



Akademia  
Techniczno-Humanistyczna  
w Bielsku-Białej

## PRACA DYPLOMOWA

**WYDZIAŁ  
BUDOWY MASZYN I INFORMATYKI**  
KIERUNEK: Informatyka  
SPECJALNOŚĆ: Inżynieria Oprogramowania

BARTOSZ DOBIJA

nr albumu: 057151

Praca INŻYNIERSKA

**APLIKACJA Z GRAFICZNYM INTERFEJSEM UŻYTKOWNIKA**

**DO PRZETWARZANIA OBRAZÓW**

Kategoria pracy: PRACA PROJEKTOWA

Promotor: mgr inż. ŁUKASZ HAMERA

Bielsko-Biała, rok akademicki 2022 /2023

A handwritten signature in blue ink, appearing to read 'Lukasz Hamera'.



# Spis treści

<b>Wstęp</b>	<b>6</b>
<b>Cel i zakres pracy</b>	<b>7</b>
<b>1 Część teoretyczna</b>	<b>8</b>
1.1 Rys historyczny . . . . .	8
1.1.1 Wywoływanie zdjęć . . . . .	8
1.1.2 Transmisja obrazów . . . . .	9
1.1.3 Cyfrowe przetwarzanie obrazów . . . . .	9
1.2 Przegląd istniejących rozwiązań . . . . .	10
1.2.1 Photoshop . . . . .	10
1.2.2 ImageJ . . . . .	11
1.2.3 Fiji . . . . .	12
1.2.4 GIMP . . . . .	13
1.2.5 Adobe Substance Designer . . . . .	14
<b>2 Zakres użytych technologii i opis wykorzystywanych narzędzi</b>	<b>15</b>
2.1 Język programowania . . . . .	15
2.1.1 C# . . . . .	16
2.2 Platforma programowa . . . . .	16
2.2.1 .NET Core . . . . .	16
2.3 Interfejs użytkownika . . . . .	16
2.3.1 Windows Presentation Foundation . . . . .	17
2.3.2 Nodify . . . . .	17
2.3.3 WPF UI . . . . .	18
2.4 Biblioteka do przetwarzania obrazów . . . . .	19

2.4.1	OpenCvSharp . . . . .	19
2.5	Biblioteka do walidacji . . . . .	19
2.5.1	Fluent Validation . . . . .	19
2.6	Biblioteka do serializacji . . . . .	20
2.6.1	Json.NET . . . . .	20
2.7	Środowisko deweloperskie . . . . .	20
2.7.1	Microsoft Visual Studio . . . . .	21
2.7.2	Git . . . . .	21
<b>3</b>	<b>Realizacja projektu</b>	<b>22</b>
3.1	Zastosowania . . . . .	22
3.2	Analiza wymagań projektu . . . . .	22
3.2.1	Wymagania funkcjonalne . . . . .	22
3.2.2	Wymagania niefunkcjonalne . . . . .	23
3.3	Architektura aplikacji . . . . .	23
3.3.1	Model . . . . .	24
3.3.2	View . . . . .	26
3.3.3	ViewModel . . . . .	30
3.4	Zaimplementowane operacje . . . . .	33
3.4.1	Blur . . . . .	33
3.4.2	ChangeColorspace . . . . .	34
3.4.3	Crop . . . . .	35
3.4.4	EdgeDetect . . . . .	36
3.4.5	Resize . . . . .	37
3.5	Przykładowe ciągi operacji . . . . .	38
<b>4</b>	<b>Podsumowanie</b>	<b>40</b>

<b>Bibliografia</b>	<b>41</b>
<b>Spis rysunków</b>	<b>45</b>
<b>Spis tabel</b>	<b>47</b>

# Wstęp

Przetwarzanie obrazów to zbiór wielu algorytmów które należy rozróżnić na dwie dziedziny ze względu na rodzaj reprezentowanych i manipulowanych informacji. Pierwsza dotyczy analogowej reprezentacji danych [1], gdzie informacje są w formie ciągłych sygnałów odwzorowujących obrazy. Operacje na nich są realizowane przez modyfikacja impulsów elektrycznych reprezentujących informacje. Technologia ta umożliwiła rozwój transmisji telewizyjnej [2]. Cyfrowe przetwarzanie obrazów [3] jest nowszą technologią. Dane są reprezentowane po przez wielowymiarowe macierze wartości [4]. Mogą one odpowiadać kolorowi obrazu w danym miejscu, wartościom chmury punktów lub innym informacjom które mogą zostać przetworzone cyfrowo. Modyfikacja odbywa się przez zmiany wartości macierzy. Obie dziedziny mają swoje miejsca i zastosowania ale niniejsza praca zajmie się cyfrowym przetwarzaniem obrazów.

Przetwarzania obrazów jest wykorzystywane w wielu różnych dziedzinach:

- **Medycyna:** Diagnostyka, planowanie zabiegów chirurgicznych i monitorowanie stanu pacjentów [5]. Dane medyczne uzyskiwane w formie obrazów cyfrowych są ustandaryzowane przez Digital Imaging and Communications in Medicine [6].
- **Przemysł:** Kontrola jakości, automatyzacja i inspekcja. Systemy wizyjne potrafią szybciej i dokładniej odkryć defekty produktów na liniach produkcyjnych [7].
- **Bezpieczeństwo:** Monitorowanie i rozpoznawanie twarzy. Drony ze wsparciem przetwarzania obrazów są używane przez służby bezpieczeństwa w trudnych warunkach [8].
- **Rozrywka:** Tworzenie efektów specjalnych dla filmów, gier i animacji. Transformacje przestrzeni kolorów i korekcje nagrań w celu poprawienia efektu końcowego [9].

