

# ImageEditor - aplikacja do obróbki zdjęć

Natalia Choszczyk, Mateusz Deptuch

Marzec 2025

## Spis treści

<b>1 Wprowadzenie</b>	<b>2</b>
<b>2 Specyfikacja techniczna</b>	<b>2</b>
2.1 Wykorzystane biblioteki . . . . .	2
<b>3 Interfejs użytkownika</b>	<b>2</b>
<b>4 Struktura aplikacji</b>	<b>3</b>
4.1 app . . . . .	4
4.1.1 app.py . . . . .	4
4.1.2 logo.png . . . . .	4
4.2 tools . . . . .	4
4.2.1 Convolve.py . . . . .	4
4.2.2 CreateMatrix.py . . . . .	5
4.2.3 Blur.py . . . . .	5
4.2.4 Brightness.py . . . . .	6
4.2.5 ContrastAdjuster.py . . . . .	6
4.2.6 EdgeDetect.py . . . . .	7
4.2.7 Grayscale.py . . . . .	8
4.2.8 Negative.py . . . . .	9
4.2.9 Plots.py . . . . .	9
4.2.10 Saturation.py . . . . .	10
4.2.11 Sharpen.py . . . . .	11
4.2.12 Threshold.py . . . . .	12
4.2.13 Mask.py . . . . .	12
<b>5 Źródła</b>	<b>13</b>

# 1 Wprowadzenie

Image Editor to aplikacja umożliwiająca zaawansowane przetwarzanie obrazów. Zawiera takie funkcje, jak zmiana jasności, kontrastu, przekształcenie do skali szarości lub negatywu, binaryzacja, rozmycie, wyostrzenie, czy detekcja krawędzi. Dodatkowo aplikacja zawiera wizualizacje w postaci histogramów i wykresów projekcji pionowej oraz poziomej. Przetworzony obraz można wyeksportować.

## 2 Specyfikacja techniczna

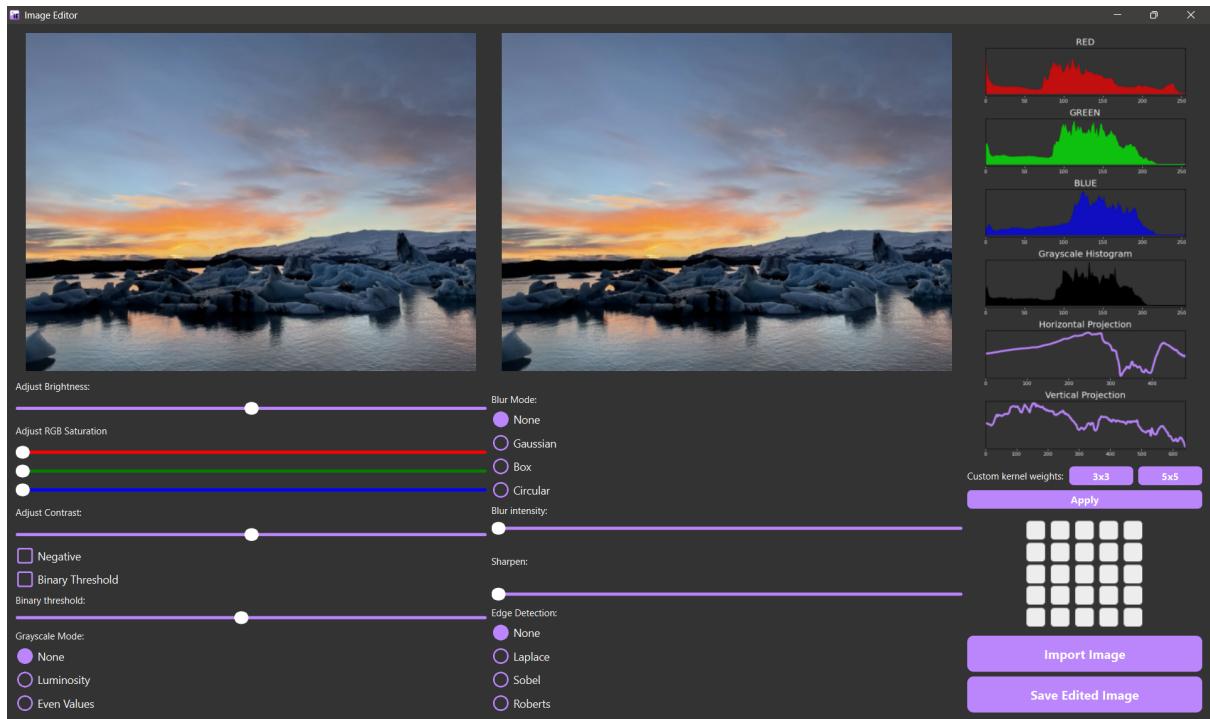
Aplikacja została napisana w języku Python. Cały program jest zaprojektowany w modularny sposób, aby umożliwić dodawanie nowych funkcjonalności w prosty sposób. Kod implementujący każdy element znajduje się w oddzielnym pliku, a szablon aplikacji jest podzielony na wiele segmentów dotyczących poszczególnych grup elementów. Importowanie oraz eksportowanie zdjęć obsługuje pliki w formacie: Images (\*.png \*.jpg \*.bmp \*.jpeg).

