

# Biometria Projekt 1 - Analiza i Przetwarzanie Obrazów

Karolina Dunal i Zofia Kamińska

21 marca 2025

## Spis treści

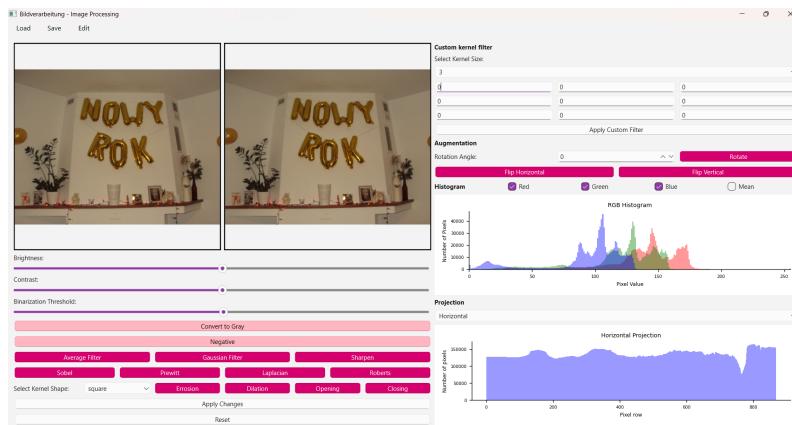
<b>1 Cel projektu</b>	<b>2</b>
<b>2 Opis aplikacji</b>	<b>2</b>
2.1 Obsługa aplikacji . . . . .	2
<b>3 Opis zaimplementowanych metod i prezentacja wyników</b>	<b>3</b>
3.1 Interaktywna modyfikacja parametrów obrazu . . . . .	3
3.1.1 Jasność . . . . .	3
3.1.2 Kontrast . . . . .	3
3.1.3 Binaryzacja . . . . .	3
3.1.4 Odcienie szarości . . . . .	3
3.1.5 Negatyw . . . . .	3
3.2 Filtry . . . . .	4
3.2.1 Filtr uśredniający . . . . .	4
3.2.2 Filtr gaussowski . . . . .	5
3.2.3 Filtr wyostrzający . . . . .	5
3.3 Wykrywanie krawędzi . . . . .	5
3.3.1 Operator Sobela . . . . .	5
3.3.2 Operator Prewitta . . . . .	5
3.3.3 Krzyż Roberts'a . . . . .	6
3.3.4 Operator Laplace'a . . . . .	6
3.4 Filtr niestandardowy (Custom Filter) . . . . .	6
3.5 Operacje morfologiczne . . . . .	6
3.5.1 Implementacja funkcji <i>convolute_binary</i> i <i>get_kernel_bin</i> . . . . .	6
3.5.2 Erozja . . . . .	7
3.5.3 Dylatacja . . . . .	7
3.5.4 Otwarcie (Opening) . . . . .	7
3.5.5 Zamknięcie (Closing) . . . . .	7
3.6 Augmentacja obrazu (rotacje i odbicia) . . . . .	7
3.7 Histogram i projekcje . . . . .	7
<b>4 Przykłady i wnioski</b>	<b>8</b>
4.1 Modyfikacje kolorystyczne . . . . .	8
4.2 Rotacje i odbicia . . . . .	9
4.3 Filtry . . . . .	9
4.4 Wykrywanie krawędzi . . . . .	10
4.5 Operacje morfologiczne . . . . .	11
<b>5 Podsumowanie</b>	<b>11</b>
<b>6 Źródła</b>	<b>12</b>

# 1 Cel projektu

Celem projektu było zaprojektowanie aplikacji okienkowej umożliwiającej edycję obrazów poprzez operacje na pikselach oraz zastosowanie filtrów graficznych, bez użycia zaawansowanych bibliotek do przetwarzania obrazów, takich jak *OpenCV*, *PIL (Python Imaging Library)* czy *scikit-image*. Aplikacja pozwala na wykonanie podstawowych operacji na obrazach, takich jak zmiana kolorystyki, modyfikacja parametrów obrazu oraz zastosowanie różnych filtrów graficznych.

## 2 Opis aplikacji

Projekt został zrealizowany w języku Python z wykorzystaniem biblioteki *PyQt6* do stworzenia interfejsu graficznego. Do przetwarzania obrazów zastosowano bibliotekę *NumPy*, umożliwiającą szybkie operacje na macierzach, oraz *OpenCV* do wczytywania i zapisu zdjęć. Do wizualizacji danych wykorzystano *Matplotlib*, która umożliwia generowanie wykresów. Aplikacja składa się z trzech głównych modułów: *image\_processor*, implementującego klasę *ImageProcessor* odpowiedzialną za operacje przetwarzania obrazów, *gui\_welcome*, zarządzającego oknem powitalnym, oraz *gui\_main*, obsługującego interfejs użytkownika. Zwrócono uwagę na utrzymanie czytelnego i intuicyjnego interfejsu. Celem było zapewnienie prostoty obsługi, przy jednoczesnym zachowaniu przejrzystości i łatwości dostępu do wszystkich wymaganych funkcji. Poniżej zamieszczono zrzut ekranu przedstawiający wygląd aplikacji.