Przetwarzanie obrazów jako manipulacja danymi cyfrowymi została stworzona przez programistów [10]. Kod to podstawowa wersja interakcji człowieka z komputerem, ale większość użytkowników tych urządzeń nie potrafi programować. Żeby dziedzina tej nauki była przystępna dla większej liczby osób należy stworzyć aplikację z prostym i funkcjonalnym graficznym interfejsem użytkownika.

W tej pracy naukowej proponujemy nowe oprogramowanie do przetwarzania obrazów oparte na bibliotece OpenCV. Aplikacja posiada funkcjonalny interfejs graficzny, który ułatwia użytkownikom końcowym obsługę tej biblioteki bez potrzeby znajomości programowania.

# Cel i zakres pracy

Celem tej pracy jest zaprojektowanie i implementacja oprogramowania do przetwarzania obrazów z funkcjonalnym interfejsem graficznym.

Zakres pracy objął analizę istniejących rozwiązań na podstawie której stwierdzono, że nie ma aktualnego i gotowego produktu który realizuje dokładnie założenie zastąpienia programistycznego podejścia do obsługi bibliotek przetwarzania obrazów. Jest wiele rozwiązań które realizuje część biblioteki lub jej obsługa nie pozwala na przetwarzanie obrazów w niedestruktacyjny sposób bez dodatkowego wysiłku od użytkownika jak używanie warstw czy inkrementalnych zapisów.

Jako platformę wybrano komputer osobisty z powodu znaczącej mocy obliczeniowej, dostępu do dużej ilości pamięci RAM oraz do operacji procesorów x86 z których biblioteki do przetwarzania obrazów czasem korzystają do akceleracji obliczeń [11]. Jako docelowy system operacyjny wybrano Microsoft Windows 11 ponieważ jest to system używany przez autora tej pracy. Do tworzenia aplikacji skorzystano z języka C# oraz Windows Presentation Framework.

W ramach czasu, który został poświecony na niniejszy projekt zakres możliwości ograniczono do prostych operacji przetwarzania obrazów bez uwzględniania wideo oraz wszelkiego rodzaju danych oprócz samych obrazów.

Pomysł na ten projekt powstał dzięki doświadczeniu autora pracy w używaniu różnych systemów opartych o edytory typu węzlowego jak Adobe Substance Designer, edytor Blueprints w Unreal Engine oraz edytor shaderów czy geometry nodes w Blenderze. Ten typ interfejsu pozwala na interakcję użytkownika z wizualnym przedstawieniem jakichś metod, funkcji czy bloków logicznych pozwalających na przesyłanie wartości w podobny sposób jak można robić to w kodzie, ale zdecydowanie bardziej przystępne dla osób które nie mają odpowiedniego doświadczenia w programowaniu.

Dużą zaletą tego podejścia w porównaniu do edycji w programach posługujących się warstwami jest to, że możemy wrócić w każdym momencie do dowolnej poprzedniej operacji. Użytkownik może zmienić wartość operacji, zmienić ich kolejność i w żadnym momencie nie traci wyniku operacji, ponieważ każda zmiana powoduje obliczenie rezultatu jeszcze raz. Przeliczenie całego drzewa na nowym obrazie wejściowym i otrzymanie wyniku którego się spodziewamy to zmiana jednego parametru.

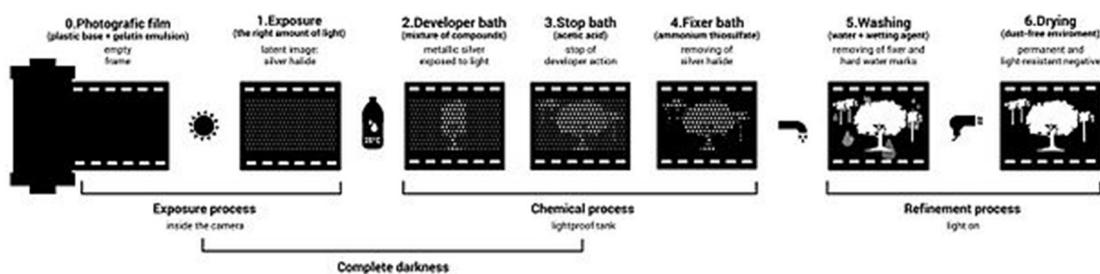
# 1 Część teoretyczna

## 1.1 Rys historyczny

Dziedzina tej pracy jaką jest przetwarzanie obrazów ma początki swojej historii wcześniej niż urządzenia które obecnie nazywamy komputerami.

### 1.1.1 Wywoływanie zdjęć

Przetwarzanie obrazów to nie jest technologia, która mogła zacząć istnieć po wynalezieniu komputerów. Używając aparatów korzystających z kliszy filmowej, po ekspozycji należy go poddać procesowi wywoływania Rys. 1. Polega on na wyciągnięciu pierwotnego efektu naświetlenia do obrazu, który oddaje scenę uchwyconą przez aparat [12]. Proces w przypadku zdjęć aparatem na kliszę polega na zanurzaniu jej w odpowiednich związkach chemicznych na określone ilości czasu by otrzymać zamierzony efekt.



Rys. 1: Wywoływanie biało czarnej kliszy [13].

Istnieją wariacje na temat takiego przetwarzania, różni się ono trochę w zależności od technologii kliszy. W niektórych przypadkach należy wywołać pozytyw zamiast negatywu [14]. Następnie po wywoływaniu można poddać obróbce dalszymi chemikaliami jak np. siarczek sodu dla uzyskania efektu sepii [15].

