
[118]: apply_transformation(
    image_scaling,
    imgs=[IMAGES_HIGH_RES[4], IMAGES_HIGH_RES[2]],
    scale = 0.3,
    interpolation_method = cv2.INTER_CUBIC,
)
```



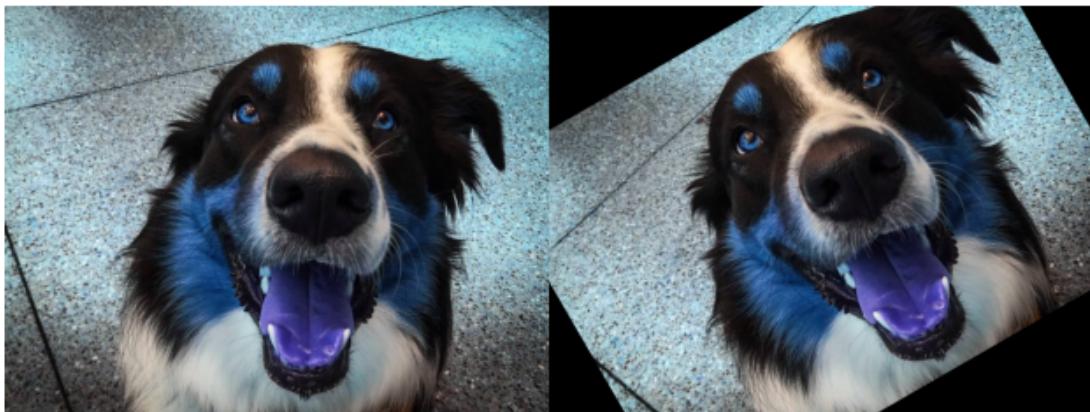
2.0.2 Obrót

Zmienia orientację obrazu, a odpowiednia metoda interpolacji może wpływać na zachowanie szczegółów i jakość obrazu.

```
[34]: from scipy.ndimage import rotate

def rotation_transformation(img, angle):
    return rotate(img, angle, reshape=False)
```

```
[36]: apply_transformation_on_random(
    rotation_transformation,
    imgs=IMAGES_HIGH_RES,
    angle=30,
)
```

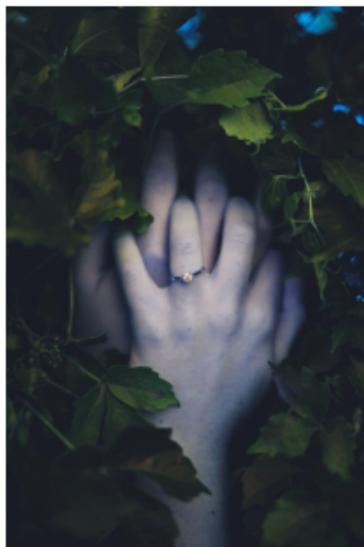
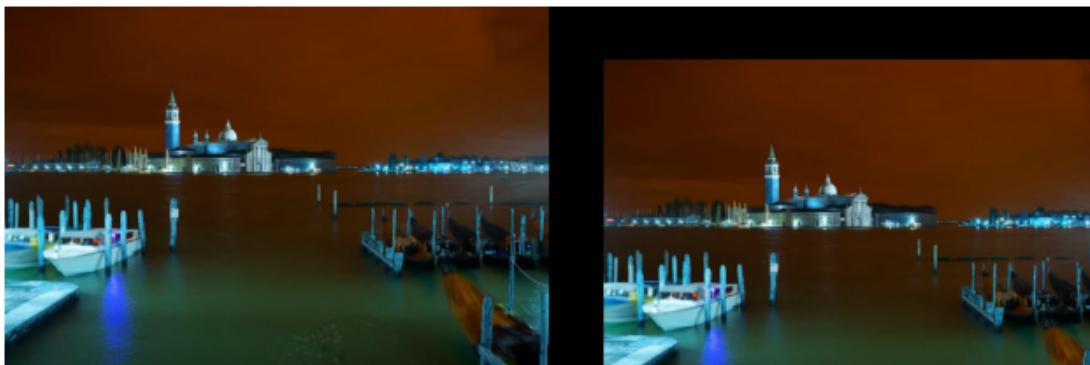


2.0.3 Przesunięcie

Przesuwa obraz względem osi, a interpolacja może wpływać na jakość przesunięcia.