Rysunek 1: Zrzut ekranu aplikacji przedstawiający interfejs użytkownika.

### 2.1 Obsługa aplikacji

Aplikacja oferuje intuicyjny interfejs z prostymi w obsłudze elementami do pracy z obrazami. Poza standardowymi przyciskami aktywującymi konkretne metody oraz suwakami, dostępne są również dodatkowe guziki związane z obsługą plików, które znajdują się zarówno w menu, jak i bezpośrednio w interfejsie użytkownika. W celu uruchomienia aplikacji należy uruchomić plik *gui\_main.py* znajdujący się w folderze *code*.

#### Przyciski wewnętrz interfejsu

- **Apply Changes:** zatwierdza bieżące zmiany wprowadzone do obrazu, zapisując je jako nowe "aktualne" zmiany, co umożliwia nakładanie na siebie kilku operacji. Po jego naciśnięciu, zmiany stają się trwałe, a poprzedni obraz jest przechowywany, co umożliwia późniejsze cofnięcie zmian. Ponadto, guziki *Redo* (ponowienie) zostają wyłączone, a guziki *Undo* (cofanie) zostają aktywowane.
- **Reset:** przywraca obraz do pierwotnej wersji, czyli do wersji przed jakimkolwiek zmianami. Wszystkie wprowadzone modyfikacje są anulowane, a obraz wyświetlany w interfejsie zostaje zresetowany do wersji oryginalnej.

## Menu Aplikacji

W górnej części okna aplikacji znajduje się pasek menu, który oferuje następujące opcje:

- **Load** (Załaduj):
  - Load Image (Załaduj obraz) – pozwala na załadowanie obrazu z dysku. Aktywuje także automatycznie funkcje aktualizacji histogramu i projekcji obrazu.
- **Save** (Zapisz):
  - Save as JPG, PNG, BMP (Zapisz jako...) – zapisuje plik do wybranego formatu.
- **Exit** (Zakończ):
  - Exit (Zakończ) – zamyka aplikację.
- **Edit** (Edycja):
  - Undo (Cofnij) – umożliwia powrót do poprzedniej wersji zapisanej przy użyciu *Apply Changes*. Skrót klawiszowy: Ctrl+Z.
  - Redo (Ponów) – umożliwia ponowienie ostatniego cofniętego zapisu z *Apply Changes*. Skrót klawiszowy: Ctrl+Y.

## 3 Opis zaimplementowanych metod i prezentacja wyników

Poniżej przedstawiono zaimplementowane metody przetwarzania obrazu wraz z ich krótkim opisem.

### 3.1 Interaktywna modyfikacja parametrów obrazu

W aplikacji umożliwiono modyfikację, w sposób interaktywny, parametrów obrazu, takich jak jasność czy kontrast. Użytkownik może dynamicznie zmieniać te wartości i natychmiast zobaczyć efekty na obrazie.

#### 3.1.1 Jasność

Metoda została zaimplementowana w funkcji *adjust\_brightness*. Służy do zmiany jasności obrazu poprzez dodanie do wartości pikseli zadanej wartości, pobieranej dynamicznie z odpowiedniego suwaka w interfejsie graficznym. Po wykonanej operacji wartości pikseli są przycinane do wartości z przedziału 0–255, aby zachować poprawny zakres jasności.

#### 3.1.2 Kontrast

Metoda została zaimplementowana w funkcji *adjust\_contrast*. Służy do zmiany kontrastu obrazu poprzez przesunięcie wartości pikseli względem środka zakresu (128) i ich skalowanie o zadany współczynnik, pobierany dynamicznie z odpowiedniego suwaka w interfejsie graficznym. Po wykonanej operacji wartości pikseli są przycinane do wartości z przedziału 0–255, aby zachować poprawny zakres jasności.

#### 3.1.3 Binaryzacja

Metoda została zaimplementowana w funkcji *binarization*. Służy do binaryzacji obrazu poprzez ustawienie pikseli o wartościach powyżej zadanego progu na 255, a poniżej na 0. Wartość progu pobierana jest dynamicznie z odpowiedniego suwaka w interfejsie graficznym.

#### 3.1.4 Odcienie szarości

Metoda została zaimplementowana w funkcji *convert\_to\_gray*. Służy do konwersji obrazu na skalę szarości, wykorzystując ważoną sumę składowych RGB, a następnie duplikację ich wartości w trzech kanałach dla kompatybilności z implementacją wyświetalnia obrazu w interfejsie graficznym.

#### 3.1.5 Negatyw

Metoda została zaimplementowana w funkcji *negative*. Służy do stworzenia negatywu obrazu poprzez odejmowanie wartości poszczególnych pikseli od 255.

## 3.2 Filtry

Do implementacji filtrów wykorzystano ogólną metodę *convolve*, która przechodzi przez macierz obrazu i nakłada na nią filtr przekazany jako argument. Każdy z filtrów (opisywanych poniżej) korzysta z tej funkcji używając odpowiedniego jądra filtru, które jest zdefiniowane w słowniku w klasie *ImageProcessor*.