### 2.1 Wykorzystane biblioteki

- **PyQt6** - do interfejsu graficznego użytkownika (GUI) oraz do stworzenia desktopowej aplikacji.
- **NumPy** - do podstawowych operacji macierzowych.
- **Pillow (PIL)** - do wczytania i zapisywania obrazu.
- **Matplotlib** - do generowania wykresów takich jak histogramy, czy projekcje.

## 3 Interfejs użytkownika

Jest to aplikacja desktopowa. Po uruchomieniu, użytkownik może zaimportować zdjęcie za pomocą przycisku "Import Image". Po wykonaniu tej operacji, w aplikacji wyświetla się dwa zdjęcia. Z lewej strony zdjęcie oryginalne, a z prawej strony zdjęcie obrabione. Po prawej stronie zdjęć znajdziemy wykresy, histogramy dla każdego z kanałów RGB osobno, oraz histogram dla obrazu w skali szarości pozwalający na analizę jasności zdjęcia. Poniżej znajdują się wykresy projekcji pionowej oraz poziomej. Użytkownik może nakładać przekształcenia zdjęć oraz regulować ich intensywność za pomocą przycisków i suwaków. Po wykonaniu przekształceń obrazu, użytkownik może zapisać plik za pomocą przycisku "Save Edited Image".



Rysunek 1: Widok aplikacji po zimportowaniu zdjęcia

### Dostępne przekształcenia:

- **Adjust Brightness:** pozwala na regulację jasności zdjęcia, poziom filtra regulujemy za pomocą suwaka.
- **Adjust RGB Saturation:** składa się z trzech suwaków dla każdego kanału RGB. Zwiększa nasycenie poszczególnych kolorów RGB.
- **Adjust Contrast:** pozwala na regulację kontrastu zdjęcia, poziom filtra regulujemy za pomocą suwaka.
- **Grayscale Mode:** przekształca obraz do skali szarości. Do wyboru są dwie strategie: "Luminosity" tworzy skalę szarości za pomocą kombinacji liniowej o wagach 0.4 dla koloru czerwonego, 0.59 dla koloru zielonego oraz 0.11 dla koloru niebieskiego oraz "Even Values" tworzy skalę szarości z wagami 0.43 dla każdego koloru.
- **Negative:** przekształca obraz na negatyw.
- **Binary Threshold:** przekształca obraz na kolor biały i czarny. Za pomocą suwaka możemy regulować próg odcięcia.
- **Blur:** generuje rozmycie obrazu. Dostępne są trzy strategie: "Gaussian" uśrednia piksele zgodnie z rozkładem normalnym, "Box" uśrednia piksele w kwadracie, "Circular" uśrednia piksele w okręgu. Intensywność rozmycia możemy regulować za pomocą suwaka.
- **Sharpen:** pozwala na wyostrzenie obrazu. Moc wyostrzenia regulujemy za pomocą suwaka.
- **Edge Detection:** przekształcenie wykrywające krawędzie w obrazie. Mamy do wyboru trzy opcje: "Laplace", "Sobel", "Roberts".
- **Custom kernel weights:** pozwala na ręczne zdefiniowanie maski o rozmiarze 3x3 lub 5x5. Następnie dokonywany jest splot macierzy obrazu z maską.

