

Biometria - przetwarzanie obrazów

Martyna Leśniak, Aleksandra Samsel

27 marca 2025

1 Wstęp

Celem poniższej pracy jest przedstawienie podstawowych metod stosowanych do przetwarzania obrazów w tym operacji na pikselach, nakładania filtrów, technik używanych do wykrywania krawędzi oraz analiz graficznych takich jak histogram i projekcje. Dodatkowo omówiona zostanie implementacja tych metod w naszej aplikacji oraz sposoby jej użytkowania.

2 Opis aplikacji

Aplikacja została napisana w języku Python z wykorzystaniem bibliotek NumPy do obliczeń macierzowych, OpenCV do wczytywania obrazów, Matplotlib do wizualizacji wykresów oraz PyQt5 do obsługi interfejsu graficznego. Jest to aplikacja okienkowa, umożliwiająca modyfikowanie zdjęć oraz wyświetlanie analizy przetworzonych obrazów. Przetworzone zdjęcia można dodatkowo zapisać.

2.1 Struktura aplikacji

Aplikacja podzielona została na kilka plików:

- **main.py** – główny plik uruchamiający aplikację, zarządzający interfejsem użytkownika oraz interakcjami.
- **pixel_ops.py** – operacje na pikselach, w tym zmiana jasności, kontrastu oraz generowanie negatywu obrazu.
- **filters.py** – implementacja filtrów obrazu, takich jak rozmycie, wyostrzanie czy wykrywanie krawędzi.
- **histogram.py** – generowanie wykresów histogramu oraz projekcji.
- **transformations.py** – implementacja obrotów oraz odbicia lustrzanego zdjęcia.

2.2 Funkcje aplikacji

W menu aplikacji mamy do wyboru następujące zakładki:

- **File** - Pozwala na wybór zdjęcia do edycji, przywrócenie oryginalnej wersji oraz zapis zmodyfikowanego obrazu.
- **Transformations** - W tej zakładce możemy obrócić zdjęcie.
- **Pixel Operations** - Umożliwia dostosowanie jasności i kontrastu zmianą skali barw na skalę szarości lub zbinaryzowanie zdjęcia a także zaaplikowane negatywu.
- **Filters** - Pozwala na zaaplikowanie filtrów oraz metod wykrywania krawędzi.
- **Analysis** - Pozwala na wybór wykresu do analizy, który zostanie wyświetlony w nowym oknie.

Po zastosowaniu dowolnej metody, aby zaaplikować zmiany i móc wykorzystać kolejne narzędzia, należy kliknąć przycisk Save changes.

3 Opis metod

3.1 Operacje na pikselach

3.1.1 Konwersja do skali szarości

Konwersja do skali szarości składa się z dwóch kroków. Po pierwsze funkcja sprawdza czy obraz nie jest na wstępnie czarno-biały (wymiar obrazu równy 2). Jeśli jest to zwraca nie przekształcony obraz, gdy obraz jest kolorowy (wymiar obrazu równy 3) to funkcja korzysta z następującego wzoru:

$$Y = (R, G, B) \longrightarrow Y' = (I, I, I), \text{ gdzie } I = R \cdot 0.299 + G \cdot 0.587 + B \cdot 0.114.$$

Y to obraz bazowy a I to intensywność reprezentująca jasność dla obrazu przekształconego Y' .

3.1.2 Korekta jasności

Funkcja zmieniająca jasność obrazu działa zgodnie z poniższą regułą:

$$Y = (R, G, B) \longrightarrow Y' = (R + v, G + v, B + v),$$

gdzie v to zadana wartość zmiany jasności. Każdy piksel obrazu reprezentowany jest jako trójka wartości (R, G, B) w zakresie 0–255. Dodanie wartości v do każdego kanału powoduje rozjaśnienie obrazu, jeśli $v > 0$, lub przyćiemnienie, jeśli $v < 0$.