Poniżej zamieszczony został fragment kodu implementujący funkcję *convolve*.

```
def convolve(self, kernel):
    """
    Custom convolve function to apply a kernel to the image.
    """
    kernel_size = kernel.shape[0]
    pad = kernel_size // 2

    image_padded = np.pad(self.image, ((pad, pad), (pad, pad), (0, 0)), mode='edge')
    output = np.zeros_like(self.image, dtype=np.float32)

    for c in range(self.image.shape[2]):
        for i in range(self.image.shape[0]):
            for j in range(self.image.shape[1]):
                region = image_padded[i:i + kernel_size, j:j + kernel_size, c]
                output[i, j, c] = np.sum(region * kernel)

    return np.clip(output, 0, 255).astype(np.uint8)
```

Opis kluczowych części implementacji:

1. **Obliczanie rozmiaru jądra:** Na początku obliczany jest rozmiar jądra (`kernel_size`), a następnie ustalana jest wielkość wypełnienia (`pad`), aby zachować rozmiar obrazu po operacji splotu.
2. **Dodanie wypełnienia (padding):** Obraz jest rozszerzany poprzez dodanie wypełnienia do krawędzi obrazu. Użycie trybu `edge` zapewnia, że nowe piksele na krawędzi przyjmują wartości krawędzi oryginalnego obrazu.
3. **Pętla konwolucyjna:** Funkcja iteruje przez każdy kanał obrazu (np. R, G, B), a następnie dla każdego piksela obrazu wybiera odpowiedni region z wypełnionego obrazu. Na tym regionie wykonywana jest operacja mnożenia elementów regionu (mnożenie element po elemencie) przez wartości jądra i sumowanie wyników.
4. **Przycinanie wartości:** Po wykonaniu konwolucji wynikowe wartości są przycinane do zakresu 0-255, co zapewnia, że wartości pikseli znajdują się w poprawnym zakresie dla obrazów RGB.

### 3.2.1 Filtr uśredniający

Metoda została zaimplementowana w funkcji *average\_filter*. Służy do uzyskania efektu rozmycia obrazu i redukcji szumów poprzez obliczenie średniej arytmetycznej wartości pikseli w oknie filtru i przypisanie tej wartości do centralnego piksela. W implementacji zastosowano filtr 3x3, którego jądro wygląda następująco:

$$\begin{bmatrix} \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{bmatrix}$$

### 3.2.2 Filtr gaussowski

Metoda została zaimplementowana w funkcji *gaussian\_filter*. Służy do uzyskania efektu rozmycia obrazu i redukcji szumów poprzez zastosowanie jądra o rozkładzie Gaussa, które przypisuje wagę pikselom na podstawie ich odległości od środka jądra. Dzięki temu piksele bliższe środkowi mają większy wpływ na wynik, a piksele dalsze – mniejszy. Przykładowe jądro gaussowskie o wymiarze 3x3, które zostało użyte w implementacji:

$$\begin{bmatrix} \frac{1}{16} & \frac{2}{16} & \frac{1}{16} \\ \frac{2}{16} & \frac{4}{16} & \frac{2}{16} \\ \frac{1}{16} & \frac{2}{16} & \frac{1}{16} \end{bmatrix}$$

Jądro zostało znormalizowane przez podzielenie przez 16, co zapobiega zmianie jasności obrazu.

### 3.2.3 Filtr wyostrzający

Metoda została zaimplementowana w funkcji *sharpen*. Służy do uzyskania efektu wyostrzenia obrazu poprzez zwiększenie kontrastu między pikselami. Zwykle osiąga się to poprzez przydzielenie dodatnich wartości w centralnym pikselu i ujemnych wartości w jego sąsiedztwie. W implementacji zastosowano filtr 3x3, którego jądro wygląda następująco:

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

## 3.3 Wykrywanie krawędzi

Wszystkie poniżej wymienione metody zostały zaimplementowane w funkcji *edge\_detection*, różniąc się jedynie zastosowanymi jądrami. Operatory, takie jak Sobela i Prewitta, można zaimplementować za pomocą ośmiu różnych jąder, co umożliwia wykrywanie krawędzi w ośmiu kierunkach (z odstępami co 45 stopni). Jednak ze względu na długi czas operacji splotu zastosowano tylko dwa kierunki – pionowy i poziomy. Są one wystarczające do wykrycia większości krawędzi, co można zobaczyć na przykładowych wynikach w sekcji 4.

### 3.3.1 Operator Sobela

Operator Sobela służy do wykrywania krawędzi na obrazie poprzez obliczanie przybliżenia gradientów intensywności w dwóch kierunkach: poziomym (macierz  $G_x$ ) i pionowym (macierz  $G_y$ ). Gradient poziomy wykrywa zmiany intensywności w kierunku lewo-prawo, natomiast gradient pionowy w kierunku góra-dół. Wykorzystuje do tego następujące macierze:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Wynikowy gradient obrazu uzyskuje się poprzez połączenie gradientów poziomego  $G_x$  i pionowego  $G_y$  zgodnie z poniższą formułą:

$$G = |G_x| + |G_y|$$

### 3.3.2 Operator Prewitta

Operator Prewitta działa podobnie jak operator Sobela. Jest on mniej wrażliwy na zmiany natężenia niż Sobel, ponieważ wykorzystuje prostsze jądra. Gradienty w obu kierunkach są następnie łączone, zgodnie z formułą podaną w opisie operatora Sobela, aby wykryć krawędzie w różnych orientacjach. Operator Perwittha wykorzystuje następujące macierze:

