

Biometria - Projekt 1

Aplikacja do przetwarzania obrazów

Dominika Gimzicka, Aleksandra Wójcik

27 marca 2025

Spis treści

1	Wprowadzenie	2
2	Implementacja	2
2.1	Ogólna implementacja	2
2.2	Używane biblioteki	2
3	Funkcjonalności aplikacji	3
3.1	Import i zapis plików	3
3.2	Konwersja do odcienni szarości	3
3.3	Korekta jasności	4
3.4	Korekta kontrastu	4
3.5	Negatyw	5
3.6	Binaryzacja	5
3.7	Filtры graficzne	6
3.7.1	Usredniający	6
3.7.2	Gaussa	6
3.7.3	Wystrzajacy	6
3.8	Resetowanie filtrów	7
3.9	Histogramy obrazów	7
3.10	Projekcje	7
3.11	Obsługa błędów	8
4	Przykłady użycia	9
4.1	Przykład 1	9
4.2	Przykład 2	10
4.3	Przykład 3	11
4.4	Projekcje	12
5	Wnioski	12

1 Wprowadzenie

Projekt stanowi prosty edytor obrazów BMP napisany w języku Python z wykorzystaniem biblioteki Tkinter do tworzenia interfejsu graficznego. Aplikacja umożliwia użytkownikowi wczytanie obrazu oraz stosowanie na nim szeregu filtrów, takich jak: konwersja do odcieni szarości, negatyw, binaryzacja, korekta jasności i kontrastu, a także filtry uśredniający, Gaussa i wyostrzający. Kluczowym elementem implementacji jest zastosowanie stosu filtrów, dzięki czemu wszystkie operacje na obrazie wykonywane są w oryginalnej kolejności, co pozwala zachować spójność przetwarzania. Aplikacja daje również możliwość szerszej analizy cech obrazów dzięki wyświetlaniu dotyczących ich wykresów.

2 Implementacja

2.1 Ogólna implementacja

Aplikacja została zorganizowana w formie graficznego interfejsu użytkownika (GUI) opartego na bibliotece Tkinter. Główne funkcjonalności obejmują:

- Wczytywanie obrazu (z możliwością wyboru pliku przez użytkownika).
- Zapis przetworzonego obrazu do pliku (ze wskazaniem nazwy i folderu docelowego).
- Wyświetlanie obrazu na dwóch oddzielnych kanwach – jednej dla oryginalnego obrazu oraz drugiej dla przetworzonego obrazu.
- Stosowanie serii filtrów, przy czym każdy filtr jest implementowany jako funkcja przyjmująca tablicę NumPy reprezentującą obraz i zwracającą przetworzony obraz.
- Implementację mechanizmu stosu filtrów, realizowanego przez listę `img_filter_stack`. Dzięki temu filtry są nakładane na obraz w kolejności, w jakiej zostały wybrane przez użytkownika. Każda zmiana czy filtr jest wykonywany na bazie oryginalnego obrazu, a wynik kolejnych operacji zależy od uporządkowanego nakładania się funkcji zapisanych na stosie.
- Wyświetlanie dwóch histogramów odpowiadających obrazom oryginalnemu i przetworzonemu.
- Wczytanie obrazu binarnego w celu obliczenia i wyświetlenia jego projekcji.

2.2 Używane biblioteki

W implementacji projektu wykorzystano następujące biblioteki:

- **tkinter** – służy do tworzenia graficznego interfejsu użytkownika, obsługi zdarzeń oraz wyświetlania obrazów na kanwach.
- **PIL (Pillow)** – umożliwia wczytywanie, edycję oraz konwersję obrazów, a także przekształcenie ich do formatu kompatybilnego z Tkinter.
- **NumPy** – używana do operacji na tablicach numerycznych, co pozwala na wydajne przetwarzanie obrazów.
- **scikit-image** (moduł `skimage.util.view_as_windows`) – wykorzystywany przy implementacji filtra uśredniającego do dzielenia obrazu na okna.
- **math** – wykorzystywana w funkcjach obliczeniowych, np. przy tworzeniu jądra Gaussa.

- **Matplotlib** – moduł `matplotlib.backends.backend_tkagg` pozwala wyświetlać wykresy Matplotlib w oknie aplikacji Tkinter, a moduł `matplotlib.figure` - tworzy obiekt wykresu.

3 Funkcjonalności aplikacji

Dla każdej zaimplementowanej w aplikacji operacji podane są funkcje, które odpowiadają za jej wykonanie. Funkcje rozpoczynające się od `to_` lub `adjust_` (w tych funkcjach jako argument przekazywana jest tablica NumPy reprezentująca obraz, zawierającą wartości pikseli w formacie RGB) implementują logikę przetwarzania obrazu, natomiast druga funkcja (np. `grayscale()`) odpowiada za dodanie lub usunięcie danej operacji ze stosu filtrów.

