

Biometria - Projekt 1

Tomasz Żywicki, Bartłomiej Wójcik

24 marca 2025

1 Opis aplikacji

Nasz zespół przygotował prostą aplikację do przetwarzania zdjęć. Została ona wykonana w języku Python. Użyte biblioteki to:

1. Streamlit - interfejs graficzny
2. PIL - operacje na obrazach
3. Numpy - operacje na pikselach
4. Matplotlib - do wykonania histogramów i projekcji

2 Opis metod

2.1 Konwersja do odcieni szarości (`grayscale()`)

Przetwarzanie obrazu często wymaga redukcji informacji o kolorze w celu uproszczenia analizy i przetwarzania (np. projekcje czy wykrywanie krawędzi). Konwersja do odcieni szarości zachowuje jedynie jasność pikseli. Istnieje kilka metod konwersji obrazu do skali szarości:

2.1.1 Metoda średniej składowych RGB

Jednym z najprostszych sposobów jest obliczenie średniej wartości składowych R, G i B dla każdego piksela:

$$I_{gray} = \frac{R + G + B}{3} \quad (1)$$

Metoda ta równomiernie traktuje każdą składową, ale nie oddaje poprawnie percepcyjnej jasności obrazu.

2.1.2 Metoda luminancji

Lepszym podejściem jest wykorzystanie wzoru wagowego, który uwzględnia różną percepcję składowych barwnych przez ludzkie oko:

$$I_{gray} = 0.2989R + 0.5870G + 0.1140B \quad (2)$$

Metoda ta zapewnia bardziej realistyczne odwzorowanie jasności obiektów.

2.1.3 Jedna składowa

W niektórych przypadkach można wykorzystać tylko jedną składową (np. R, G lub B), np.:

$$I_{gray} = G \quad (3)$$

Wybór zielonej składowej (G) często daje najlepsze rezultaty, ponieważ ludzkie oko jest na nią najbardziej czułe.

2.1.4 Metoda desaturacji (Desaturation)

Metoda ta polega na obliczeniu średniej wartości maksymalnej i minimalnej składowej RGB dla każdego piksela:

$$I_{gray} = \frac{\max(R, G, B) + \min(R, G, B)}{2} \quad (4)$$

Podejście to pozwala zachować kontrast obrazu i często daje naturalnie wyglądające wyniki.

2.1.5 Metoda dekompozycji (Decomposition)

Dekompozycja wykorzystuje tylko jedną wartość spośród składowych RGB – maksymalną lub minimalną:

$$I_{gray}^{max} = \max(R, G, B) \quad (5)$$

$$I_{gray}^{min} = \min(R, G, B) \quad (6)$$

Wybór wartości maksymalnej powoduje jaśniejszy obraz, natomiast wartości minimalnej – ciemniejszy. Metoda ta może być użyteczna w analizie kontrastu lub przy obróbce obrazów o wysokim kontraście.

2.2 Korekta jasności (adjust_brightness())

Nasza implementacja korekty jasności polega na dodaniu stałej wartości do każdej składowej RGB piksela. Operacja ta może być zapisana wzorem:

$$I' = \text{clip}(I + B, 0, 255) \quad (7)$$

gdzie:

- I – oryginalna wartość składowej koloru (R, G lub B),
- B – zadany współczynnik jasności,
- $\text{clip}(\cdot, 0, 255)$ – funkcja ograniczająca wartości do zakresu $[0, 255]$.

Zwiększenie wartości B rozjaśnia obraz, natomiast jej zmniejszenie przyciemnia go. Korekta jasności znajduje zastosowanie w wielu obszarach, w tym:

- **Poprawa widoczności obrazu** – rozjaśnianie zbyt ciemnych lub przyciemnianie prześwietlonych zdjęć.
- **Efekty wizualne** – zmiana oświetlenia w grach komputerowych i aplikacjach graficznych.

- **Medycyna** – poprawa widoczności detali w obrazach rentgenowskich i tomografii.
- **Przetwarzanie wideo** – dynamiczna korekta jasności w kamerach monitoringu i transmisjach na żywo.