### **1.1.2 Transmisja obrazów**

Technologia którą znamy dobrze w obecnych czasach powstała w latach 20. XX wieku dzięki pracy John Logie Baird'a [2]. Zaprezentował on pierwszą na świecie transmisję telewizyjną na żywo. W 1928 roku [16] firma założona przez niego emitowała też pierwsze nagranie przez atlantyk. Wszystko to dzięki przetwarzaniu analogowego sygnału video w celu zmniejszenia ilości danych potrzebnych do wyświetlenia obrazu w telewizji [17].

### **1.1.3 Cyfrowe przetwarzanie obrazów**

Początki nowoczesnego przetwarzania obrazów zostały stworzone w latach 60. XX w. w Bell Laboratories, Jet Propulsion Laboratory, Massachusetts Institute of Technology i University of Maryland [10]. Początkowymi obszarami zastosowania były obrazy satelitarne, przesyłanie obrazów przez linie telefoniczne, diagnostyka obrazowa, videofony, rozpoznawanie znaków i ulepszanie fotografii.

Inicjalnie dużo skupiano się na ulepszeniu jakości obrazu. Pierwszym znaczącym użyciem tych technologii było mapowanie powierzchni księżyca za pomocą zdjęć z sondy kosmicznej w 1964 roku, gdzie naukowcy z Jet Propulsion Laboratory na podstawie tysięcy zdjęć odtworzyli powierzchnię księżyca. Przy następnej okazji mieli dostęp do 100000 zdjęć, na podstawie których mogli stworzyć mapę topograficzną, mapę kolorową oraz panoramiczną mozaikę księżyca, które przyczyniły się do pierwszego lądowania człowieka na księżycu [3].

Technologia ta jednak była ograniczona przez bardzo małe możliwości oraz trudną dostępność komputerów tamtych czasów. Wraz z ich rozwojem i wzrostem popularności cyfrowe przetwarzanie obrazów przestało być trudno dostępną dziedziną naukową i stopniowo zostało wdrażane do naszego codziennego życia. Obecnie około 80% populacji krajów rozwiniętych jest w posiadaniu smartfona [18], który na swoim wyposażeniu ma kamerę cyfrową. Są one ograniczone poprzez fizyczny rozmiar matrycy oraz jakość soczewek, które mogą zmieścić się w telefonach komórkowych. Pomimo tego dzięki przetwarzaniu obrazów można uchwycić za ich pomocą imponujące zdjęcia [19]. Nowoczesne telefony są w stanie zachować na zdjęciu nocnym detale które może być ciężko zobaczyć ludzkim okiem [20] przez szybkie łączenie wielu obrazów o różnych długościach ekspozycji, redukcji szumów i korekcji kolorów.

## 1.2 Przegląd istniejących rozwiązań

### 1.2.1 Photoshop



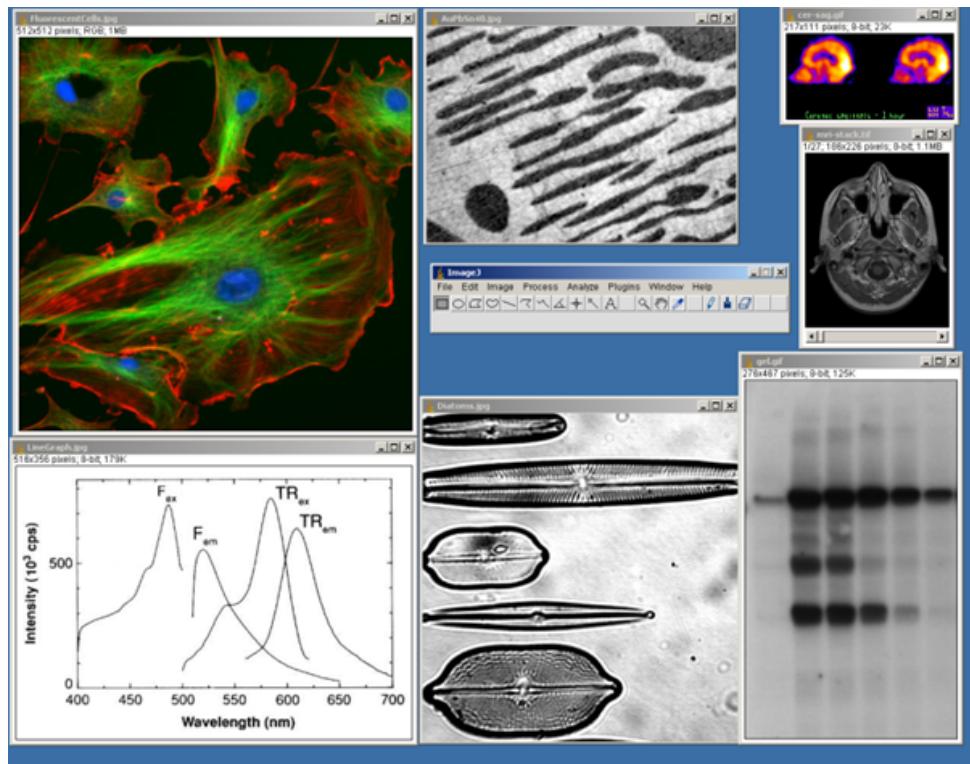
Rys. 2: Adobe Photoshop [21].

Wydany w 1990 roku do tworzenia i edytowania obrazów rastrowych jest jednym z najbardziej popularnych programów, a jego nazwa przedarła się do języka potocznego jako zamiennik dla fotomontażu. Użytkownikami tego oprogramowania są przeważnie artyści, fotografowie i twórcy internetowych memów. To narzędzie jest przystosowane do łatwości obsługi przez mniej zaawansowanych użytkowników i wszystko posiada przyjazny graficzny interfejs użytkownika.