```
[119]: def move_image(img, offset_x, offset_y, interpolation=cv2.INTER_LINEAR):
    height, width = img.shape[:2]
    offset_matrix = np.float32([[1, 0, offset_x], [0, 1, offset_y]])
    return cv2.warpAffine(img, offset_matrix, (width, height), flags=interpolation)
```

```
[122]: apply_transformation_on_random(
    move_image,
    imgs=IMAGES_HIGH_RES,
    offset_x = 100,
    offset_y = 100,
)
```

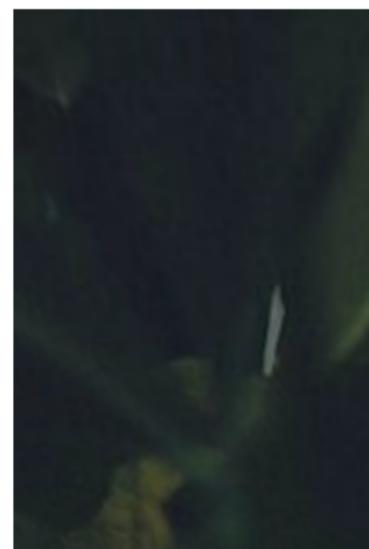


2.0.4 Perspektywa

Zmienia perspektywę obrazu, przy czym efekty mogą obejmować korekcję perspektywy i tworzenie efektów trójwymiarowych.

```
[123]: def change_perspective(img, points):
    src_points = np.float32(points)
    height, width = img.shape[:2]
    dst_points = np.float32([[0, 0], [width, 0], [0, height], [width, height]])
    matrix = cv2.getPerspectiveTransform(src_points, dst_points)
    return cv2.warpPerspective(img, matrix, (width, height))
```

```
[136]: apply_transformation(
    change_perspective,
    imgs=[IMAGES_HIGH_RES[3], IMAGES_HIGH_RES[1]],
    points = [[100, 50], [200, 50], [50, 200], [200, 200]]
)
```

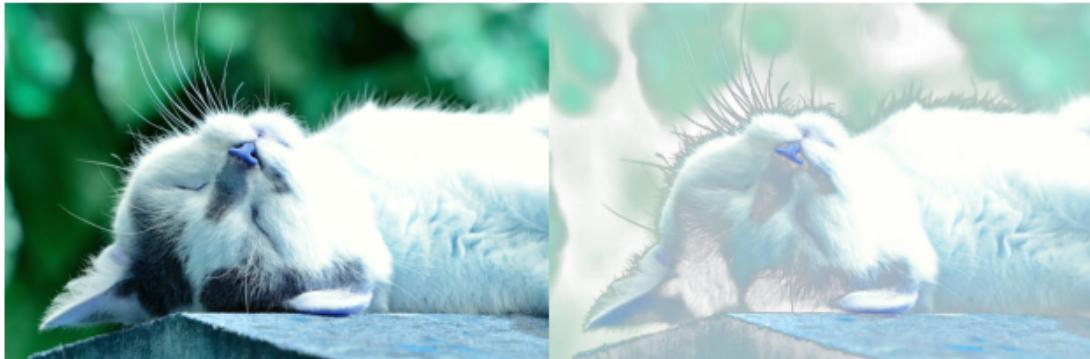


2.0.5 Transformacje falkowe

Wykorzystuje transformacje falkowe, takie jak transformata Fouriera czy transformata falkowa, do analizy częstotliwościowej obrazu i wyodrębniania szczegółów.

```
[154]: def wavelet_transform(img, wavelet_type='haar', level=1):
    img = img.astype('float32') / 255.0
    coeffs = pywt.wavedec2(img, wavelet_type, level=level)
    reconstructed_image = pywt.waverec2(coeffs, wavelet_type)
    return (reconstructed_image * 255).astype('uint8')
```