2.3 Korekta kontrastu (`contrast_correction()`)

Korekta kontrastu polega na modyfikacji rozkładu jasności pikseli w celu zwiększenia lub zmniejszenia różnic między obszarami jasnymi i ciemnymi

$$I_{\text{out}}(x, y) = \frac{259(c + 255)}{255(259 - c)} \cdot (I_{\text{in}}(x, y) - 128) + 128 \quad (8)$$

dla współczynnika kontrastu $c \in [-100, 100]$.

2.4 Negatyw (`negative()`)

2.5 Binaryzacja (`binarize()`)

Proces redukcji obrazu do dwóch wartości (czerni i bieli) poprzez progowanie.

$$I_{\text{bin}}(x, y) = \begin{cases} 255 & \text{gdy } I_{\text{gray}}(x, y) > T \\ 0 & \text{w przeciwnym przypadku} \end{cases} \quad (9)$$

gdzie $T \in [0, 255]$ to próg binaryzacji, a $I_{\text{gray}}(x, y)$ to piksele zdjęcia przekonwertowanego do odcieni szarości.

Zastosowania:

- OCR - przygotowanie tekstu do rozpoznawania
- Segmentacja obiektów
- Kompresja obrazów
- Analiza kształtów

2.6 Filtry graficzne (filter())

Filtры реализованные через конволюцию с маской 3x3:

$$I_{\text{out}}(x, y) = \sum_{i=-1}^1 \sum_{j=-1}^1 M(i, j) \cdot I_{\text{in}}(x + i, y + j) \quad (10)$$

где $M(i, j)$ это (i, j) -ий элемент маски.

Filtr	Maska
Average	$\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$
Gaussa	$\frac{1}{32} \begin{bmatrix} 1 & 3 & 1 \\ 3 & 16 & 3 \\ 1 & 3 & 1 \end{bmatrix}$
Wystrzajacy	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$
Emboss	$\begin{bmatrix} -2 & -1 & 0 \\ -1 & 1 & 1 \\ 0 & 1 & 2 \end{bmatrix}$

Tabela 1: Przykładowe filtry konwolucyjne użyte w aplikacji

2.7 Histogram

Narzędzie analizy statystycznej rozkładu jasności pikseli. Histogram intensywności pikseli definiujemy jako:

$$H(k) = \sum_{x=0}^{W-1} \sum_{y=0}^{H-1} \delta(I(x, y), k) \quad \text{dla } k \in [0, 255] \quad (11)$$

где:

- W, H - szerokość i wysokość obrazu
- $\delta(i, j)$ - funkcja Kroneckera (1 gdy $i=j$, 0 w przeciwnym przypadku)

Zastosowania

- Automatyczna korekta jasności/kontrastu
- Dobór parametrów binaryzacji
- Wykrywanie prześwietleń/niedoświetleń
- Analiza kanałów RGB

2.8 Projekcje

Projekcje poziome i pionowe są technikami analizy obrazu, które polegają na sumowaniu wartości pikseli wzdłuż określonej osi (poziomej lub pionowej). Pozwalają one na ekstrakcję strukturalnych informacji o obrazie, takich jak rozmieszczenie obiektów, segmentacja tekstu czy identyfikacja krawędzi.

2.8.1 Metodologia

Proces obliczania projekcji składa się z następujących kroków:

- Konwersja obrazu do skali szarości, co ujednolica wartości pikseli i upraszcza analizę.
- Binarizacja obrazu poprzez progowanie – piksele jaśniejsze od ustalonej wartości progowej są ustawiane na wartość maksymalną, a ciemniejsze na minimalną.
- Sumowanie wartości pikseli wzdłuż wybranej osi:
 - **Projekcja pozioma** – sumowanie wartości pikseli wzdłuż kolumn obrazu, co daje wgląd w poziome struktury.
 - **Projekcja pionowa** – sumowanie wartości pikseli wzdłuż wierszy obrazu, co ujawnia pionowe struktury.

Wynikiem tych operacji jest wektor wartości reprezentujących uśrednioną intensywność dla każdej linii lub kolumny obrazu, który można wizualizować jako histogram.