Funkcje, które pozwalają na przetwarzanie obrazów często nie odnoszą się bezpośrednio do operacji zawartych w znanych bibliotekach do przetwarzania obrazów. Ich obsługa jest uproszczona i przystosowana do bardziej artystycznych zastosowań. Bez zastosowania szczególnej ostrożności - jak tworzenie nowych warstw po każdej operacji lub częste zapisywanie kopii zapasowej obrazu – przetwarzanie obrazów często jest destruktywne, nie możemy odnieść się do dowolnego momentu w ciągu naszych operacji w celu dopasowania jej parametrów. Przetwarzanie innego obrazu za pomocą tego samego procesu jest problematyczne, gdyż

wymaga to ciągłego śledzenia warstw. Wiele operacji trzeba powtórzyć indywidualnie ponieważ nie zapisujemy jej parametrów tylko rezultat poprzedniego wykonania. Wiele bardziej zaawansowanych procesów nie jest możliwych do zrealizowania za pomocą standardowych opcji dostępnych w programie. Historia operacji jest też ograniczona i nie można wyświetlić pełnej historii na jednym ekranie - można jedynie zamieniać stan aktualny na wcześniejszy.

### 1.2.2 ImageJ



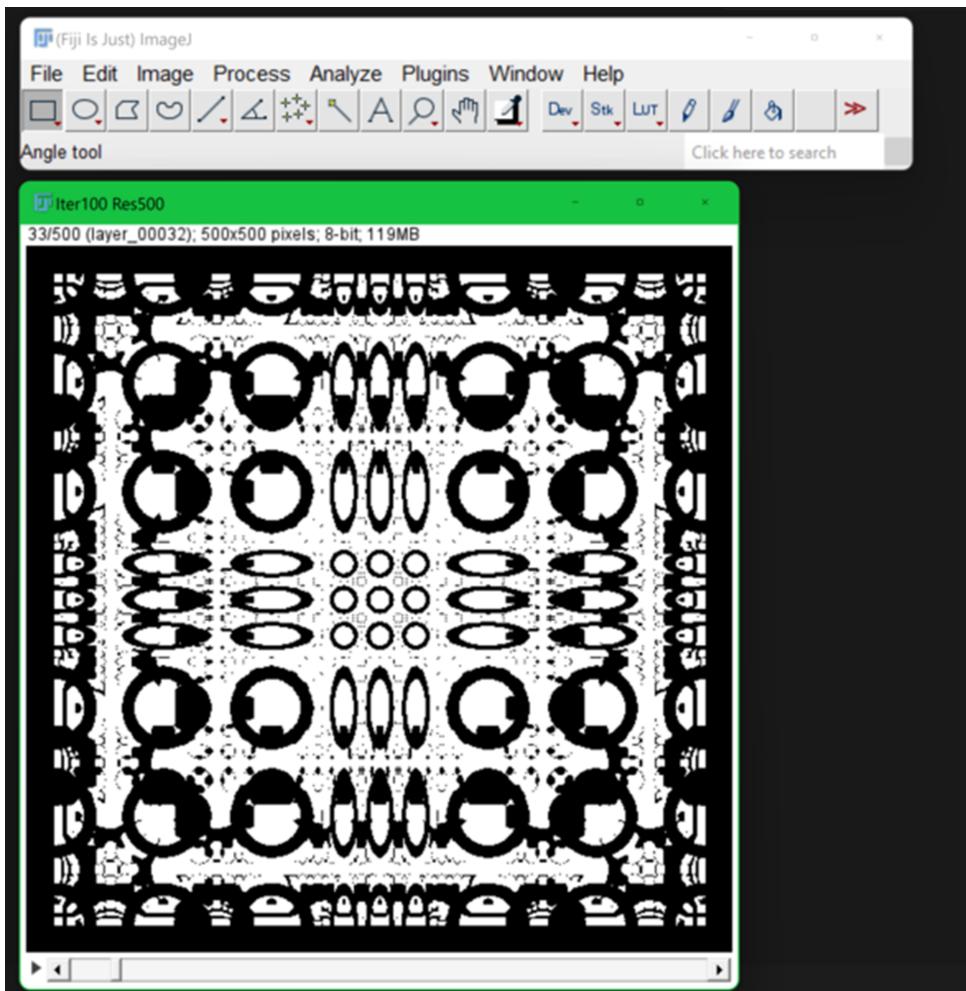
Rys. 3: ImageJ [22].

Wydany w 1997 roku program ImageJ został wykonany przez National Institutes of Health, został tworzony podstawowo z zamiarem analizy obrazów w zastosowaniach medycznych. Odtwarza dokładnie wiele operacji z bibliotek do przetwarzania obrazów i kod aplikacji jest otwartoźródłowy - każdy może go zobaczyć, pomóc w rozwoju lub stworzyć własną wersję. Napisany został w języku Java i ma bardzo niskie wymagania sprzętowe jak na obecne czasy.

Program jest jednak przestarzały, mogą pojawić się problemy z uruchomieniem na nowszzych systemach. Interfejs użytkownika wygląda archaicznie oraz jego układ jest niepraktyczny biorąc pod uwagę przetwarzanie obrazów. Wszystkie operacje są schowane pod 1-2 poziomami

menu. Operacje są destrukcyjne - zmieniają dane, które przetwarzamy więc użytkownik musi pilnować aby mieć kopię swoich obrazów. Nie da się przygotować ciągów operacji wcześniej za pomocą interfejsu graficznego, trzeba użyć do tego ich języka programowania ImageJ Macro [23]. Od dawnego nie jest rozwijany, został zastąpiony przez ImageJ2, który pozwala na przetwarzanie obrazów wielowymiarowych z dodatkowymi danymi np. z mikroskopów elektronowych, skanerów itp.