## 4 Struktura aplikacji

Aplikacja jest podzielona na foldery: app oraz tools. Dodatkowo aplikacja jest wywoływana w pliku **main.py**. Poniżej znajdują się opisy poszczególnych funkcji.

## 4.1 app

### 4.1.1 app.py

Jest to główny plik aplikacji wykonanej z wykorzystaniem biblioteki PyQt6. Zawiera klasę ImageEditor, w której są zdefiniowane kolejne funkcje, opisane poniżej.

- **initUI**: funkcja definiująca layouty aplikacji. Opisuje strukturę podzieloną na pojedyncze układy, połączone w spójną całość. Definiuje wszystkie suwaki, przyciski i rozmiary poszczególnych układów. Dodatkowo opisuje wygląd aplikacji - kolory, czcionki, kształty i inne.
- **resizeEvent**: funkcja automatycznie zmieniająca rozmiar zdjęć w przypadku zmiany wielkości okna aplikacji.
- **display\_image**: funkcja konwertująca obraz na format QPixmap. Obraz jest skalowany, aby dopasować się do rozmiaru QLabel, zachowując proporcje.
- **update\_image**: funkcja przetwarza obraz na podstawie wybranych ustawień: jasności, kontrastu, trybu szarości, binaryzacji, negatywu, rozmycia, wyostrzania i detekcji krawędzi. Korzysta z funkcji zdefiniowanych w folderze tools. Obraz oraz wykresy są aktualizowane.
- **import\_image**: otwiera okno dialogowe do wyboru pliku obrazu, ładuje obraz do aplikacji, konwertując go na tablicę NumPy i wyświetla go w QLabel. Po załadowaniu obrazu wywołuje funkcję **update\_image**, aby zastosować domyślne filtry.
- **save\_image**: otwiera okno dialogowe do wyboru lokalizacji i formatu zapisu obrazu. Zapisuje przetworzony obraz w formacie PNG, JPG, BMP lub JPEG, używając biblioteki PIL do zapisu.

### 4.1.2 logo.png

Logotyp aplikacji w formacie PNG.

## 4.2 tools

Poniżej zostały opisane poszczególne opcje przekształceń. Każde przekształcenie zostało wykonane na przykładowym zdjęciu pokazanym poniżej.



Rysunek 2: Przykładowe zdjęcie

### 4.2.1 Convolve.py

Plik zawiera funkcję **convolve** przeprowadzającą operację splotu na obrazie używając podanego jądra przekształcenia. Do obrazu dodawany jest odpowiedni margines, następnie przeprowadzany jest splot:

```
for i in range(image_height):  
    for j in range(imagge_width):  
        output[i, j] = np.sum(kernel * padded_image[i:i+kernel_height, j:j+kernel_width])
```

Zwrócony zostaje przetworzony obraz.

#### 4.2.2 CreateMatrix.py

Plik służy do definiowania własnych macierzy przekształcenia o rozmiarze 3x3 lub 5x5. Ręcznie wpisujemy wartości macierzy w aplikacji.

#### 4.2.3 Blur.py

Plik zawiera definicję funkcji służących do nakładania efektu rozmycia. Składa się z trzech funkcji pomocniczych definiujących trzy różne jądra generujące efekt blur: box, gaussian oraz circular oraz funkcji blur, w której nakładany jest efekt.