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

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

### 3.3.3 Krzyż Robertsa

Operator Krzyża Robertsa również służy do wykrywania krawędzi na obrazie poprzez obliczanie gradientów obrazu w dwóch kierunkach: poziomym i pionowym, natomiast poprzez zastosowanie dwóch jąder o wymiarach 2x2. Operator oblicza różnice intensywności pomiędzy sąsiadującymi pikselami w obrazie. Gradienty w obu kierunkach są następnie łączone, zgodnie z formułą podaną w opisie operatora Sobela, aby wykryć krawędzie w różnych orientacjach. Krzyż Robertsa jest mniej efektywny w wykrywaniu krawędzi w porównaniu do wyżej wspomnianych metod, a także wykazuje dużą wrażliwość na szумy w obrazie. Metoda ta wykorzystuje następujące macierze:

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

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

### 3.3.4 Operator Laplace'a

Operator Laplace'a jest operatorem różniczkowym wykorzystywanym w detekcji krawędzi, który wykorzystuje drugą pochodną obrazu. Służy do wykrywania punktów, w których intensywność obrazu zmienia się najbardziej gwałtownie, co wskazuje na obecność krawędzi. Operator Laplace'a nie rozróżnia kierunków krawędzi, przez co może być bardziej wrażliwy na szumy. Mimo to, daje wyraźniejsze wyniki w wykrywaniu krawędzi w obszarach o jednolitej teksturze. Jądro operatora Laplace'a wygląda następująco:

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

## 3.4 Filtr niestandardowy (Custom Filter)

W tej sekcji użytkownik może wprowadzać własne filtry, określając wagi dla poszczególnych elementów maski filtra. Dzięki temu możliwe jest tworzenie bardziej zaawansowanych efektów i manipulowanie obrazem w bardziej spersonalizowany sposób. Użytkownik ma możliwość wyboru jednego z trzech dostępnych rozmiarów filtrów: 3x3, 5x5 oraz 7x7. Wagi filtrów dla wybranego rozmiaru są wprowadzane przez użytkownika bezpośrednio w interfejsie aplikacji. Przy obecnej implementacji należy uważać na wprowadzane wartości, ponieważ wprowadzenie zbyt dużych wartości powoduje zwiększenie jasności obrazu. Oczywiście możliwe jest późniejsze dopasowanie jasności odpowiednim suwakiem. Operacja została zaimplementowana w funkcji *apply\_custom\_filter*.

## 3.5 Operacje morfologiczne

Operacje morfologiczne to zestaw technik przetwarzania obrazów, głównie binarnych, które pozwalają na analizę i modyfikację ich struktury. W ramach niniejszej sekcji omówione zostaną cztery podstawowe operacje: erozja, dylatacja, otwarcie i zamknięcie. Przy obecnej implementacji przed użyciem operacji opisanych w tej sekcji należy zastosować binaryzację obrazu.

### 3.5.1 Implementacja funkcji *convolute\_binary* i *get\_kernel\_bin*

Funkcje *get\_kernel\_bin* oraz *convolute\_binary* współpracują w realizacji operacji morfologicznych na obrazie binarnym.

Funkcja *get\_kernel\_bin* odpowiada za generowanie jądra o zadanym kształcie. W zależności od wybranego typu zwraca odpowiednią macierz reprezentującą jądro: kwadratowe, krzyżowe, liniowe pionowe lub poziome. Jądro to jest następnie wykorzystywane w operacjach morfologicznych.

Funkcja *convolute\_binary* realizuje operacje erozji i dylatacji na obrazie binarnym. Proces ten odbywa się poprzez analizę sąsiedztwa piksela i jego modyfikację zgodnie z działaniem wybranej operacji. Dla każdej pozycji jądra sprawdzane są wartości pikseli w sąsiedztwie, a następnie podejmowana jest decyzja o zmianie wartości piksela centralnego w zależności od zadanej operacji.

$$\begin{array}{cccc} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} & \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix} & \begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix} & \begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix} \\ \text{square} & \text{cross} & \text{vertical line} & \text{horizontal line} \end{array}$$

### 3.5.2 Erozja

Erozja jest operacją, na obrazie binarnym. W implementacji funkcja *convolute\_binary* wykonuje erozję poprzez analizę sąsiedztwa piksela i usuwanie cienkich struktur. W przypadku, gdy dowolny element jądra (kernela) nie znajduje pełnego dopasowania w analizowanym fragmencie obrazu, piksel centralny zostaje ustawiony na kolor tła (biały).

### 3.5.3 Dylatacja

Dylatacja to operacja odwrotna do erozji, polegająca na rozszerzaniu obszarów. Implementacja funkcji *convolute\_inary* realizuje dylatację poprzez sprawdzenie, czy środkowy piksel analizowanego regionu jest czarny. Jeśli tak, to wszystkie piksele w obrębie jądra zostają ustawione na czarne, co prowadzi do poszerzenia obszarów obiektów na obrazie.