### 1.2.3 Fiji

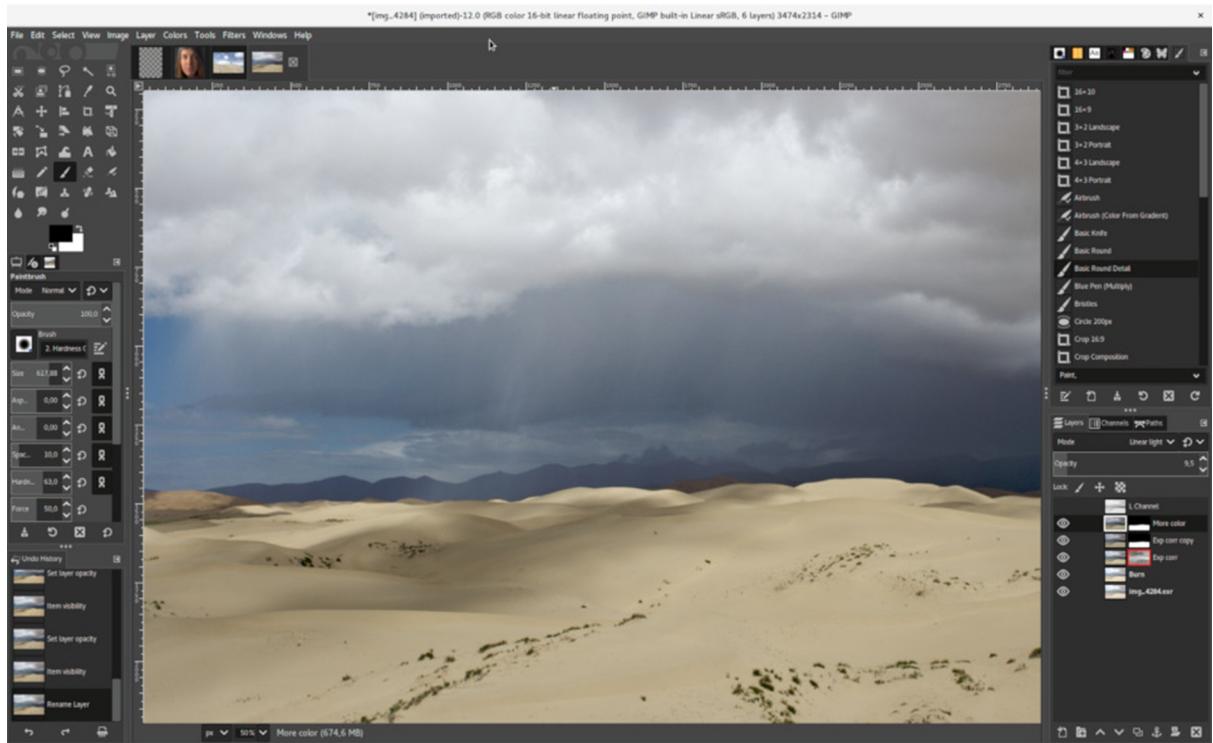


Rys. 4: Fiji - sukcesor ImageJ [24].

Projekt open source oparty o ImageJ2. Oprócz podstawowych operacji wbudowanych w ImageJ posiada też wiele pluginów znacznie rozszerzających możliwości programu. Są one skupione na wspomaganiu przetwarzania obrazów skupionych na dziedzinie neurobiologii, ale możliwości są na tyle szerokie, że wiele innych dziedzin nauki z niego korzysta.

Podstawowe działanie programu nie różni się od ImageJ więc, wszystkie jego problemy są też tutaj obecne. Jednakże poprawiona została jego kompatybilność z najnowszymi systemami operacyjnymi.

#### 1.2.4 GIMP

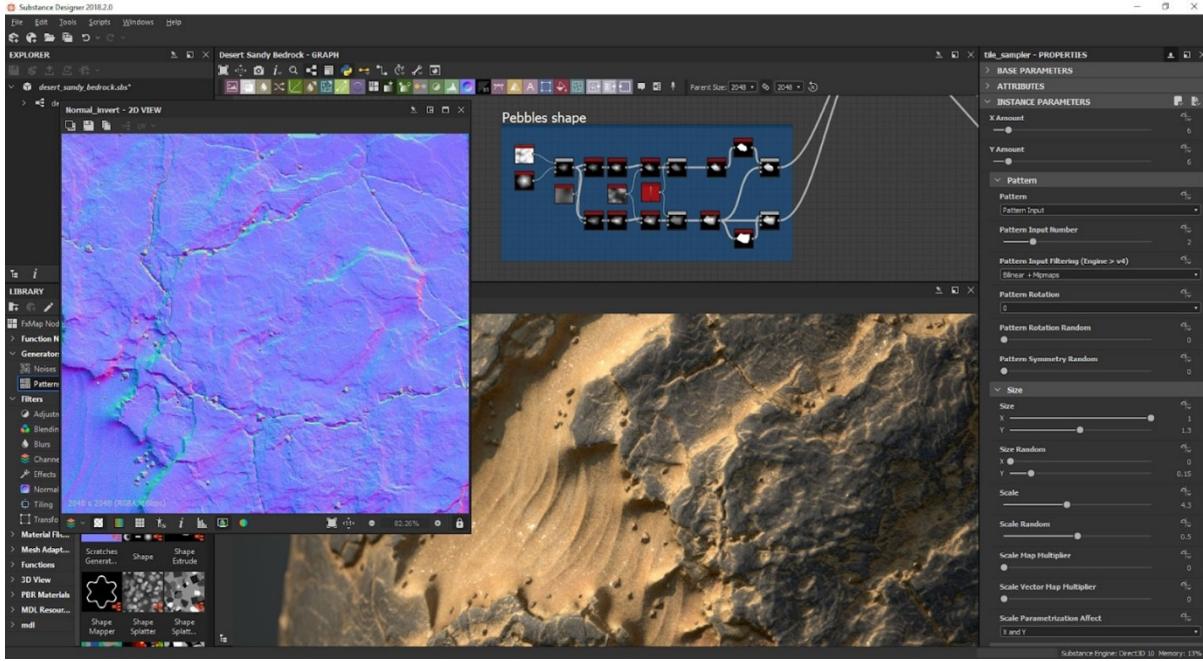


Rys. 5: GIMP wersja 2.10 [25].