Aby uniknąć przekroczenia dopuszczalnego zakresu wartości, funkcja wykorzystuje operację `clip`, która ogranicza wartości do przedziału $[0, 255]$. Cały proces można zapisać jako:

$$Y' = \max(0, \min(255, Y + v))$$

3.1.3 Korekta kontrastu

Funkcja zmieniająca kontrast obrazu działa zgodnie z następującą regułą:

$$Y = (R, G, B) \longrightarrow Y' = \bar{Y} + f \cdot (Y - \bar{Y}),$$

gdzie:

- $Y = (R, G, B)$ to oryginalna wartość piksela
- \bar{Y} to średnia jasność obrazu, obliczona jako średnia wartość wszystkich pikseli,
- f to współczynnik zmiany kontrastu,
- Y' to nowa wartość piksela po zmianie kontrastu.

Działanie funkcji polega na przesunięciu wartości pikseli względem średniej jasności obrazu. Gdy $f > 1$, różnice między wartościami pikseli a średnią zostają zwiększone, co powoduje wzrost kontrastu. Gdy $0 < f < 1$, różnice te są zmniejszane, co prowadzi do osłabienia kontrastu.

Aby zapewnić, że wartości pozostają w dopuszczalnym zakresie $[0, 255]$, stosowana jest operacja `clip`.

3.1.4 Negatyw obrazu

Funkcja przekształcająca obraz na negatyw korzysta z następującej funkcji:

$$Y = (R, G, B) \longrightarrow Y' = (255 - R, 255 - G, 255 - B),$$

ponieważ piksel w 8 bitowym reprezentowany jest trzema liczbami (R, G, B) , które przyjmują wartości z zakresu 0 - 255. Wykonanie powyższej operacji przekształca biały $(255, 255, 255)$ na czarny $(0, 0, 0)$ i na odwrot oraz każdy inny kolor na jego negatyw.

3.1.5 Binaryzacja obrazu

Binaryzacja obrazu składa się z dwóch kroków. Po pierwsze obraz jest przekształcany do skali szarości za pomocą wcześniej opisanego algorytmu 3.1.1, czyli teraz każdy kanał pojedynczego piksela ma równą wartość. Następnie dla ustalonego progu p obraz jest przekształcany następująco:

$$Y = (I, I, I) \longrightarrow Y' = \begin{cases} (255, 255, 255) \text{ jeżeli } I \geq p \\ (0, 0, 0), \text{ w.p.p} \end{cases}$$

Powstaje w ten sposób obraz złożony jedynie z czarnych i białych pikseli.

3.2 Filtry graficzne korzystające z jądra

W tej sekcji opisane są filtry, których działanie opiera się na zastosowaniu jądra przekształcenia. Każdy taki filtr działa w następujący sposób:

1. Definiujemy jądro K o wymiarach $k \times k$ i ustalamy jego wagę (jądro definiuje filtr - dla każdego filtra będziemy używać innego jądra).
2. Dodajemy otoczkę do obrazu, aby uniknąć problemów na krawędziach. Obramowanie jest wielkości $\lfloor \frac{k}{2} \rfloor$ i jest wykonywane metodą odbicia.
3. Dla każdego piksela obrazu bazowego $Y[i, j]$ wyznaczamy:

$$\text{Okno}_{ij} = \{Y[x, y] : x \in [i - o, i + o], y \in [j - o, j + o]\},$$

gdzie otoczenie $o = \lfloor \frac{k}{2} \rfloor$.

4. Następnie możemy zastosować jądro

$$Y'_{ij} = \sum_{x, y \in \text{Okno}_{ij}} Y[x, y] \times K[x, y],$$

gdy mamy do czynienia z obrazem kolorowym - wykonujemy to dla każdego kanału osobno.

5. Upewniamy się, że otrzymane wartości mieszczą się w przedziale $[0, 255]$.

Po wykonaniu powyższych kroków dla każdego piksela otrzymujemy przetworzony obraz Y' .