### 3.5.4 Otwarcie (Opening)

Operacja otwarcia jest kombinacją erozji, a następnie dylatacji. Jej głównym celem jest eliminacja drobnych szumów oraz izolowanych pikseli. Funkcja *opening* najpierw wykonuje erozję na obrazie, a następnie przeprowadza dylatację na uzyskanym wyniku, co pozwala zachować większe struktury przy jednoczesnym usunięciu niewielkich elementów zakłócających obraz.

### 3.5.5 Zamknięcie (Closing)

Zamknięcie jest odwrotnością otwarcia, tj. najpierw wykonywana jest dylatacja, a następnie erozja. Celem tej operacji jest wypełnianie drobnych luk w obiektach. Implementacja funkcji *closing* rozszerza najpierw obszary poprzez dylatację, a następnie wykonuje erozję w celu przywrócenia pierwotnego kształtu dużych struktur.

## 3.6 Augmentacja obrazu (rotacje i odbicia)

Aplikacja pozwala na przeprowadzanie augmentacji obrazu, w tym rotacji i odbić (flipów). Operacje te znajdują szerokie zastosowanie w przetwarzaniu obrazów, zwłaszcza w kontekście wzbogacania zbiorów danych w zadaniach takich jak rozpoznawanie obrazów, detekcja obiektów, a także w innych dziedzinach związanych z analizą obrazu. Ich głównym celem jest zwiększenie różnorodności danych wejściowych, co pozwala na poprawę wydajności i generalizacji modeli. Operacje rotacji oraz odbicia zostały zaimplementowane w funkcjach *rotate* oraz *flip* odpowiednio.

Aby ręcznie zaimplementować rotację, na początku kąt obrotu konwertowany jest na radiany. Następnie, w oparciu o wymiary obrazu, obliczany jest jego środek, który będzie stanowił punkt obrotu. Tworzy się macierz rotacji 2D, która jest używana do transformacji współrzędnych każdego piksela. Kolejnym krokiem jest określenie nowych wymiarów obrazu na podstawie analizy przekształconych współrzędnych rogów obrazu. Wartości te pozwalają na stworzenie pustego obrazu o odpowiednich rozmiarach, do którego przypisywane są piksele z oryginalnego obrazu na podstawie obliczonej macierzy rotacji.

W przypadku odbicia obrazu, implementacja obejmuje dwie opcje: odbicie poziome i pionowe. Dla odbicia poziomego, każdy wiersz obrazu jest odwracany, co skutkuje lustrzanym odbiciem w poziomie. Dla odbicia pionowego, cała kolejność wierszy jest odwracana, co daje efekt lustrzanego odbicia w pionie.

## 3.7 Histogram i projekcje

W ramach analizy obrazu wprowadzono funkcję generowania histogramu. Przedstawia on rozkład jasności pikseli w obrazie z podziałem na poszczególne kanały RGB, co może być pomocne przy analizie jakości obrazu, progowaniu czy dostosowywaniu kontrastu.

Zaimplementowano również opcję tworzenia projekcji obrazu, zarówno poziomej, jak i pionowej. Rodzaj projekcji obrazu można wybrać za pomocą rozwijanego menu (dropdown menu) umieszczonego nad wykresem. Projekcje te przedstawiają sumę wartości pikseli wzduż odpowiednich osi obrazu (poziomej lub pionowej), co umożliwia analizę rozkładu intensywności pikseli w wybranych kierunkach. Projekcje poziome pokazują, jak wartości pikseli zmieniają się w poziomie, a projekcje pionowe - w pionie.