- **generate\_box\_kernel:** funkcja generująca jądro o kształcie kwadratu (maskę) o boku długości `kernel_size`, z równymi wartościami. Jądro jest normalizowane, aby jego suma wynosiła 1. Poniżej przykładowe jądro dla `kernel_size = 5`.

$$\begin{bmatrix} 0.04000 & 0.04000 & 0.04000 & 0.04000 & 0.04000 \\ 0.04000 & 0.04000 & 0.04000 & 0.04000 & 0.04000 \\ 0.04000 & 0.04000 & 0.04000 & 0.04000 & 0.04000 \\ 0.04000 & 0.04000 & 0.04000 & 0.04000 & 0.04000 \\ 0.04000 & 0.04000 & 0.04000 & 0.04000 & 0.04000 \end{bmatrix}$$

- **generate\_gaussian\_kernel:** funkcja generująca jądro o kształcie kwadratu (maskę) o boku długości `kernel_size` przy użyciu rozkładu Gaussa, który rozmywa obraz w sposób bardziej naturalny, nadając większą wagę pikselom znajdującym się bliżej środka. Używa parametru `sigma`, który kontroluje szerokość rozkładu, a jądro jest normalizowane, aby jego suma wynosiła 1. Funkcja działa dla nieparzystych wartości `kernel_size`. Poniżej przykładowe jądro dla `kernel_size = 5, sigma = 4`.

$$\begin{bmatrix} 0.03520 & 0.03866 & 0.03989 & 0.03866 & 0.03520 \\ 0.03866 & 0.04246 & 0.04381 & 0.04246 & 0.03866 \\ 0.03989 & 0.04381 & 0.04520 & 0.04381 & 0.03989 \\ 0.03866 & 0.04246 & 0.04381 & 0.04246 & 0.03866 \\ 0.03520 & 0.03866 & 0.03989 & 0.03866 & 0.03520 \end{bmatrix}$$

- **generate\_circular\_kernel:** funkcja generująca jądro o kształcie okręgu w obrębie kwadratowej maski o boku długości `kernel_size`. Wszystkie wartości wewnętrz okręgu są ustawione na 1, a poza nim na 0. Jądro jest normalizowane, aby jego suma wynosiła 1. Funkcja działa dla nieparzystych wartości `kernel_size`. Poniżej przykład dla `kernel_size = 5`.

$$\begin{bmatrix} 0 & 0 & 0.07692 & 0 & 0 \\ 0 & 0.07692 & 0.07692 & 0.07692 & 0 \\ 0.07692 & 0.07692 & 0.07692 & 0.07692 & 0.07692 \\ 0 & 0.07692 & 0.07692 & 0.07692 & 0 \\ 0 & 0 & 0.07692 & 0 & 0 \end{bmatrix}$$

- **blur:** funkcja wykonująca rozmycie obrazu na podstawie wybranego typu filtra (`blur_type`) i rozmiaru jądra (`kernel_size`). Obsługiwane typy to "box", "gaussian" i "circular". W zależności od wybranego typu, generowane jest odpowiednie jądro. Następnie, każdy kanał koloru obrazu jest rozmywany poprzez użycie funkcji `convolve`, która dokonuje operacji splotu, wykorzystując wygenerowane wcześniej jądro. Wynikowy obraz jest przycinany, aby wartości pikseli były w zakresie od 0 do 255, a następnie konwertowany na typ `uint8`.



Rysunek 3: Gaussian kernel



Rysunek 4: Gaussian kernel - większe rozmycie



Rysunek 5: Box kernel



Rysunek 6: Circular kernel

#### 4.2.4 Brightness.py

Plik zawiera definicję funkcji `adjust_brightness` przyjmującej macierz obrazu oraz argument `value`  $\in [-255, 255]$  mówiący o stopniu zmiany jasności obrazu. Funkcja ta dodaje wartość argumentu `value` do każdego z elementów macierzy obrazu i zwraca obraz o zmienionej jasności.



Rysunek 7: Zdjęcie rozjaśnione



Rysunek 8: Zdjęcie przyciemnione

#### 4.2.5 ContrastAdjuster.py

Plik zawiera definicję funkcji `adjust_contrast` przyjmującej macierz obrazu oraz argument `factor` mówiący o stopniu zmiany kontrastu obrazu. Wzór tej operacji wygląda jak poniżej:

$$a_c = (a - \bar{a}) \cdot \text{factor} + \bar{a},$$

gdzie  $a$  to wartość intensywności piksela, a  $\bar{a}$  to średnia intensywność. Po wykonaniu tej operacji, wartości są przycinane do zakresu  $[0, 255]$  i zwracane jest przetworzone zdjęcie.