Główną funkcją odpowiadającą za zastosowanie wszystkich aktywnych operacji i filtrów oraz wyświetlenie wynikowego obrazu i jego histogramu jest `apply_filters()`.

```

1 def apply_filters():
2     clear_error()
3     global img_filter_np, img_filter
4
5     # Start from an original image
6     img_filter = img_original.copy()
7     img_filter_np = np.array(img_filter)
8
9     # Apply each filter from the stack
10    for filter_func in img_filter_stack:
11        img_filter_np = filter_func(img_filter_np)
12
13    # Update canvas
14    img_filter = Image.fromarray(img_filter_np)
15    img_tk_filter = ImageTk.PhotoImage(img_filter)
16    canvas2.create_image(0, 0, anchor=tk.NW, image=img_tk_filter)
17    canvas2.image = img_tk_filter
18
19    draw_rgb_histogram(hist_frame2, img_filter_np)
```

3.1 Import i zapis plików

Funkcje: `load_image()`, `save_image()`

Aplikacja umożliwia załadowanie pliku graficznego do obróbki w formacie .jpg lub .bmp oraz zapisanie przetworzonego obrazu, także w jednym z tych dwóch formatów.

3.2 Konwersja do odcieni szarości

Funkcje: `grayscale()`, `to_grayscale(np_image)`

Konwersja obrazu do odcieni szarości polega na przekształceniu obrazu kolorowego (RGB) w taki, w którym każdy piksel posiada tylko jedną wartość jasności, reprezentującą skalę szarości od 0 (czerń) do 255 (biel). Istnieje kilka powszechnie stosowanych metod tego przekształcenia:

1. **Średnia arytmetyczna** – wartość piksela to średnia z kanałów czerwonego, zielonego i niebieskiego: $\frac{R+G+B}{3}$.
2. **Wybór jednego kanału** – zazwyczaj wykorzystywany jest kanał zielony (G), który ma największy

wpływ na postrzeganą jasność.

3. **Metoda ważona (luminancja)** – najdokładniejsza percepcyjnie metoda, w której poszczególne kanały mają przypisane różne wagi, odpowiadające wrażliwości ludzkiego oka.

W naszej aplikacji zastosowano właśnie metodę ważoną, wykorzystując współczynniki:

$$\text{Gray} = 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B$$

Dzięki temu kanał zielony (G), mający największy wpływ na postrzeganą jasność, jest odpowiednio uwzględniony, natomiast kanały czerwony i niebieski mają mniejszy wpływ. Efektem tej transformacji jest bardziej naturalny i realistyczny obraz w odcieniach szarości, zgodny z tym, jak człowiek postrzega światło.

3.3 Korekta jasności

Funkcje: `brightness()`, `adjust_brightness(np_image)`

Korekta jasności polega na zwiększeniu lub zmniejszeniu ogólnej intensywności pikseli w obrazie. W praktyce oznacza to przesunięcie wartości każdego piksela o stałą wartość – dodatnią (rozjaśnienie) lub ujemną (przyciemnienie). W naszej aplikacji użytkownik może określić poziom zmiany w zakresie od -100 do 100.

Proces ten przebiega następująco:

1. Program pobiera od użytkownika wartość jasności i sprawdza, czy mieści się ona w dopuszczalnym zakresie.
2. Do każdego piksela w obrazie dodawana jest ta wartość.
3. Ostateczna wartość każdego piksela jest ograniczana do przedziału 0–255, aby zachować poprawny zakres jasności.

Zwiększenie jasności sprawia, że obraz wygląda na bardziej oświetlony, natomiast jej zmniejszenie może być użyteczne np. w celu stworzenia efektu przyciemnienia lub analizy szczegółów w jaśniejszych partiach obrazu.

3.4 Korekta kontrastu

Funkcje: `contrast()`, `adjust_contrast(np_image)`

Korekta kontrastu pozwala na zwiększenie lub zmniejszenie różnic pomiędzy jasnymi a ciemnymi obszarami obrazu. W naszej aplikacji użytkownik może określić poziom kontrastu w zakresie od -100 do 100, a jego wartość jest przeliczana na tzw. współczynnik kontrastu (ang. contrast factor) według wzoru:

$$\text{factor} = \frac{259 \cdot (\text{contrast_val} + 255)}{255 \cdot (259 - \text{contrast_val})}$$

Nowa wartość każdego piksela obliczana jest następnie jako:

$$\text{pixel_new} = \text{factor} \cdot (\text{pixel} - 128) + 128$$

co oznacza przeskalowanie odchyлеń od wartości środkowej (128) i przesunięcie ich z powrotem do zakresu odcieni szarości lub koloru.

Proces korekty kontrastu w aplikacji przebiega według następujących kroków:

1. Pobierana jest wartość kontrastu i sprawdzana pod względem poprawności.
2. Obraz konwertowany jest do typu `float32`, aby zapewnić precyzyjne obliczenia.
3. Obliczany jest współczynnik kontrastu zgodnie ze wzorem powyżej.
4. Nowe wartości pikseli są obliczane i ograniczane do zakresu [0, 255].
5. Obraz przekształcany jest z powrotem do typu całkowitego (`uint8`).