2.8.2 Zastosowania projekcji

Projekcje poziome i pionowe znajdują zastosowanie w wielu dziedzinach, w tym:

- **Analiza i rozpoznawanie tekstu** – segmentacja linii i znaków w systemach OCR (Optical Character Recognition).
- **Analiza struktury obrazu** – wykrywanie granic obiektów w obrazach binarnych i monochromatycznych.
- **Biometria** – analiza układu linii papilarnych i wzorców w obrazach biometrycznych.

2.9 Detekcja krawędzi

Detekcja krawędzi jest kluczową techniką w przetwarzaniu obrazów, umożliwiającą identyfikację istotnych struktur poprzez wykrywanie obszarów o gwałtownych zmianach jasności. Jest wykorzystywana w segmentacji obrazów, rozpoznawaniu obiektów, analizie kształtów oraz w aplikacjach takich jak systemy wizyjne i biometria.

2.9.1 Podejścia do detekcji krawędzi

Istnieje kilka podejść do detekcji krawędzi, wśród których najpopularniejsze to:

- Metody różniczkowe, wykorzystujące gradient intensywności pikseli (np. operator Sobela, operator Canny'ego).
- Metody statystyczne i transformacyjne (np. transformata Hougha do wykrywania linii).

2.9.2 Detekcja krawędzi metodą Sobela

W naszej aplikacji postanowiliśmy skorzystać z metody Sobela do detekcji krawędzi. Operator Sobela jest jedną z najczęściej stosowanych metod wykrywania krawędzi. Opiera się na obliczaniu gradientu obrazu poprzez konwolucję z dedykowanymi filtrami.

Operator Sobela wykorzystuje dwie macierze konwolucyjne:

- **Filtr poziomy** G_x wykrywający krawędzie w kierunku poziomym:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad (12)$$

- **Filtr pionowy** G_y wykrywający krawędzie w kierunku pionowym:

$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \quad (13)$$

Gradienty w kierunkach x i y są następnie wykorzystywane do obliczenia gradientu całkowitego:

$$G = \sqrt{G_x^2 + G_y^2} \quad (14)$$

2.9.3 Przetwarzanie obrazu

Proces detekcji krawędzi metodą Sobela obejmuje następujące kroki:

1. Konwersja obrazu do skali szarości.
2. Zastosowanie filtru Gaussa w celu wygładzenia obrazu i redukcji szumów.
3. Konwolucja obrazu z filtrami G_x i G_y w celu uzyskania gradientów.
4. Obliczenie modułu gradientu oraz jego normalizacja.
5. Progowanie wartości gradientu w celu identyfikacji krawędzi (im mniejszy próg, tym więcej krawędzi zostanie wykrytych).

2.9.4 Zastosowania detekcji krawędzi

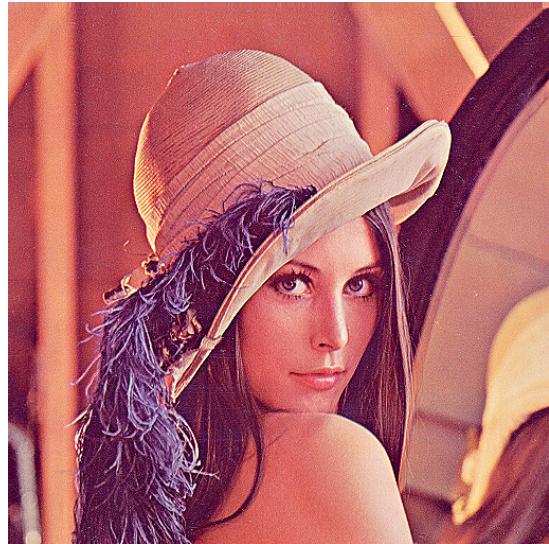
Detekcja krawędzi znajduje szerokie zastosowanie w różnych dziedzinach, takich jak:

- **Analiza obrazów medycznych** – wykrywanie struktur anatomicznych w obrazach tomograficznych i rentgenowskich.
- **Systemy rozpoznawania znaków** – identyfikacja konturów liter i cyfr w systemach OCR.
- **Robotyka i systemy wizyjne** – lokalizacja obiektów w środowisku.
- **Przetwarzanie wideo** – analiza ruchu i segmentacja scen.