Rysunek 9: Zmniejszony kontrast



Rysunek 10: Zwiększyony kontrast

#### 4.2.6 EdgeDetect.py

Plik zawiera implementację trzech różnych filtrów wykrywających: `roberts_cross`, `sobel_operator` oraz `laplace_filter`. Każda z tych metod korzysta z funkcji `convolve` do przeprowadzenia operacji splotu z odpowiednimi maskami.

- Funkcja `roberts_cross` do wykrycia krawędzi wykorzystuje dwie maski:

$$K_A = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}, \quad K_B = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$$

Dla każdego kanału obrazu obliczane są wartości splotu z tymi maskami, a następnie wynikowa macierz obliczana jest jako:

$$R = \sqrt{A^2 + B^2}$$

gdzie  $A$  i  $B$  to wyniki splotu z maskami  $K_A$  i  $K_B$ . Ostateczne wartości są przycinane do zakresu  $[0, 255]$ .

- Funkcja `sobel_operator` stosuje operator Sobela, który korzysta z następujących masek:

$$K_A = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}, \quad K_B = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

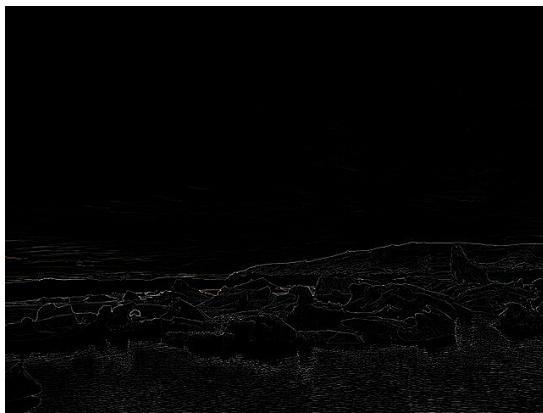
Podobnie jak w przypadku operatora Roberts Cross, wartości krawędzi są obliczane jako:

$$R = \sqrt{A^2 + B^2}$$

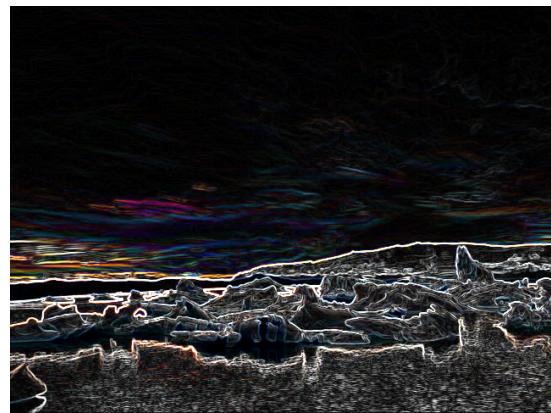
- Funkcja `laplace_filter` stosuje filtr Laplace'a, który wykrywa krawędzie w oparciu o maskę postaci:

$$K = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Każdy kanał obrazu jest przetwarzany osobno, a końcowe wartości są przycinane do zakresu  $[0, 255]$ .



Rysunek 11: Laplace



Rysunek 12: Sobel



Rysunek 13: Krzyż Robertsa

#### 4.2.7 Grayscale.py

Funkcja `apply_grayscale` konwertuje obraz kolorowy na skalę szarości, wykorzystując dwie różne metody ważenia składowych RGB: *luminosity* oraz *even values*.

- **Metoda Luminosity**

Przy wyborze trybu "luminosity", intensywność piksela w skali szarości obliczana jest jako ważona suma składowych RGB według wzoru:

$$G = 0.4R + 0.59G + 0.11B$$

gdzie  $R$ ,  $G$  i  $B$  to wartości składowych koloru w obrazie.

- **Metoda Even Values**

Tu składowe RGB są traktowane równoważnie, a intensywność piksela obliczana jest jako:

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

,  
co odpowiada średniej arytmetycznej wartości kolorów.