Wyższy kontrast podkreśla detale i sprawia, że obraz wydaje się bardziej wyrazisty. Z kolei niższy kontrast powoduje rozmycie różnic tonalnych, co może dać efekt bardziej stonowany lub "wyblakły".

3.5 Negatyw

Funkcje: `negative()`, `to_negative(np_image)`

Efekt negatywu polega na odwróceniu jasności obrazu – jasne obszary stają się ciemne, a ciemne – jasne. Osiąga się to poprzez odjęcie wartości każdego kanału koloru (R, G, B) od maksymalnej wartości, czyli 255. Operacja ta wykonywana jest dla każdego piksela obrazu osobno, zgodnie ze wzorem:

$$\text{Negative}(R, G, B) = (255 - R, 255 - G, 255 - B)$$

Taka transformacja powoduje wizualne „odwrócenie” obrazu, przypominające efekt kliszy fotograficznej. Jest to jedna z najprostszych operacji punktowych stosowanych w przetwarzaniu obrazów.

3.6 Binaryzacja

Funkcje: `binary()`, `to_binary(np_image)`

Binaryzacja to proces konwersji obrazu do postaci dwuwartościowej, w której każdy piksel przyjmuje jedną z dwóch wartości: 0 (czerń) lub 255 (biel). Działanie opiera się na zastosowaniu progu jasności, względem którego każda wartość piksela jest porównywana. Wartość piksela po binaryzacji określa wzór:

$$\text{pixel_binary} = \begin{cases} 255 & \text{jeśli } \text{pixel_value} > \text{threshold}, \\ 0 & \text{w przeciwnym razie} \end{cases}$$

W naszej aplikacji binaryzacja przebiega w dwóch etapach:

1. Weryfikacja, czy obraz jest w skali szarości. Jeżeli nie, następuje konwersja z formatu RGB do skali szarości za pomocą wzoru:

$$\text{Gray} = 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B$$

2. Następnie stosowany jest próg binaryzacji równy 128. Piksele jaśniejsze od progu przyjmują wartość 255, pozostałe – 0.

Binaryzacja jest szczególnie przydatna w zadaniach takich jak detekcja krawędzi, rozpoznawanie obiektów czy analiza kształtów.

3.7 Filtry graficzne

3.7.1 Uśredniający

Funkcje: `averaging()`, `to_averaging(np_image)`

Filtr uśredniający (inaczej filtr medianowy) to jeden z najprostszych filtrów wygładzających. Działa poprzez zastąpienie wartości każdego piksela średnią wartością pikseli z jego otoczeniem, co skutkuje redukcją szumów oraz wygładzeniem obrazu. W naszej aplikacji użytkownik może określić siłę działania filtra poprzez parametr wejściowy w zakresie 0–100, który decyduje o rozmiarze jądra (maski) stosowanego do obliczeń.

Proces działania filtra:

1. Obraz zostaje odpowiednio rozszerzony (ang. padded), aby można było przetwarzać piksele na krawędziach.
2. Dla każdego piksela pobierane jest okno o wymiarach zależnych od wybranego rozmiaru jądra.
3. Obliczana jest średnia wartość intensywności pikseli w tym oknie.
4. Ostateczna wartość piksela zostaje zastąpiona tą średnią.

Filtr ten dobrze sprawdza się w przypadku usuwania drobnych zakłóceń i wygładzania krawędzi, jednak może prowadzić do utraty szczegółów obrazu.

3.7.2 Gaussa

Funkcje: `gaussian()`, `to_gaussian(np_image)`

Filtr Gaussa to zaawansowana technika wygładzania obrazu, oparta na splotach z jądrem Gaussa. Różni się od filtra uśredniającego tym, że piksele bliżej środka maski mają większy wpływ na wynik niż te oddalone. Dzięki temu obraz zostaje wygładzony w sposób bardziej naturalny, bez tak intensywnej utraty szczegółów.

Proces działania:

1. Na podstawie wartości parametru σ (określonego przez użytkownika w zakresie 0–100) tworzona jest maska Gaussa – dwuwymiarowa macierz o wartościach odpowiadających funkcji Gaussa.
2. Dla każdego piksela wyznaczany jest fragment obrazu odpowiadający rozmiarowi jądra.
3. Wartości pikseli w tym fragmencie są mnożone przez wartości maski Gaussa i sumowane.
4. Wynik podstawiany jest jako nowa wartość piksela.

Zaletą filtra Gaussa jest jego zdolność do wygładzania obrazu przy jednoczesnym zachowaniu przejętych tonalnych. Jest szeroko stosowany w przetwarzaniu obrazów, m.in. jako wstępny etap przed detekcją krawędzi.