```
[158]: apply_transformation(
    wavelet_transform,
    imgs=[IMAGES_HIGH_RES[2], IMAGES_HIGH_RES[5]],
)
```

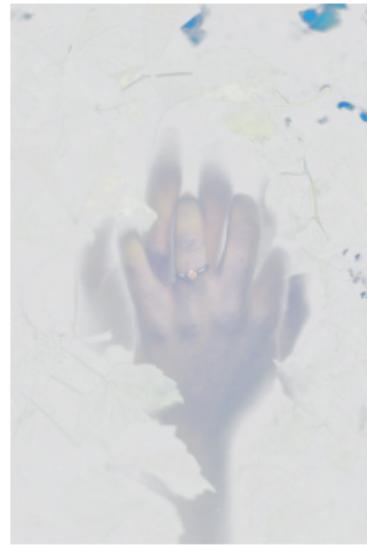
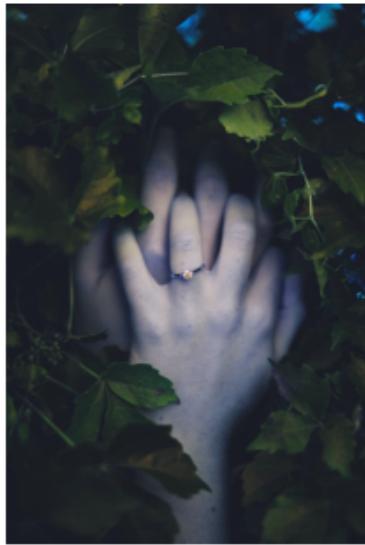


2.0.6 Korekcja chromatyczna

Poprawia balans kolorów w obrazach, eliminując niepożądane odcienie i dostosowując reprodukcję kolorów.

```
[159]: def chromatic_corrections(img):
    image_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    image_rgb = image_rgb.astype(np.float32) / 255.0
    mean_r = np.mean(image_rgb[:, :, 0])
    mean_g = np.mean(image_rgb[:, :, 1])
    mean_b = np.mean(image_rgb[:, :, 2])
    ratio_r = mean_g / mean_r
    ratio_b = mean_g / mean_b
    image_rgb[:, :, 0] *= ratio_r
    image_rgb[:, :, 2] *= ratio_b
    image_rgb = np.clip(image_rgb, 0, 1)
    return cv2.cvtColor(image_rgb, cv2.COLOR_RGB2BGR)
```

```
[161]: apply_transformation(
    wavelet_transform,
    imgs=[IMAGES_HIGH_RES[1], IMAGES_HIGH_RES[5]],
)
```

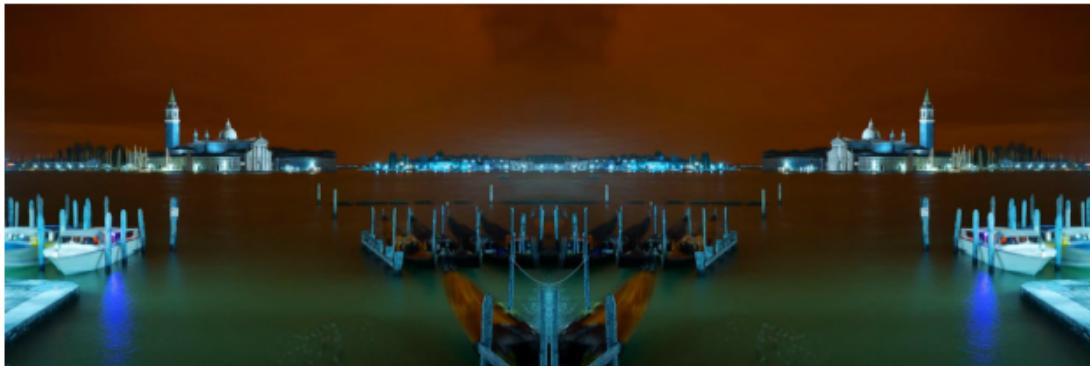


2.0.7 Odwracanie obrazu

Zmienia wartości pikseli na przeciwną, co może tworzyć efekty lustrzane lub artystyczne.

```
[169]: def reverse_image(img):
        return cv2.flip(img, 1)
```

```
[171]: apply_transformation(
        reverse_image,
        imgs=[IMAGES_HIGH_RES[4], IMAGES_HIGH_RES[5]],
    )
```



2.0.8 Zwiększenie rozdzielczości obrazu

Wykorzystuje informacje z wielu obrazów do zwiększenia rozdzielczości i uzyskania bardziej szczegółowego obrazu.

```
[172]: low_res_imgs = fetch_images(
    [
        "https://i.pinimg.com/736x/60/b0/95/60b095f51b0b5e689ec0b503fc1c37b.
        ↪jpg",
        "https://i1.sndcdn.com/artworks-000486312813-afsh6w-t500x500.jpg",
        "https://64.media.tumblr.com/93a9f8679b5f8639b973a1c1a04a02ab/
        ↪tumblr_oql1x2Bk1H1v9ks36o1_1280.jpg",
    ]
)
```