3 Wyniki



(a) Oryginalne zdjęcie



(b) Wyostrzone zdjęcie

Rysunek 1: Zastosowanie filtra wyostrzającego

```
for i in range(d, n-d):
    for j in range(d, m-d):
        sub_R = R[i-d:i+d+1, j-d:j+d+1]
        sub_G = G[i-d:i+d+1, j-d:j+d+1]
        sub_B = B[i-d:i+d+1, j-d:j+d+1]

        R_new[i, j] = np.sum(np.multiply(sub_R, mask))
        G_new[i, j] = np.sum(np.multiply(sub_G, mask))
        B_new[i, j] = np.sum(np.multiply(sub_B, mask))

R_new = np.clip(R_new, 0, 255).astype(np.uint8)
G_new = np.clip(G_new, 0, 255).astype(np.uint8)
B_new = np.clip(B_new, 0, 255).astype(np.uint8)
```

(a) Oryginalne zdjęcie

```
for i in range(d, n-d):
    for j in range(d, m-d):
        sub_R = R[i-d:i+d+1, j-d:j+d+1]
        sub_G = G[i-d:i+d+1, j-d:j+d+1]
        sub_B = B[i-d:i+d+1, j-d:j+d+1]

        R_new[i, j] = np.sum(np.multiply(sub_R, mask))
        G_new[i, j] = np.sum(np.multiply(sub_G, mask))
        B_new[i, j] = np.sum(np.multiply(sub_B, mask))

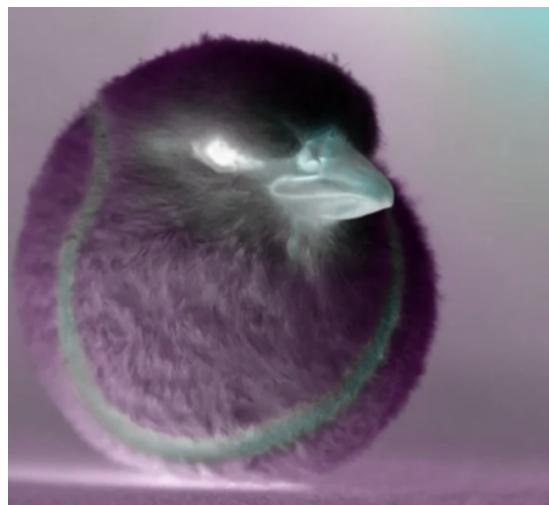
R_new = np.clip(R_new, 0, 255).astype(np.uint8)
G_new = np.clip(G_new, 0, 255).astype(np.uint8)
B_new = np.clip(B_new, 0, 255).astype(np.uint8)
```