3.7.3 Wyostrzający

Funkcje: `sharpening()`, `to_sharpening(np_image)`

Filtr wyostrzający ma na celu wzmacnianie krawędzi i szczegółów obrazu, co skutkuje jego większą ostrością i kontrastowością. W naszej aplikacji użytkownik może określić poziom wyostrzenia poprzez parametr wejściowy w zakresie 0–100. Im wyższa wartość, tym mocniejszy efekt wyostrzenia.

Działanie filtra opiera się na zastosowaniu specjalnej maski (jądra splotowego), która wzmacnia różnice jasności pomiędzy pikselem centralnym a jego sąsiadami. Wykorzystuje się w tym celu tzw. filtr Laplace'a z dodatkowym wzmacnieniem wartości środkowej.

Proces przebiega następująco:

1. Na podstawie wartości poziomu ostrości tworzona jest maska 3×3 , w której wartość środkowa wzrasta, a sąsiednie mają wartości ujemne. Dzięki temu różnice pomiędzy pikselami zostają wyeksponowane.
2. Dla każdego piksela obrazu wybierany jest fragment sąsiedztwa i wykonywany jest splot z maską.
3. Wynikowy piksel to suma przemnożonych wartości, ucięta do zakresu 0–255.

Filtr ten jest przydatny w sytuacjach, gdzie obraz jest nieostry lub chcemy uwydniczyć krawędzie obiektów. Należy jednak stosować go z umiarem, ponieważ zbyt silne wystrzenie może prowadzić do powstawania szumów lub efektów halo wokół krawędzi.

3.8 Resetowanie filtrów

Funkcja: `reset_filters()`

W aplikacji został dodany przycisk do wygodnego cofnięcia wszystkich zastosowanych dotychczas operacji i filtrów, i powrotu do oryginalnej wersji obrazu.

3.9 Histogramy obrazów

Funkcja: `draw_rgb_histogram(canvas_container, image_array)`

Funkcja rysuje histogram jasności kanałów R, G i B obrazu na podanym kontenerze `canvas_container`. Najpierw usuwa poprzednie wykresy z kontenera, a następnie tworzy nową figurę Matplotlib. Jeśli obraz jest w odcieniach szarości (2D), zostaje przekształcony na obraz RGB przez powielenie kanału. Dla każdego z kanałów rysowany jest histogram jasności w postaci półprzezroczystych słupków. Na osi X przedstawiona jest jasność (0–255), a na osi Y liczba pikseli.

3.10 Projekcje

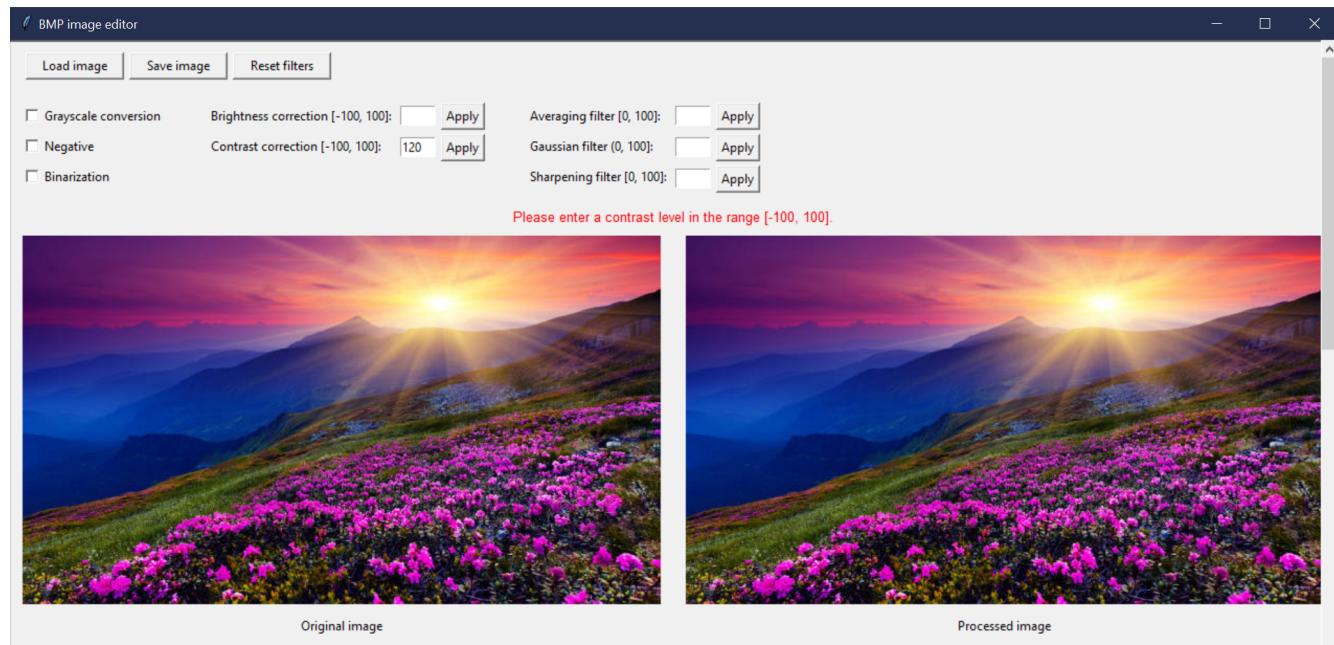
Funkcja: `plot_projections(binary_img, canvas_container)`