- **Generowanie obrazu w skali szarości** Po obliczeniu wartości jasności dla każdego piksela wynikowy obraz w skali szarości jest tworzony poprzez powielenie tej samej wartości dla wszystkich trzech kanałów (R, G, B), co pozwala zachować zgodność formatu obrazu z oryginalnym obrazem wejściowym.



Rysunek 14: Metoda Luminosity



Rysunek 15: Metoda Even Values

#### 4.2.8 Negative.py

Funkcja `negative` dokonuje dla każdego z pikseli obrazu operacji:

$$a_{\text{negative}} = 255 - a,$$

gdzie  $a$  to wartość intensywności danego piksela, co zwraca zdjęcie z odwróconymi kolorami.

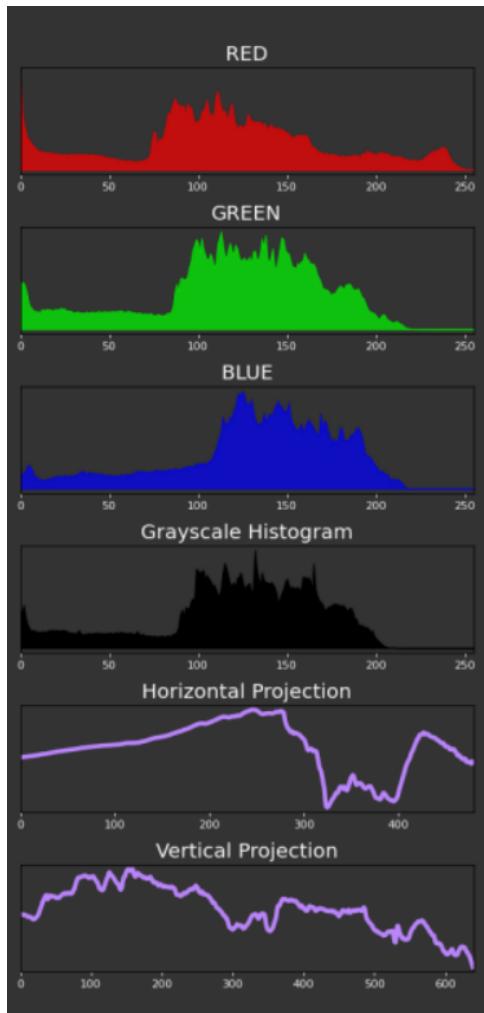


Rysunek 16: Negatyw

#### 4.2.9 Plots.py

Plik zawiera funkcję `update_plots`, która dla przekształconego obrazu generuje histogram dla każdego kanału RGB, histogram skali szarości oraz wykres projekcji poziomej i pionowej. Poniżej znajduje się szczegółowy opis poszczególnych etapów funkcji:

- **Przygotowanie danych obrazu:** na początku obraz jest konwertowany do tablicy NumPy o kształcie (`wysokość, szerokość, 3`), gdzie 3 oznacza trzy kanały kolorów RGB.
- **Tworzenie wykresów histogramu dla kanałów RGB:** funkcja generuje wykresy histogramów dla kolorów: czerwonego, zielonego i niebieskiego. Dla każdego kanału przedstawiony jest rozkład pikseli na histogramie.
- **Histogram skali szarości:** funkcja tworzy histogram dla obrazu w skali szarości, gdzie każdy z kanałów ma wagę 0.43. Przedstawia rozkład wartości intensywności od 0 do 255.
- **Projekcje poziome i pionowe:** funkcja generuje wykres projekcji poziomej oraz pionowej obrazu. Projekcja pozioma jest sumą wartości pikseli wzduż wierszy obrazu, a projekcja pionowa – sumą wartości pikseli wzduż kolumn. Są wizualizowane za pomocą wykresów liniowych.
- **Tworzenie obrazu wykresu:** wykres jest konwertowany jako obraz i wyświetlany w aplikacji.