Wydany w 1998 roku GNU Image Manipulation Program jest tworzony jako darmowa konkurencja dla Photoshopa, którego licencja jest miesięczną subskrypcją. Program też w pierwszej kolejności jest tworzony z myślą o artystach, ale posiada więcej zaawansowanych opcji jak np. konwolucje macierzowe które można dowolnie edytować, dając duże możliwości filtrowania obrazów.

Pomimo odwzorowaniu większej liczby operacji ze standardowych bibliotek do przetwarzania obrazów i większej kontroli nad niektórymi z nich nadal występuje problem destruktywnego przetwarzania obrazów. Wynika to z pracy bezpośrednio na obrazie jak w Photoshopie i wymaga szczególnej uwagi przy zarządzaniu warstwami aby nie tracić bezpowrotnie swoich pośrednich operacji w dłuższym ciągu.

### 1.2.5 Adobe Substance Designer



Rys. 6: Adobe Substance Designer [26].

Wydany w 2007 roku program był główną inspiracją dla tego projektu. Jego dużą zaletą jest to, że tworzenie algorytmów bazuje na układaniu bloczków (nodów). Ułożony graf jest wykonywany na obrazie, który importujemy do programu lub generujemy go od początku w nim. Każdy dowolny node można kliknąć i zmienić wszystkie parametry jego operacji, niezależnie w którym miejscu ciągu znajduje się. Jego wynik i wszystkie następne operacje, które są zależne od niego są obliczane ponownie na podstawie zmienionego wyniku. Zmiana przetwarzanego obrazu polega na przeciągnięciu połączenia z obecnego węzła z naszym plikiem wejściowym na nowy bloczek. Następnie propagowana jest zmiana na wszystkie kolejne operacje, których wynik jest aktualizowany automatycznie.

Opisywany program ten ma wiele ograniczeń związanych z tym, że nie jest stworzony do ogólnego przetwarzania obrazów. Został natomiast stworzony do tworzenia materiałów/tekstur do grafiki komputerowej. Obrazy są ograniczone do boków o długości  $2^n$ , najlepiej kwadratowych. Parametry operacji są często uproszczone, ponieważ mimo większej złożoności aplikacji, jest ona nadal skierowana do artystów. Rodzaje operacji i wspierane formaty pikseli w pliku są przystosowane do wymagań grafiki komputerowej.

## 2 Zakres użytych technologii i opis wykorzystywanych narzędzi

Wybór technologii ma duży wpływ na różne aspekty projektu programistycznego. Od niego zależy na jakich platformach może zostać on tworzony i uruchamiany. Dobre biblioteki potrafią znaczco ułatwić tworzenie oprogramowania pozwalając skupić się programistom na tworzeniu funkcjonalności aplikacji. Nie ma potrzeby tworzyć nowego systemu wyświetlania interfejsu użytkownika na potrzebę każdego projektu osobno, gdy jest wybór wielu bardzo dobrze wspieranych technologii na każdą platformę.

Przy wyborze technologii dla aplikacji oprócz funkcjonalności zostały wzięte pod uwagę:

- **Dobrze napisaną dokumentacją:** Obsługa biblioteki programistycznej jest dużo łatwiejsza, gdy jej twórca udostępnia dobrze napisane materiały jak z niej korzystać.
- **Otwarty kod źródłowy:** Projekty open source udostępniają swój kod - nawet w momencie, kiedy dokumentacja nie jest idealna możemy zobaczyć jak dana funkcja działa.
- **Aktywne utrzymywanie:** Biblioteka bez aktualizacji do nowszych wersji może sprawiać problemy w postaci luk bezpieczeństwa lub problemami z kompatybilnością.
- **Dojrzała technologia:** Korzystając z kodu tworzonego i używanego od wielu lat można spodziewać się większej ilości przykładowych użyci, dopracowanego interfejsu projektu oraz pozbawionego błędów działania.

### 2.1 Język programowania

Jest to kluczowy wybór, ma wpływ na większość następnie wybranych technologii. Język programowania ma wpływ na wydajność aplikacji, trudność jej tworzenia oraz późniejszego utrzymania. Dzięki wyborze popularnego języka można mieć dostęp do bardzo szerokiej gamy gotowych bibliotek rozwiązujecych popularne problemy jak komunikacja z serwerami czy zapisywanie i odczytywanie formatu JSON.

### **2.1.1 C#**

Wieloparadygmatowy język ogólnego przeznaczenia tworzony przez Microsoft od ponad 20 lat [27]. Jest on kompilowany przed uruchomieniem co daje przewagę wydajności nad językami interpretowanymi. Można obecnie pisać w nim aplikacje na komputery stacjonarne, telefony, przeglądarki czy serwery. Posiada narzędzie do zarządzania bibliotekami NuGet [28] pozwalającą na łatwe dodanie zależności do projektu.

## **2.2 Platforma programowa**

Wiele języków programowania posiada swoją platformę programistyczną - jest to narzędzie za pomocą którego można tworzyć projekty, zarządzać bibliotekami, budować i uruchamiać aplikację.