Funkcja wizualizuje projekcje poziome i pionowe czarnych pikseli obrazu binarnego w kontenerze `canvas_container`. Najpierw usuwa poprzednie wykresy z kontenera, a następnie sprawdza, czy w obrazie występują czarne piksele (wartość 0). Jeśli tak, wyznacza najmniejszy prostokątny obszar obejmujący wszystkie czarne piksele i go wyciną. Następnie oblicza projekcje: poziomą (liczba czarnych pikseli w każdym wierszu) i pionową (w każdej kolumnie). Tworzony jest wykres z trzema panelami: na górze pokazana jest pionowa projekcja jako wykres słupkowy, poniżej wyświetlany jest wycięty fragment obrazu, a z prawej strony — pozioma projekcja w postaci wykresu słupkowego poziomego.

3.11 Obsługa błędów

Funkcje: `show_error(msg)`, `clear_error()`

Użytkownik po wpisaniu wartości dla filtrów przekraczających podany zakres jest informowany o tym przez odpowiedni komunikat wyświetlony w GUI.



Rysunek 1: Komunikat o przekroczeniu zakresu wartości

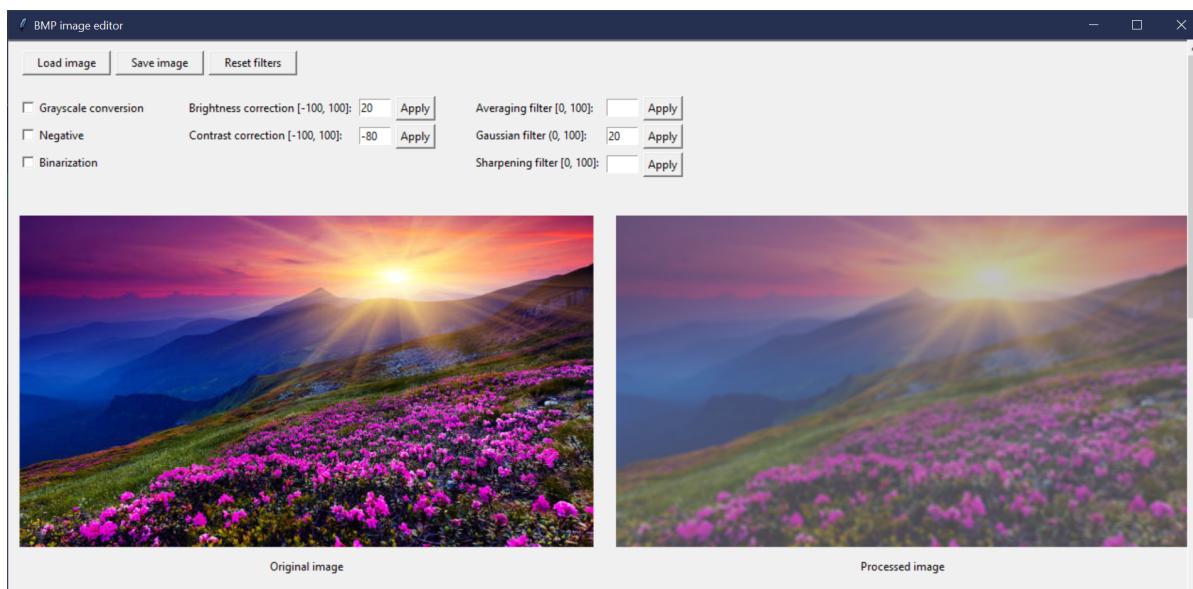
4 Przykłady użycia

Poniżej widoczny jest wygląd naszej aplikacji przy przykładowym użyciu. Dla każdego zdjęcia w opisie podane są jakie filtry i z jakimi wartościami parametrów zostały użyte.