(b) Rozmyte zdjęcie

Rysunek 2: Zastosowanie filtra rozmywającego



(a) Oryginalne zdjęcie

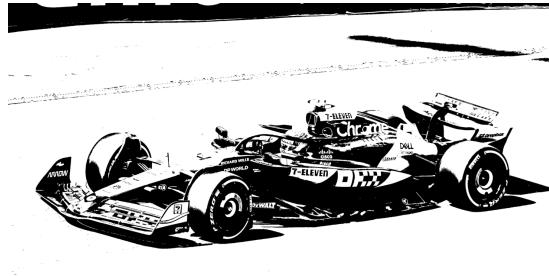


(b) Przetworzone zdjęcie

Rysunek 3: Zastosowanie negatywu

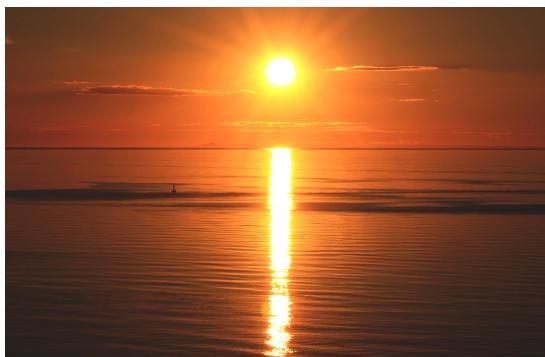


(a) Oryginalne zdjęcie

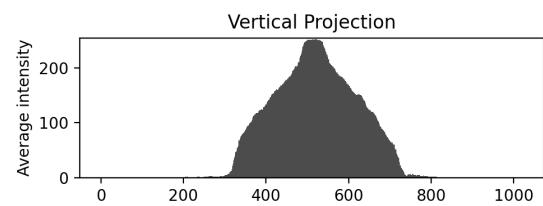


(b) Przetworzone zdjęcie

Rysunek 4: Zastosowanie binaryzacji

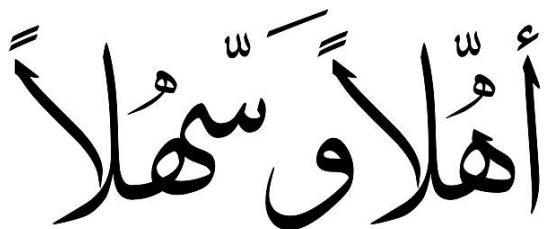


(a) Zachód słońca

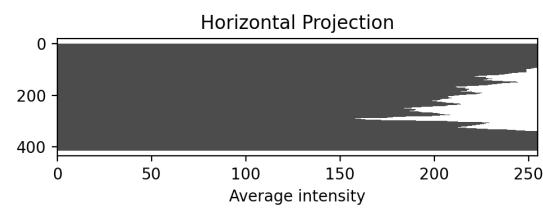


(b) Projekcja pionowa zachodu słońca

Rysunek 5: Wykorzystanie projekcji pionowej

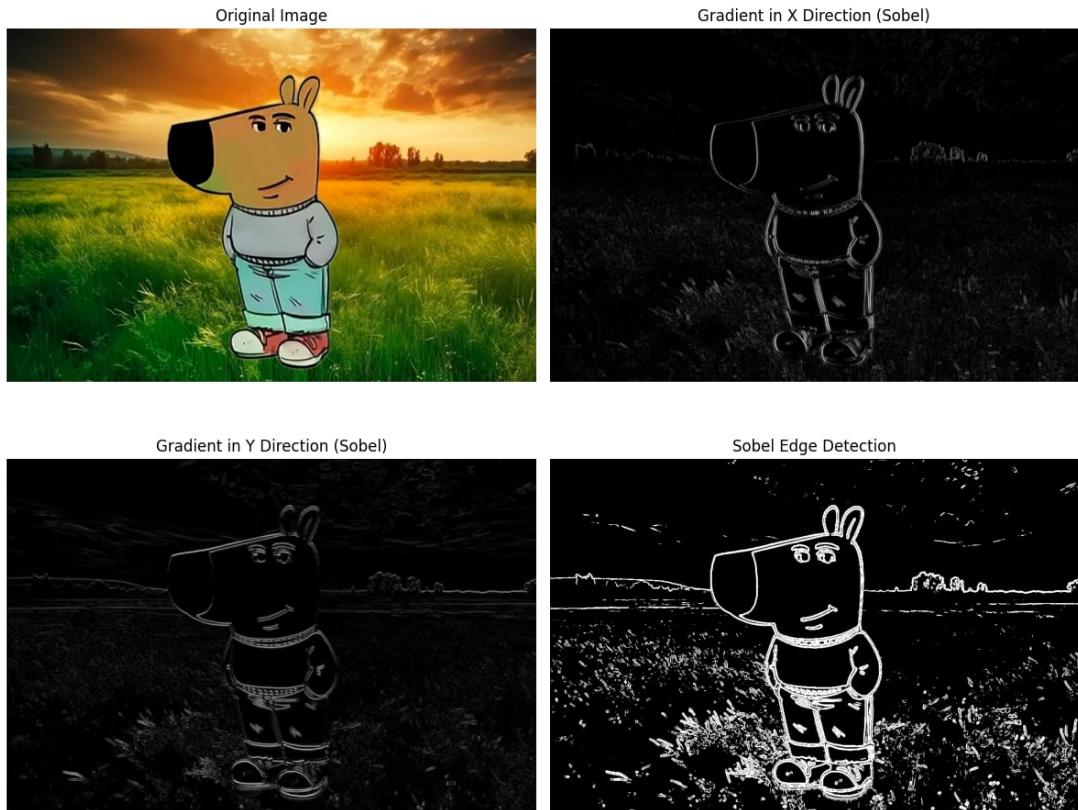


(a) Orginalny obraz



(b) Projekcja pozioma tekstu

Rysunek 6: Wykorzystanie projekcji poziomej