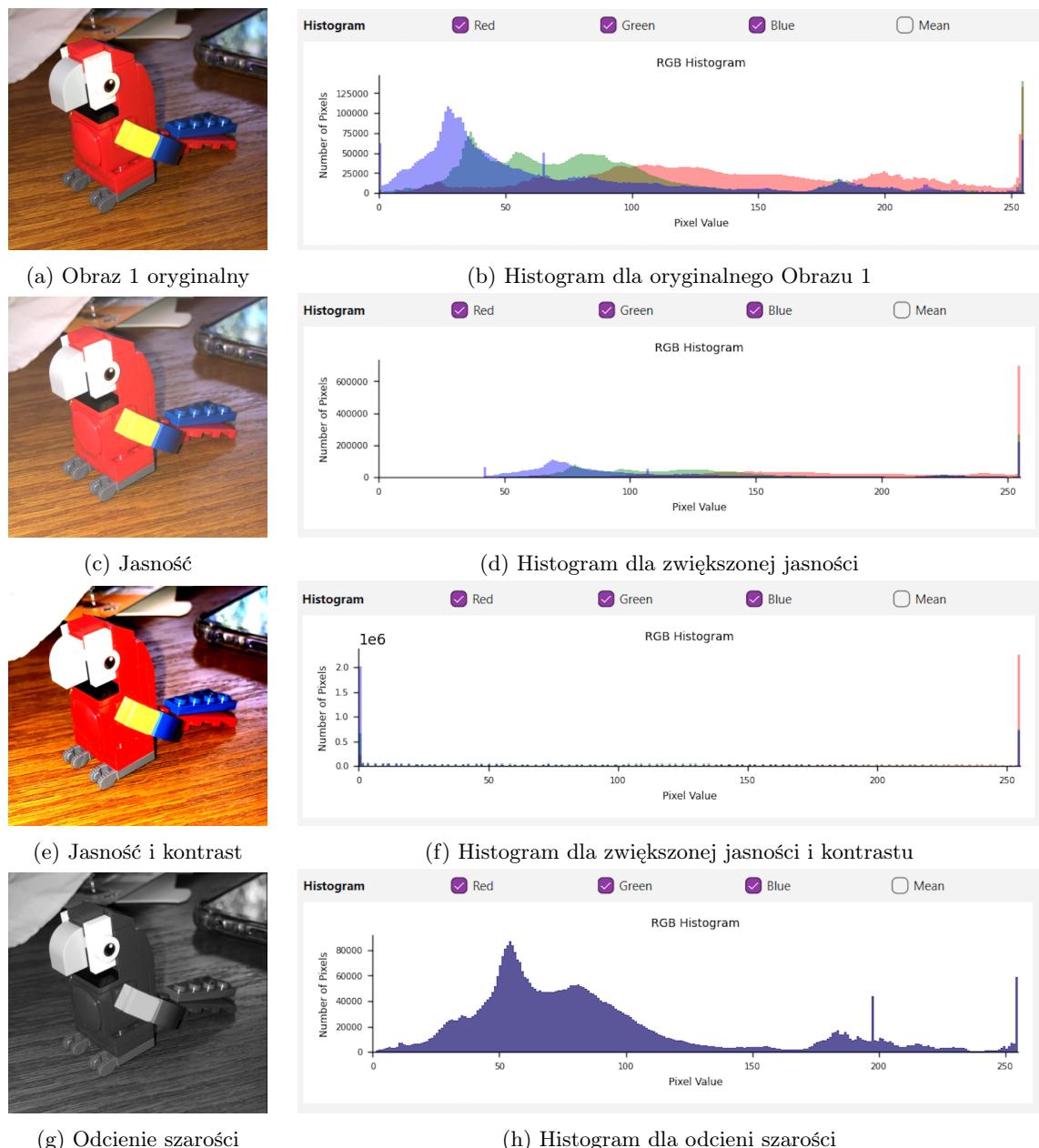
## 4 Przykłady i wnioski

### 4.1 Modyfikacje kolorystyczne

Zmiany kolorystyczne, takie jak regulacja jasności (Rysunek 2c, 2d), kontrastu (Rysunek 2e, 2f) czy konwersja do odcieni szarości (Rysunek 2g, 2h), wpływają na rozkład pikseli w obrazie. Możemy to zaobserwować na histogramach, gdzie:

- Zwiększenie jasności przesuwa słupki histogramu w prawo, co oznacza zwiększenie wartości pikseli.
- Podniesienie kontrastu powoduje rozszerzenie histogramu.
- Konwersja do skali szarości skupia histogram w węższym zakresie, eliminując informacje o barwach.

Rysunek 2 przedstawia niektóre z modyfikacji obrazów dostępnych w naszej aplikacji, wraz z odpowiadającymi im histogramami.