3.2.1 Filtr uśredniający

Filtr uśredniający polega na wygładzeniu obrazu poprzez zamianę wartości piksela na średnią jego otoczenia. Przykładowe jądro wielkości 3×3 będzie wyglądać następująco:

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

3.2.2 Filtr Gaussa

Filtr Gaussa służy do wygładzania obrazu, podobnie jak filtr uśredniający, ale zamiast równomiernego uśredniania, wykorzystuje on funkcję Gaussa do wyznaczenia wag dla poszczególnych pikseli w otoczeniu. Wartości jądra wyznaczamy w następujących krokach:

1. Tworzymy siatkę współrzędnych x, y , która jest wyśrodkowana względem zera
2. Obliczamy wartości jądra

$$K(x, y) = \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right),$$

gdzie σ określa odchylenie standardowe rozkładu, a x i y to współrzędne względne względem środka jądra.

3. Normalizujemy jądro poprzez podzielenie każdej jego wartości przez sumę wszystkich wartości. Teraz suma wszystkich wartości jest równa 1.

3.2.3 Filtr wyostrzający

Filtr wyostrzajający ma na celu podkreślenie detali i krawędzi obrazu. Zasada działania filtra opiera się na wykrywaniu różnic między wartością centralnego piksela a jego otoczeniem. W macierzy jądra możemy ustalić wartość siły wyostrzenia int :

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

Im większa wartość int , tym większy wpływ ma centralny piksel na wynik, co podkreśla różnice między nim a sąsiadami.

3.2.4 Filtr z dowolnym jądrem

Użytkownik ma możliwość podania własnego jądra, które zostanie zastosowane do obrazka. Przykładowo możemy użyć jądra Emboss

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

które po zastosowaniu tworzy iluzję trójwymiarowości - uwypukla krawędzie. Zastosowanie tego efektu możemy zobaczyć na rysunku 5.

3.3 Inne filtry graficzne

3.3.1 Filtr posterizacji

Filtr posterizacji zmniejsza liczbę różnych kolorów w obrazie. To przekształcenie działa w następujący sposób:

1. Ustalamy $krok$, który jest obliczany jako $krok = \frac{256}{poziom}$, gdzie przykładowo $poziom = 4$ oznacza, że każdy kanał (RGB) będzie miał tylko 4 możliwe wartości: 0, 64, 128 i 192.
2. Nowy obraz powstaje dzięki następującemu przekształceniu:

$$Y' = \left(\left\lfloor \frac{Y}{krok} \right\rfloor \right) \times krok$$

Wartości pikseli oryginalnego obrazu są zamienione na najbliższą wielokrotność $kroku$ co redukuje liczbę odcienni danego koloru. Zastosowanie tego efektu możemy zaobserwować na rysunku 8. Zmniejszenie liczby kolorów też idealnie widać na histogramie RGB porównując ze sobą rysunki 7 i 9.

3.4 Histogram

Histogram obrazu jest wykresem przedstawiającym rozkład jasności pikseli. Oś pozioma reprezentuje poziomy intensywności (0–255 dla obrazów 8-bitowych), a oś pionowa liczbę pikseli o danej intensywności. W projekcie zastosowałyśmy dwa rodzaje histogramów:

- **Histogram dla obrazu w skali szarości** – przedstawia rozkład intensywności światła, na podstawie którego można ocenić poprawność ekspozycji obrazu.
- **Histogram dla obrazu kolorowego (RGB)** – składa się z trzech oddzielnych wykresów, odpowiadających składowym kolorystycznym obrazu: czerwonej, zielonej i niebieskiej. Każdy kanał analizowany jest osobno, co pozwala na szczegółową ocenę struktury barwnej obrazu.

3.5 Projekcje

Projekcje są techniką analizy obrazów binarnych, w której sumuje się intensywność pikseli wzduż wybranego kierunku - poziomego lub pionowego. Wyniki tej analizy przedstawiane są na histogramie, Stosuje się je głównie w zadaniach związanych z rozpoznawaniem tekstów, wykrywaniem kształtów oraz segmentacją obiektów, takich jak słowa i linie w dokumentach tekstowych.