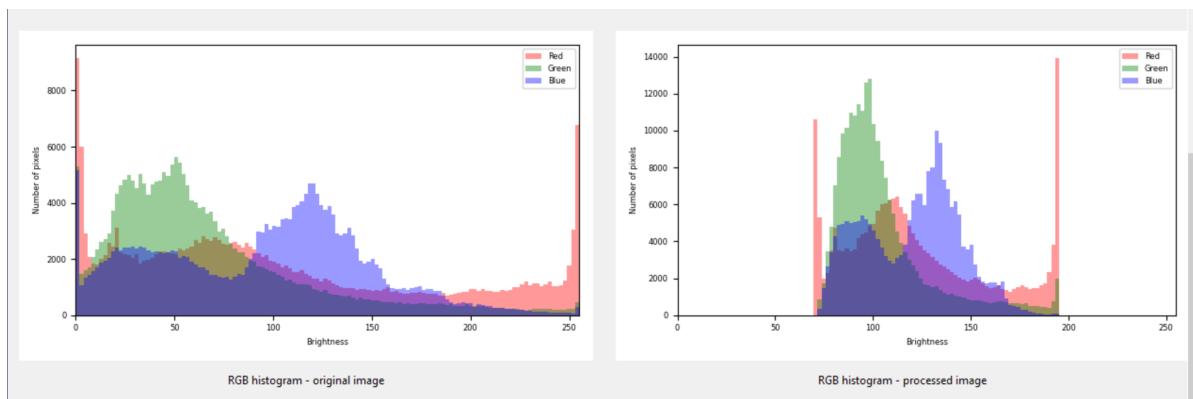
4.1 Przykład 1

W tym przykładzie nałożone zostały filtry: `brightness_correction = 20`, `contrast_correction = -80` oraz `Gaussian_filter = 20`.

Poniżej przedstawiony jest wygląd obrazka z nałożonymi filtrami oraz histogramy dla nich.