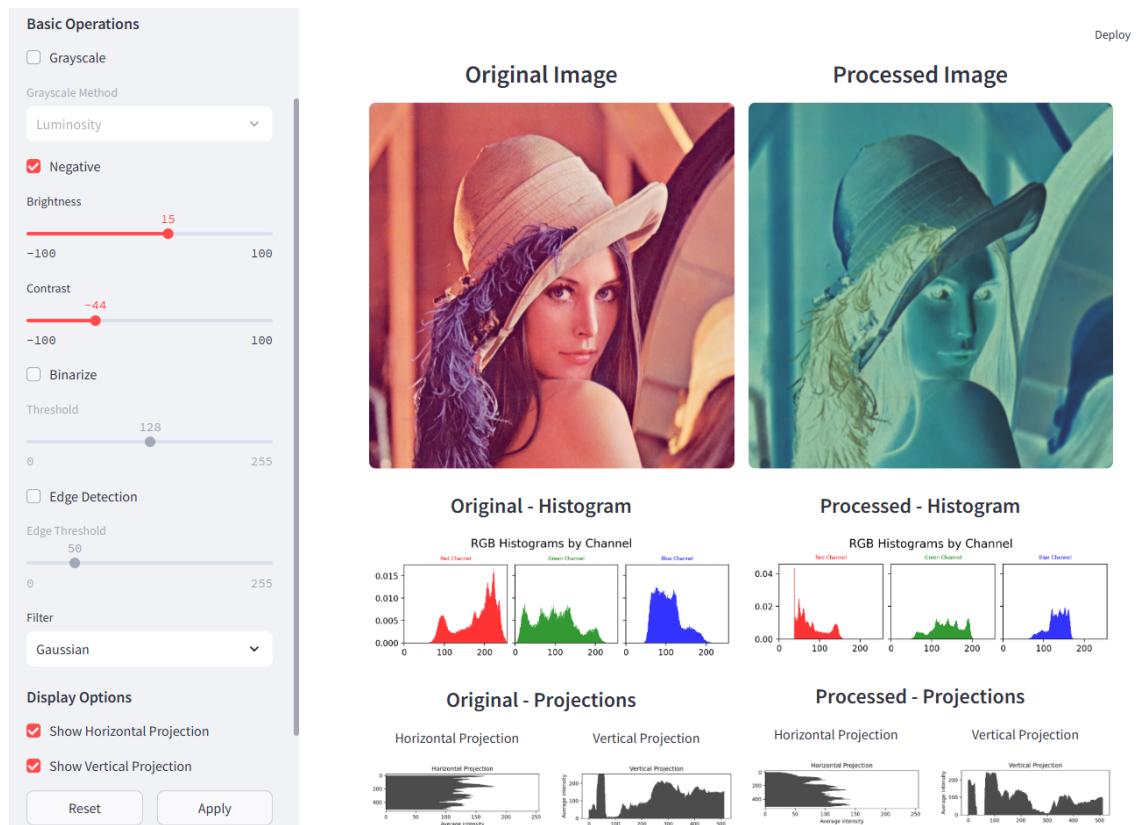


Rysunek 7: Przykład detekcji krawędzi z wykorzystaniem operatora Sobela

4 Aplikacja

W naszej aplikacji użytkownik na możliwość przetwarzania obrazu, przy użyciu wszystkich wyżej opisanych filtrów. Może je ze sobą łączyć i dostosowywać do swoich potrzeb. Ma możliwość zmiany domyślnych progów (threshold) w binaryzacji czy detekcji krawędzi.

Ciekawą opcją jest również możliwość ustawienia własnego filtra poprzez dobranie własnych wartości macierzy 3x3, pozwala to na dużą dowolność w procesie przetwarzania obrazu oraz pozwala użytkownikowi wykazać się swoją kreatywnością.



Rysunek 8: Przykładowa interakcja z naszą aplikacją