3.6 Wykrywanie krawędzi

Wykrywanie krawędzi odgrywa kluczową rolę w wielu dziedzinach, takich jak uczenie maszynowe, gdzie jest wykorzystywane do identyfikacji i rozpoznawania obiektów w obrazach, medycyna, gdzie znajduje zastosowanie w segmentacji obrazów medycznych oraz w autonomicznych pojazdach, które dzięki niemu mogą analizować otoczenie, wykrywać pasy ruchu i unikać przeszkód.

Najczęściej stosowane metody bazują na obliczaniu gradientu obrazu, który określa zmianę intensywności pikseli w kierunkach x oraz y , co można zapisać jako

$$\nabla f = \begin{bmatrix} G_x \\ G_y \end{bmatrix}$$

gdzie składowe gradientu to:

$$G_x = \frac{\partial f}{\partial x}$$

$$G_y = \frac{\partial f}{\partial y}$$

Istnieje wiele metod do obliczania gradientu, które różnią się doborem maski konwolucyjnej, jej wymiarami oraz sposobem aplikacji na obrazie. W naszej aplikacji zaimplementowane zostały trzy różne metody:

3.6.1 Krzyż Robertsa

Najprostszą z metod, ale jednocześnie najmniej dokładną ze względu na podatność na szумy jest Krzyż Robertsa. Wykorzystuje się w nim dwie maski 2x2, dzięki którym obliczany jest gradient w kierunkach poziomym i pionowym.

Pierwsza maska (gradient w kierunku poziomym):

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

Druga maska (gradient w kierunku pionowym):

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

Obliczenie gradientu obrazu polega na zastosowaniu tych masek do oryginalnego obrazu, a następnie obliczeniu sumy bezwzględnych wartości wyników filtracji dla obu masek.

3.6.2 Operator Sobela

Operator Sobela wykorzystuje maski 3x3, przez co jest bardziej dokładny i odporny na szumy w stosunku do poprzedniego omawianej metody.

Pierwsza maska (gradient w kierunku poziomym):

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

Druga maska (gradient w kierunku pionowym):

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

Stosowanie operatora Sobela polega na filtracji obrazu za pomocą tych dwóch masek, podobnie jak w przypadku Krzyża Robertsa. Gradient w każdym punkcie obrazu obliczany jest przez zastosowanie obu masek w dwóch kierunkach. Następnie, całkowity gradient w punkcie oblicza się jako:

$$G = \sqrt{G_x^2 + G_y^2}$$

3.6.3 Operator Prewitta

Operator Prewitta jest rozwinięciem Krzyża Robertsa i podobnie jak operator Sobela, wykorzystuje maski o wymiarze 3x3. Pozwala na dokładniejsze wykrywanie krawędzi, jest bardziej odporny na zakłócenia i lepiej uwzględnia sąsiedztwo pikseli w porównaniu do metod opartych na mniejszych maskach.

Podstawowe maski operatora Prewitta:

Gradient w kierunku poziomym (G_x)

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

Gradient w kierunku pionowym (G_y)

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

Podobnie jak w przypadku operatora Sobela, gradient końcowy oblicza się jako:

$$G = \sqrt{G_x^2 + G_y^2}$$

Operator Prewitta można rozszerzyć na więcej kierunków niż tylko poziomy i pionowy, poprzez nałożenie dodatkowych masek.

4 Przykładowe wykorzystanie aplikacji

W poniższej sekcji przedstawiono przykładowe rezultaty działania aplikacji. Na rysunku 1 widoczny jest oryginalny obraz, natomiast Rysunek 2 przedstawia ten sam obraz po nałożeniu filtra posterizacji.



Figure 1: Oryginalny obraz przed obróbką.



Figure 2: Obraz po zastosowaniu filtru posterizacji.