### **2.2.1 .NET Core**

.NET Core to platforma programistyczna opracowana przez firmę Microsoft, stanowiąca otwartoźródłowe i wieloplatformowe środowisko do tworzenia różnorodnych typów aplikacji, w tym aplikacji internetowych, usług sieciowych oraz aplikacji konsolowych. Obsługuje wiele języków z których najpopularniejszy jest C# [29]. Jej struktura podzielona jest na moduły takie jak środowisko uruchomieniowe, kompilacja czy wbudowane biblioteki.

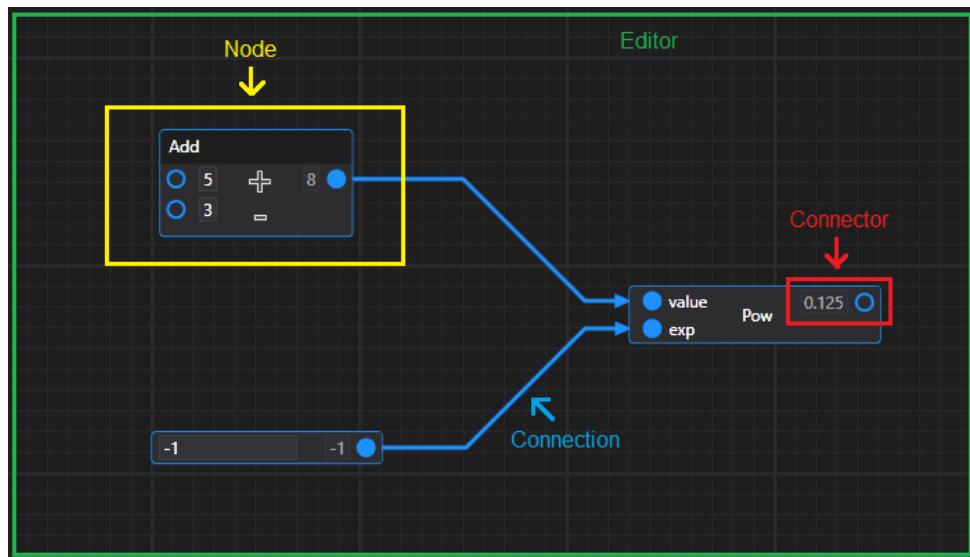
## **2.3 Interfejs użytkownika**

W celu zapewnienia użytkownikom szybkiego wykonywania operacji wybrano jako docelową platformę komputer stacjonarny. Zawęża to wybór możliwych technologii tworzenia interfejsu. Rozważone zostały dwie technologie - Avalonia UI [30] oraz WPF [31]. Pierwsza z nich jest nowsza i aktywnie rozwijana, może działać na wielu platformach w przeciwieństwie do rozwiązania stworzonego przez Microsoft. W momencie decydowania o interfejsie użytkownika jednak pojawiały się błędy w funkcjonalnościach potrzebnych do działania projektu.

### 2.3.1 Windows Presentation Foundation

Jest to platforma graficzna opracowana przez firmę Microsoft, została po raz pierwszy wprowadzona jako część .NET Framework w 2006 roku [32]. Umożliwia tworzenie bogatych interfejsów użytkownika dla aplikacji stworzonych na system Windows. Elementy są wyświetlane wektorowo co pozwala na dowolne skalowanie elementów. Widoki są definiowane w formacie XAML co pozwala na oddzielenie logiki biznesowej od warstwy prezentacji. Platforma ta umożliwia powiązanie elementów interfejsu z danymi aplikacji co pozwala na szybkie aktualizowanie widoku z najnowszymi wynikami.

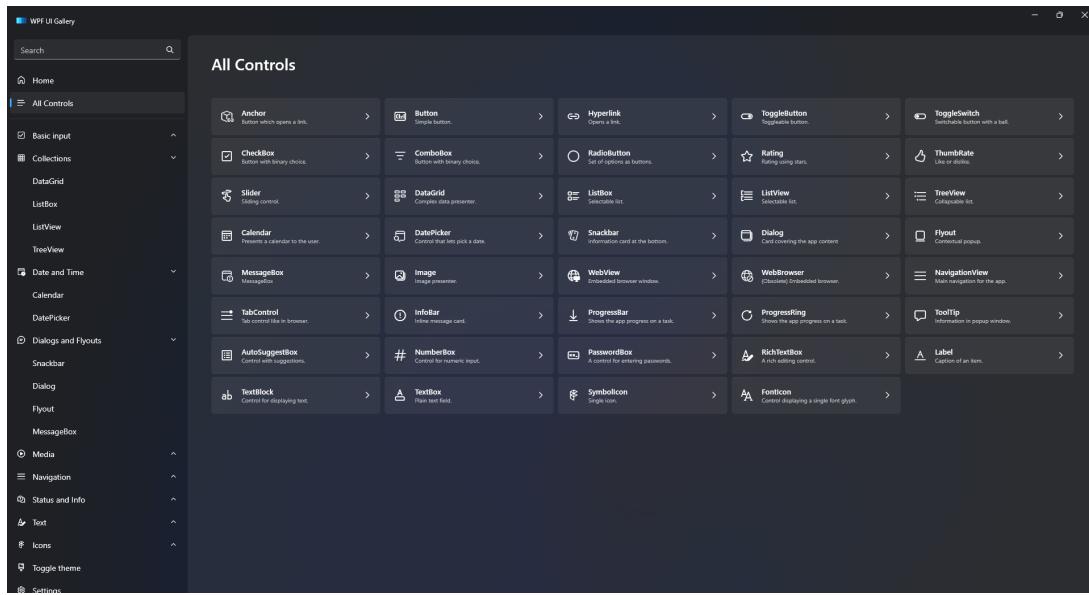
### 2.3.2 Nodify



Rys. 7: Opis elementów edytora węzłów [33].