Rysunek 17: Wyświetlone wykresy dla zdjęcia

#### 4.2.10 Saturation.py

Plik składa się z trzech funkcji - dla każdego z kanałów RGB. Są to: `adjust_red_saturation`, `adjust_green_saturation`, `adjust_blue_saturation`. Funkcje dodają wartość określoną za pomocą suwaka do odpowiedniego kanału w ten sposób: `img_array[:, :, i] += value`, gdzie `i` to liczba z przedziału `[0, 2]` odpowiadająca odpowiedniemu kanałowi.



Rysunek 18: Zwiększone nasycenie koloru czerwonego



Rysunek 19: Mocniej zwiększone nasycenie koloru czerwonego



Rysunek 20: Zwiększone nasycenie koloru zielonego



Rysunek 21: Zwiększone nasycenie koloru niebieskiego

#### 4.2.11 Sharpen.py

Plik zawiera definicję funkcji `sharpen`, dodającej filtr wyostrzający do przetwarzanego obrazu. Funkcja ta pozwala na nałożenie trzech różnych poziomów wyostrzenia: *mean-removal*, *hp1* i *hp2*, dokonując splotu (przy użyciu funkcji `convolve`) z wykorzystaniem jądrer, kolejno:

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{bmatrix}, \quad \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}, \quad \begin{bmatrix} 1 & -2 & 1 \\ -2 & 5 & -2 \\ 1 & -2 & 1 \end{bmatrix}$$

Finalnie zwracany jest przetwozony obraz.



Rysunek 22: Lekkie wyostrzenie (mean removal)



Rysunek 23: Średnie wyostrzenie (hp1)



Rysunek 24: Mocne wyostrzenie (hp2)

#### 4.2.12 Threshold.py

Funkcja `apply_threshold` konwertuje obraz kolorowy na binarny, stosując próg jasności. Intensywność  $I$  piksela obliczana jest jako suma składowych RGB. Każdy piksel jest następnie progowany według wartości `threshold`:

$$a = \begin{cases} 0, & \text{jeśli } I < 3 \cdot \text{threshold} \\ 255, & \text{w p.p.} \end{cases}$$

Wynikowy obraz jest powielany we wszystkich trzech kanałach.



Rysunek 25: Threshold - niski poziom



Rysunek 26: Threshold - wysoki poziom

#### 4.2.13 Mask.py

Funkcja `mask` pozwala na dokonanie splotu podanej macierzy obrazu z macierzą maski podaną w argumentie, przy wykorzystaniu funkcji `convolve`. Poniżej zamieszczony jest przykład dla maski postaci:

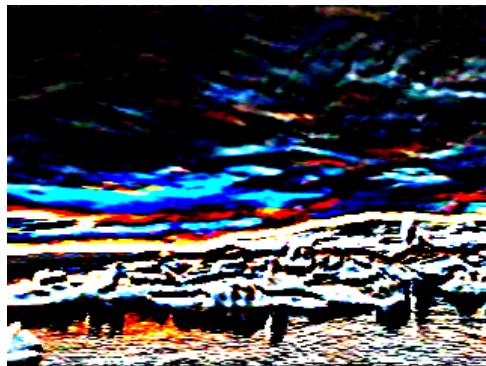
$$\begin{bmatrix} 1 & -1 & 1 \\ -1 & 1 & -1 \\ 1 & -1 & 1 \end{bmatrix}$$



Rysunek 27: Przykładowy splot z maską 3x3

Poniższy obraz powstał przy splocie z maską:

$$\begin{bmatrix} 10 & 9 & 8 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -8 & -9 & -10 \end{bmatrix}$$



Rysunek 28: Przykładowy splot z maską 5x5

## 5 Źródła

- <https://pages.mini.pw.edu.pl/~rafalkoj/www/?Dydaktyka:2024>
- <http://www.algorytm.org/przetwarzanie-obrazow/filtrowanie-obrazow.html>
- [https://www.bogotobogo.com/python/OpenCV\\_Python/python\\_opencv3\\_image\\_histogram\\_calcHist.phpgoogle\\_vignette](https://www.bogotobogo.com/python/OpenCV_Python/python_opencv3_image_histogram_calcHist.phpgoogle_vignette)
- ChatGPT