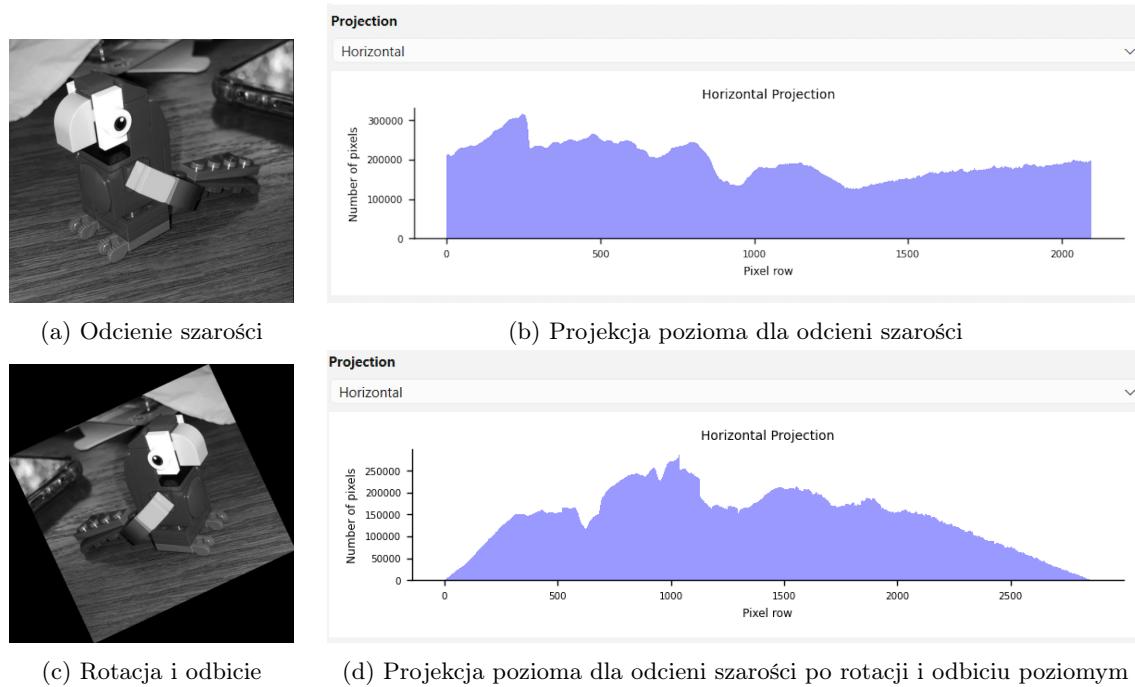


Rysunek 2: Przykłady zmian kolorystycznych i odpowiadających im zmian w histogramie

Dobrze dobrane parametry pozwalają poprawić widoczność detali, natomiast zbyt agresywna edycja może prowadzić do utraty informacji.

## 4.2 Rotacje i odbicia

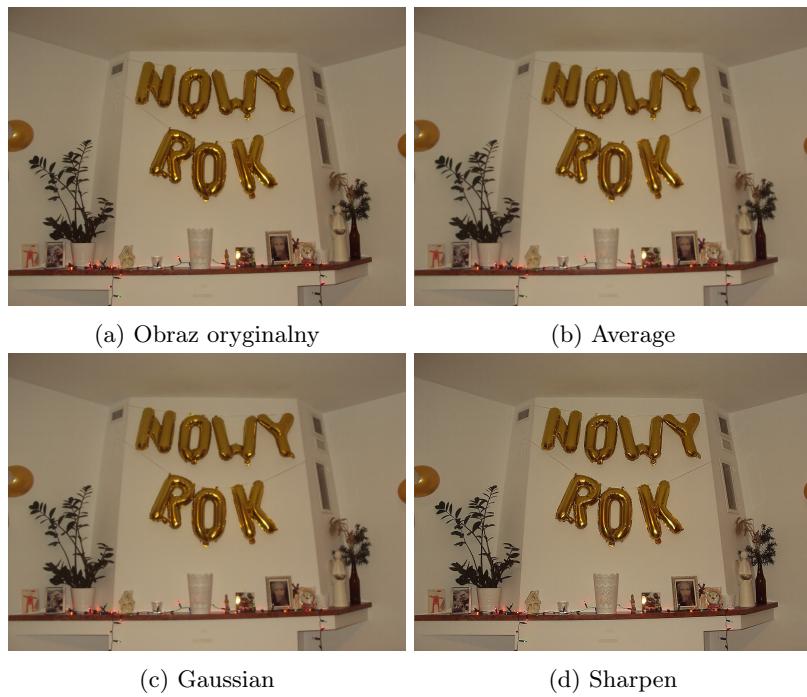
Transformacje geometryczne, takie jak rotacja i odbicie lustrzane, zmieniają orientację obiektu na obrazie, co ma wpływ na jego reprezentację w postaci projekcji. W przypadku rotacji o kąt inny niż  $0^\circ$ ,  $90^\circ$ ,  $180^\circ$  lub  $270^\circ$  zauważalne są dodatkowe czarne piksele w okolicach rogów (Rysunek 3). Powoduje to zmniejszenie wartości na krawędziach obrazu, co prowadzi do efektu przypominającego stopniowanie w projekcji zarówno poziomej jak i pionowej.



Rysunek 3: Zmiany projekcji poziomej przy rotacji o  $25$  stopni i odbiciu poziomym

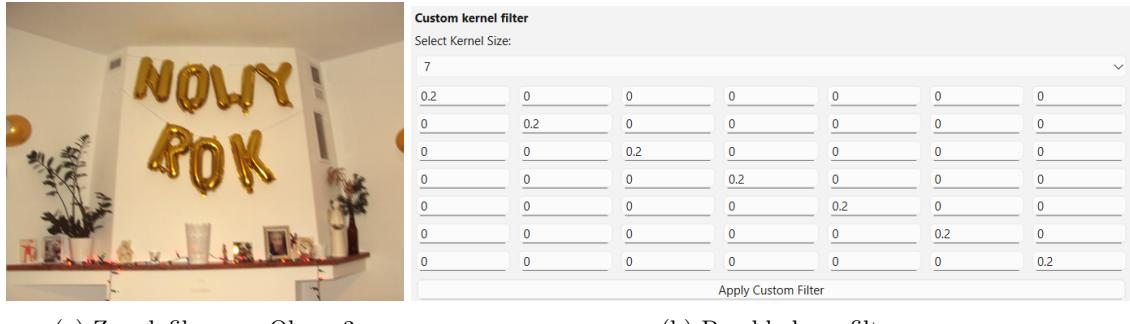
## 4.3 Filtry

Rysunek 4 przedstawia działanie podstawowych filtrów. Większe różnice w efektach działania poszczególnych operacji stają się bardziej widoczne na obrazach o wyższej rozdzielczości.



Rysunek 4: Obraz 3 po zastosowaniu różnych filtrów

Przykład nałożenia niestandardowego filtra został pokazany na Rysunku 5.