Jak widać na rysunku 1 oraz rysunku 2, zastosowanie filtrów pozwala na uzyskanie ciekawego efektu przypominającego plakat. Na rysunku została zmniejszona ilość różnych kolorów, co wypłaszcza obraz. Dodatkowo możemy zastosować tutaj korektę jasności, co możemy zaobserwować na rysunku 3.



Figure 3: Obraz po korekcie jasności.

Innym zastosowaniem naszej aplikacji może być użycie filtra pozwalającego użytkownikowi podanie własnego jądra. Po zastosowaniu jądra Emboss 3.3.1 widzimy, że krawędzie na rysunku 4 zostały uwypuklone co możemy zaobserwować na rysunku 5.



Figure 4: Oryginalny obraz przed obróbką.



Figure 5: Obraz po zastosowaniu jądra Emboss.

Aby lepiej zrozumieć działanie filtra posterizacji warto zestawić obrazy z ich histogramami RGB. Na histogramie 7 dla rysunku 6 widzimy różnorodność kolorów. Natomiast po zastosowaniu filtra posterizacji na rysunku 8 widzimy, że ilość kolorów się zmniejszyła, co jeszcze lepiej widać na histogramie 9.



Figure 6: Oryginalny obraz przed obróbką.

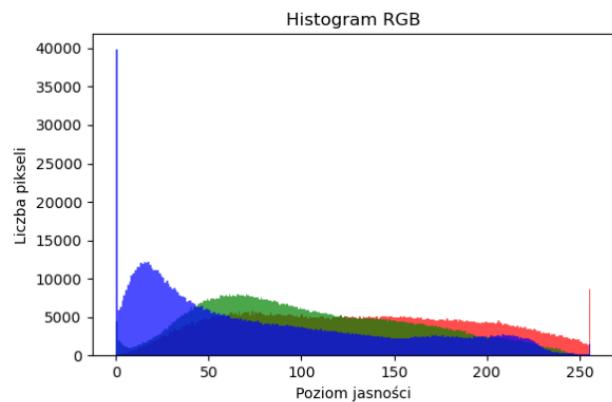


Figure 7: Histogram RGB oryginalnego obrazu przed obróbką.



Figure 8: Obraz po zastosowaniu filtra posterizacji.

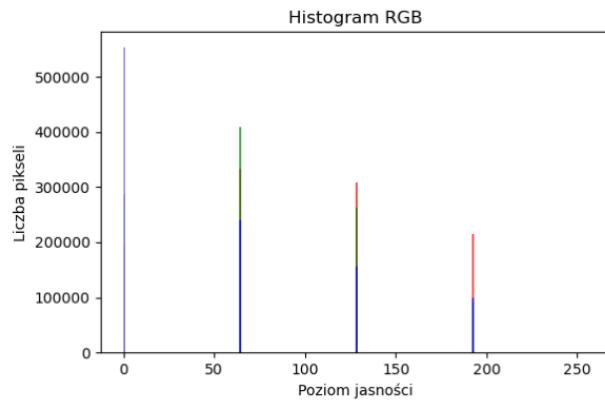


Figure 9: Histogram RGB obrazu po zastosowaniu filtra posterizacji.

5 Podsumowanie

W efekcie wykonanego projektu poznaliśmy podstawowe metody przetwarzania obrazów takie jak operacje na pikselach, przeróżne filtry czy metody wykrywania krawędzi. Wykorzystanie znanych bibliotek Pythona pozwoliło nam stworzyć aplikację, która nie tylko pozwala na modyfikację obrazów, ale również prezentuje efekty tych operacji w przejrzysty sposób. Aplikacja daje dużo możliwości od poprawienia zdjęć z wakacji po lepsze zrozumienie obrazu poprzez studiowanie jego histogramu. Jest to wstęp do dalszej części przedmiotu jaką będzie stworzenie aplikacji rozpoznajającej tęczówkę oka. Algorytmy stworzone podczas tego projektu posłużą nam w dalszych krokach.