Rysunek 2: Widok obrazków dla: jasność 20, kontrast -80, filtr Gaussa 20

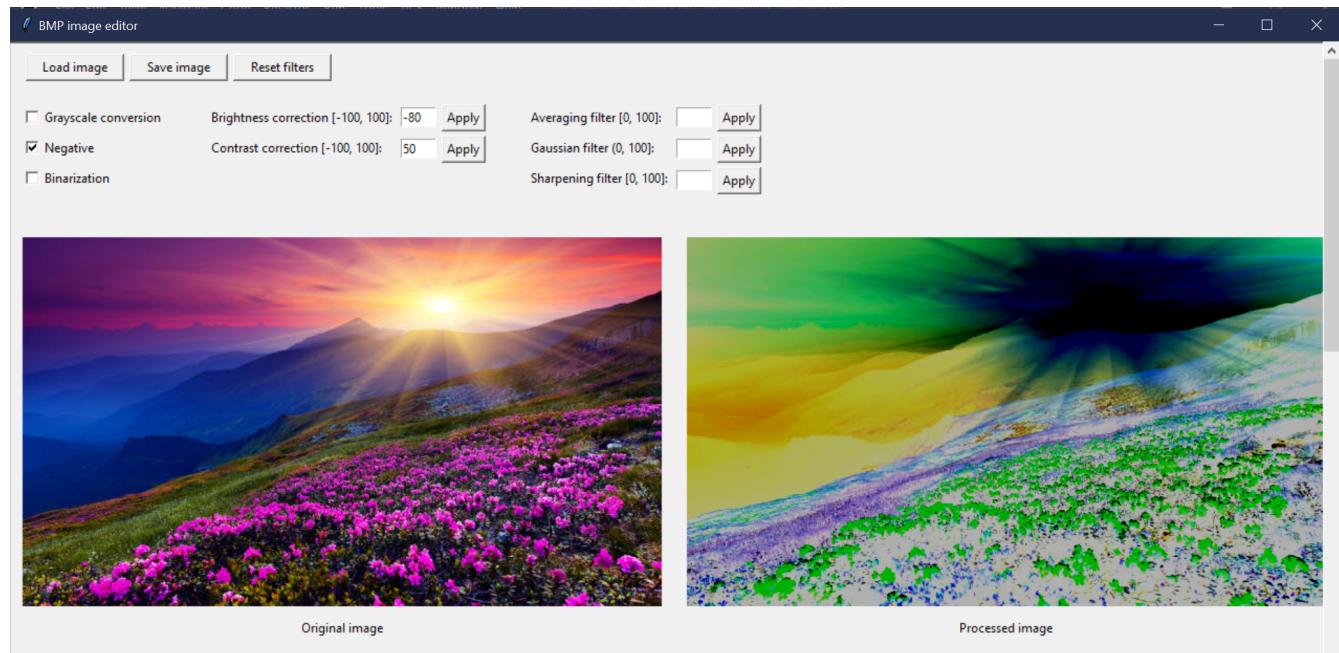


Rysunek 3: Histogramy dla: jasność 20, kontrast -80, filtr Gaussa 20

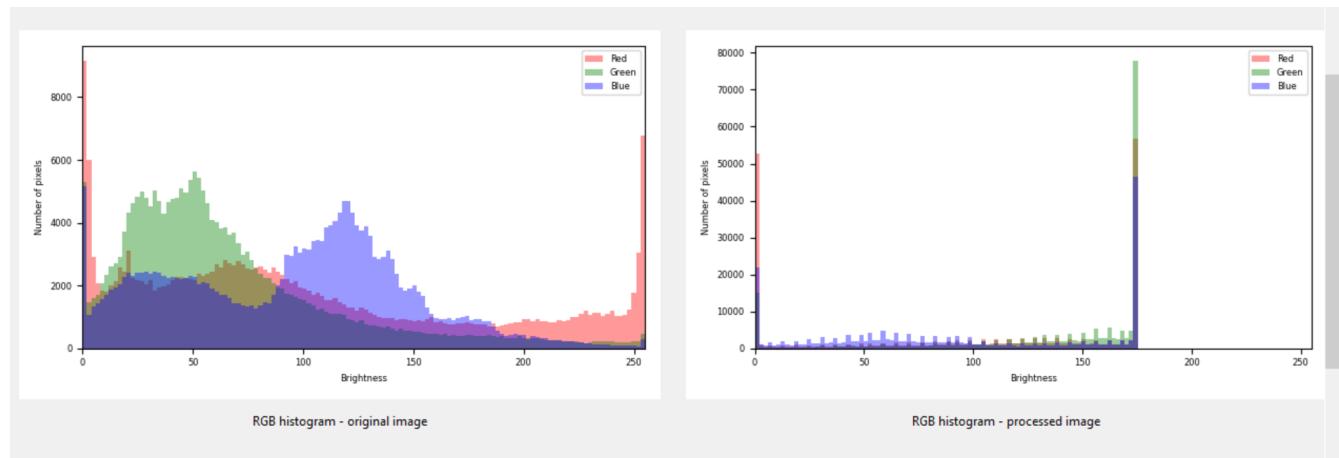
4.2 Przykład 2

W tym przykładzie nałożone zostały filtry: `brightness_correction = -80`, `contrast_correction = 50` oraz `negative`.

Poniżej przedstawiony jest wygląd obrazka z nałożonymi filtrami oraz histogramy dla nich.



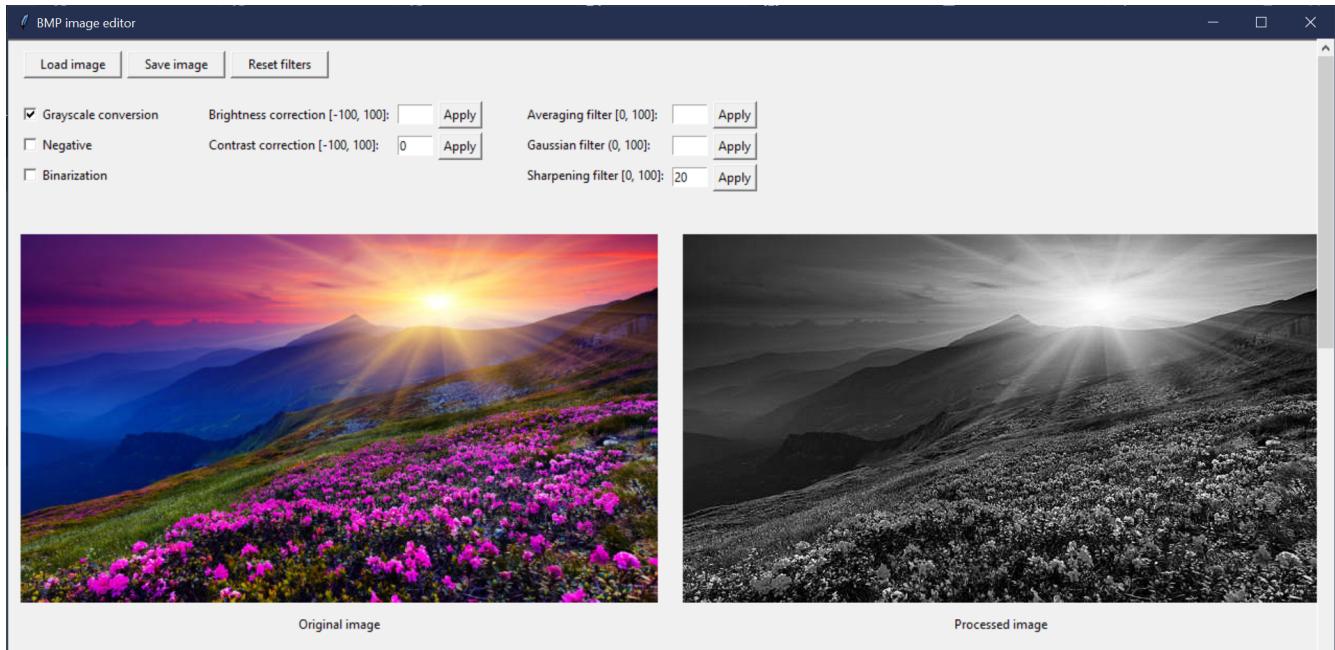
Rysunek 4: Wygląd obrazków dla: negatyw, jasność -80, kontrast 50