(a) Zmodyfikowany Obraz 3

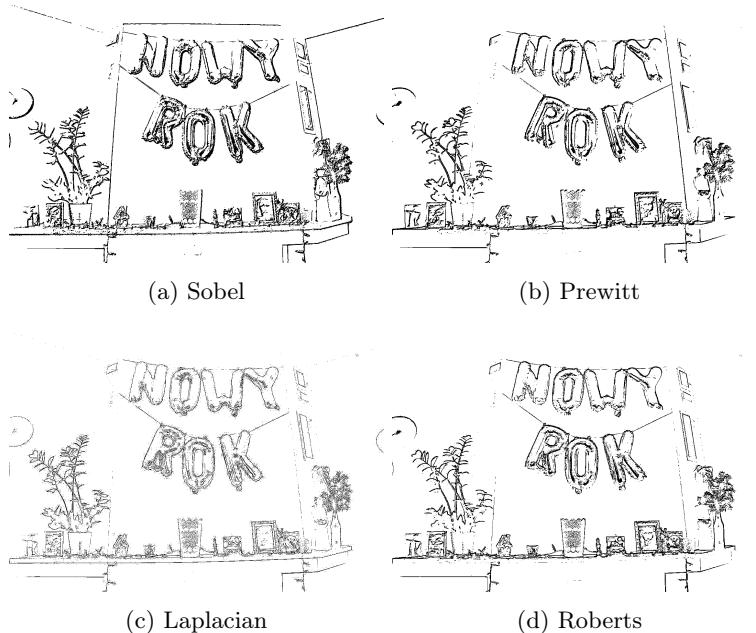
(b) Przykładowy filtr

Rysunek 5: Zastosowanie przykładowego filtru

W związku z tym, że operacja splotu została zaimplementowana ręcznie, to nakładanie filtrów na macierze jest czasochłonne, szczególnie dla obrazów o dużych rozdzielczościach, jako że trzeba przejść jądrem po całej macierzy dla każdego z kanałów RGB, co wymaga użycia trzech zagnieżdzonych pętli. Stąd, im większy rozmiar filtru, tym bardziej widoczne mogą być zmiany, ale kosztem większego czasu obliczeniowego.

#### 4.4 Wykrywanie krawędzi

W każdym przypadku, zastosowano wybraną metodę wykrywania krawędzi, nałożono negatyw, a następnie manualnie dostosowano poziom binaryzacji, dla zwiększenia czytelności obrazu wyjściowego (Rysunek 6). W przypadku metod *Laplacian* i *Roberts* próg przekazywany do funkcji binaryzacji był wyższy od pozostałych dwóch.



Rysunek 6: Obrazy po zastosowaniu różnych metod wykrywania krawędzi

Metody wykrywania krawędzi pozwalają na identyfikację konturów obiektów:

- Operator Sobela i Prewitta wykrywają krawędzie poziome i pionowe, jednak Sobel daje bardziej wyraziste efekty.
- Laplacian reaguje na szybkie zmiany jasności, przez co lepiej wykrywa drobne detale.
- Operator Robertsa, działający na najmniejszym otoczeniu, jest bardziej wrażliwy na szum.

## 4.5 Operacje morfologiczne

Aby zastosować operacje morfologiczne na obrazie (Rysunek 7a), wpierw został on poddany binaryzacji (Rysunek 7b), a następnie odpowiednią operację morfologiczną dla wybranego elementu strukturalnego (Rysunek 7c, 7d).



Rysunek 7: Obrazy po zastosowaniu różnych operacji morfologicznych

Jak można zauważyc, dylatacja poszerza ciemne obszary, co może łączyć przerwane elementy. Erozja pozwala redukować drobne szумy. Dodatkowo, wybór kształtu jądra (np. krzyżowego czy liniowego) wpływa na kierunkowość zmian.

## 5 Podsumowanie

Przedstawione operacje pokazują szerokie możliwości przetwarzania obrazów, począwszy od podstawowych zmian kolorystycznych, przez filtry i wykrywanie krawędzi, aż po operacje morfologiczne. Warto zauważyc, że wiele z tych operacji można zrealizować w znacznie krótszym czasie obliczeniowym dzięki zoptymalizowanym algorytmom oraz wykorzystaniu sprzętowej akceleracji, co znacząco przyspiesza analizę i przetwarzanie obrazów w praktycznych zastosowaniach.

## 6 Źródła

- dr inż. J. Rafałko, *Wykłady z Biometrii*, Politechnika Warszawska, 2025
- OpenAI, ChatGPT (GPT-4), <https://openai.com/chatgpt>
- <https://doc.qt.io/qtforpython-6/>
- <https://medium.com/swlh/image-processing-with-python-convolutional-filters-and-kernels-b9884d91a8fd>
- [https://sbme-tutorials.github.io/2018/cv/notes/4\\_week4.html#smoothing-kernels](https://sbme-tutorials.github.io/2018/cv/notes/4_week4.html#smoothing-kernels)
- <https://www.geeksforgeeks.org/image-edge-detection-operators-in-digital-image-processing/>