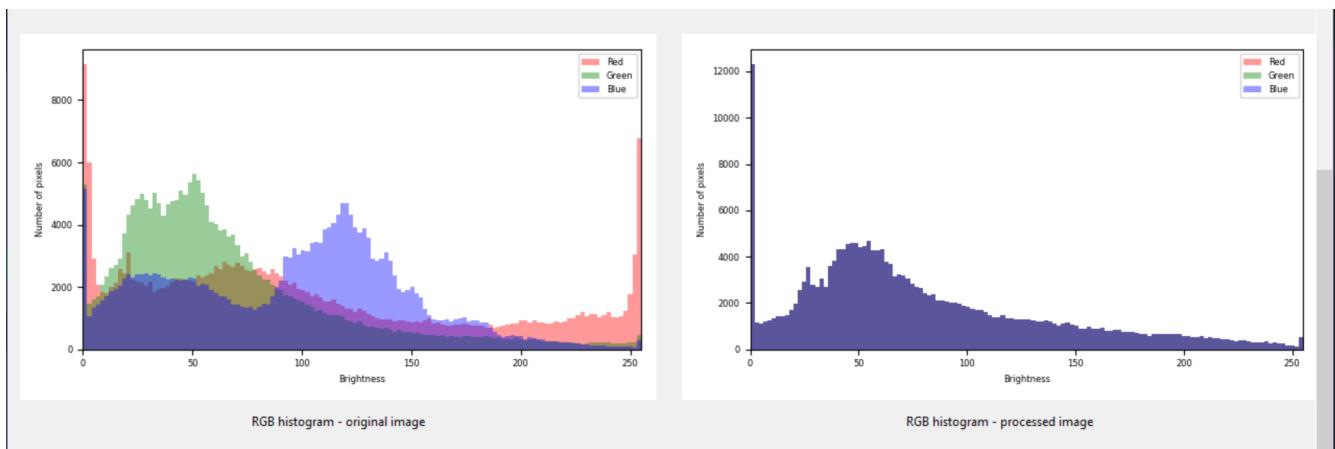
Rysunek 5: Histogramy dla: jasność -80, kontrast 50, negatyw

4.3 Przykład 3

W tym przykładzie nałożone zostały filtry: `grayscale_conversion` oraz `sharpening_filter = 20`. Poniżej przedstawiony jest wygląd obrazka z nałożonymi filtrami oraz histogramy dla nich.



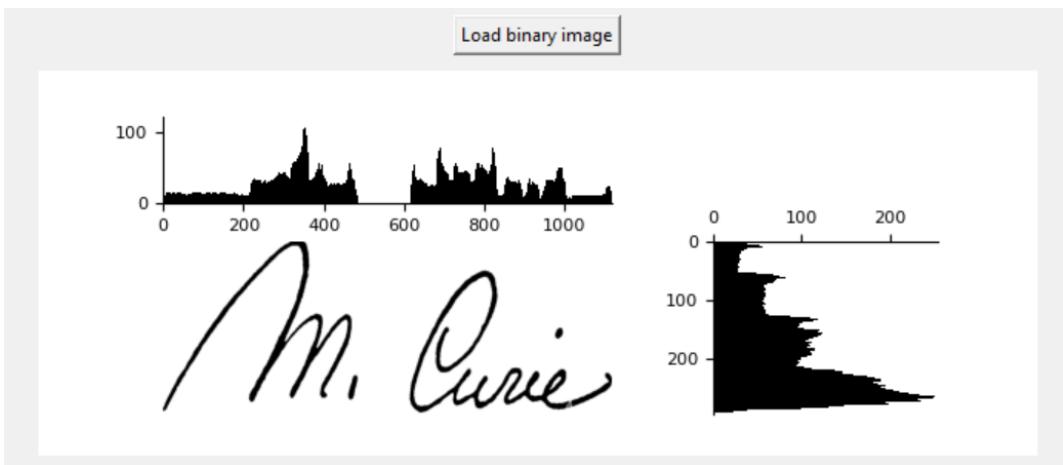
Rysunek 6: Wygląd obrazków dla: skali szarości, filtra uśredniającego 20



Rysunek 7: Histogramy dla: skali szarości, filtra uśredniającego 20

4.4 Projekcje

Poniżej zaprezentowane jest działanie naszej aplikacji dla obrazka binarnego przedstawiającego odręczny podpis.



Rysunek 8: Projekcje dla odreźnego podpisu

5 Wnioski

Na podstawie przeprowadzonych testów można stwierdzić, że aplikacja działa poprawnie i stabilnie w większości przypadków. Proces nakładania filtrów jak i inne funkcjonalności przebiegają płynnie i bezproblemowo. Dodatokowo dzięki wyświetlaniu obok siebie zdjęcia oryginalnego i产生的ego (po nałożeniu filtrów), użytkownik może łatwo porównywać, jak działa dany filtr.

Jedyną zauważalną niedogodnością jest wydłużony czas działania aplikacji w sytuacjach, gdy na obraz nakładana jest duża liczba filtrów jednocześnie. W takich przypadkach może być konieczne krótkie oczekiwanie na zakończenie przetwarzania.

Ogólnie rzecz biorąc, aplikacja spełnia swoje założenia funkcjonalne i może być z powodzeniem wykorzystywana w praktycznych zastosowaniach.