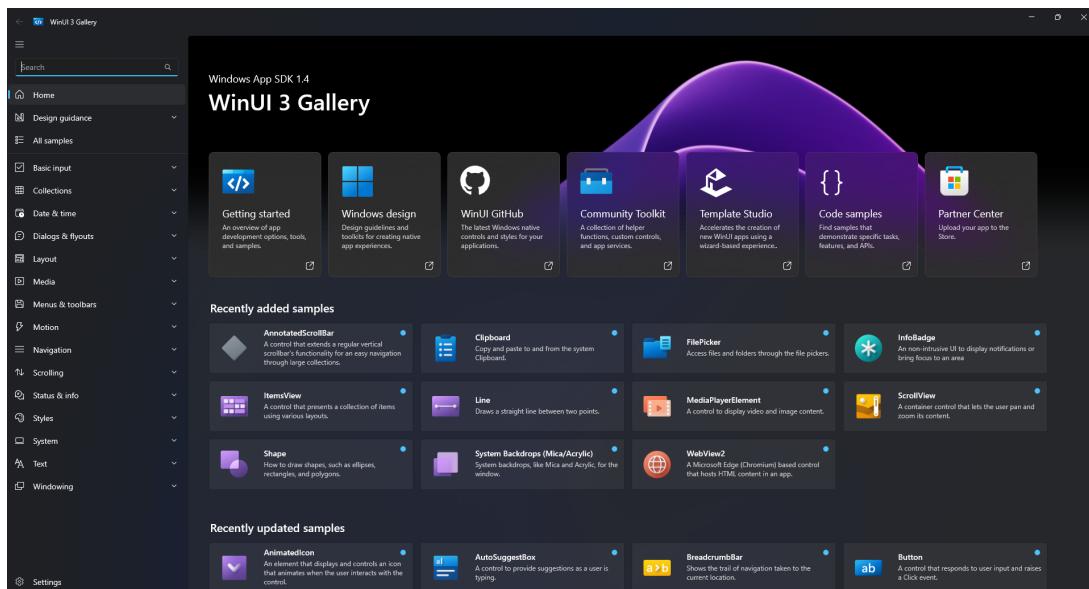
Biblioteka Nodify [34] jest otwartoźródłowym projektem implementującym edytor węzłowy Rys. 7 dla biblioteki WPF. Poprawna implementacja tego projektu daje możliwość dodawania bloczków i łączenie ich w ciągi. Wystarczy zaimplementować co dzieje się z danymi w węzłach oraz jak zostają one przekazywane dalej żeby mieć funkcjonalny interfejs użytkownika.

### 2.3.3 WPF UI



Rys. 8: Galeria elementów WPF UI. Opracowanie własne.

Biblioteka WPF UI [35] modyfikuje i dodaje elementy interfejsu używane w Windows Presentation Foundation Rys. 8. Jest ona zgodna z systemem projektowania interfejsu Fluent 2 Rys. 9 od Microsoftu używanego w wielu aplikacjach dla systemu Windows 11. Pozwala to na utrzymanie nowoczesnej stylistyki w projekcie bez potrzeby tworzenia skomplikowanych stylów imitujących WinUI 3 od zera.



Rys. 9: Galeria elementów WinUI 3. Opracowanie własne.

## 2.4 Biblioteka do przetwarzania obrazów

Jedną z popularnych bibliotek do przetwarzania obrazów jest OpenCV. Nazwa to skrót od *Open Source Computer Vision Library* i wskazuje, że jest to darmowy i otwartoźródłowy projekt skupiony na przetwarzaniu obrazów oraz widzeniu maszynowym. Pakiet ten dostępny jest dla systemów Windows, macOS i Linux. Został stworzony w Intel Research Labs w 1999 roku [36]. Biblioteka ta jest napisana w języku C++ i kładzie duży nacisk na prędkość obliczeń. Pozwala ona na przetwarzanie obrazów, wideo, posiada zaawansowane algorytmy wykrywania obiektów, klasyfikacji oraz wiele operacji korzystających z sieci neuronowych.

### 2.4.1 OpenCvSharp

Projekt jest tworzony w języku C# więc nie możemy bezpośrednio odwoływać się do biblioteki OpenCV. Powstało kilka projektów które opakowują oryginalny pakiet w metody którymi możemy się odwołać bez problemu z kodu w platformie .NET. Dwie najpopularniejsze biblioteki tego typu to EmguCV [37] oraz OpenCvSharp [38]. Obie cieszą się dużym wsparciem społeczności lecz pierwsza z nich jest mniej zgodna z API oryginalnej paczki oraz jest wolniejsza.

## 2.5 Biblioteka do walidacji

W interfejsie użytkownika należy wprowadzić dane, nie możemy założyć że będą one poprawne. Wszystkie wejścia trzeba zabezpieczyć tak by użytkownik nie mógł przerwać działania programu i zawsze wiedział dlaczego jego operacje się nie wykonują.

### 2.5.1 Fluent Validation

Biblioteka Fluent Validation [39] pozwala na proste ustalanie reguł według których ocenia czy dane są prawidłowe oraz przypisanie odpowiedniej wiadomości błędu gdy nie są. Paczka ta jest przejrzysta oraz posiada dobrą dokumentację i wiele przykładów implementacji.

## 2.6 Biblioteka do serializacji

Serializacja jest potrzebna gdy chcemy zapisać stan naszej aplikacji lub wysłać jakieś dane. Dla wygody użytkownika należy wprowadzić system zapisywania i wczytywaniu plików, żeby mógł wrócić do wcześniej stworzonych ciągów operacji oszczędzając jego czas.

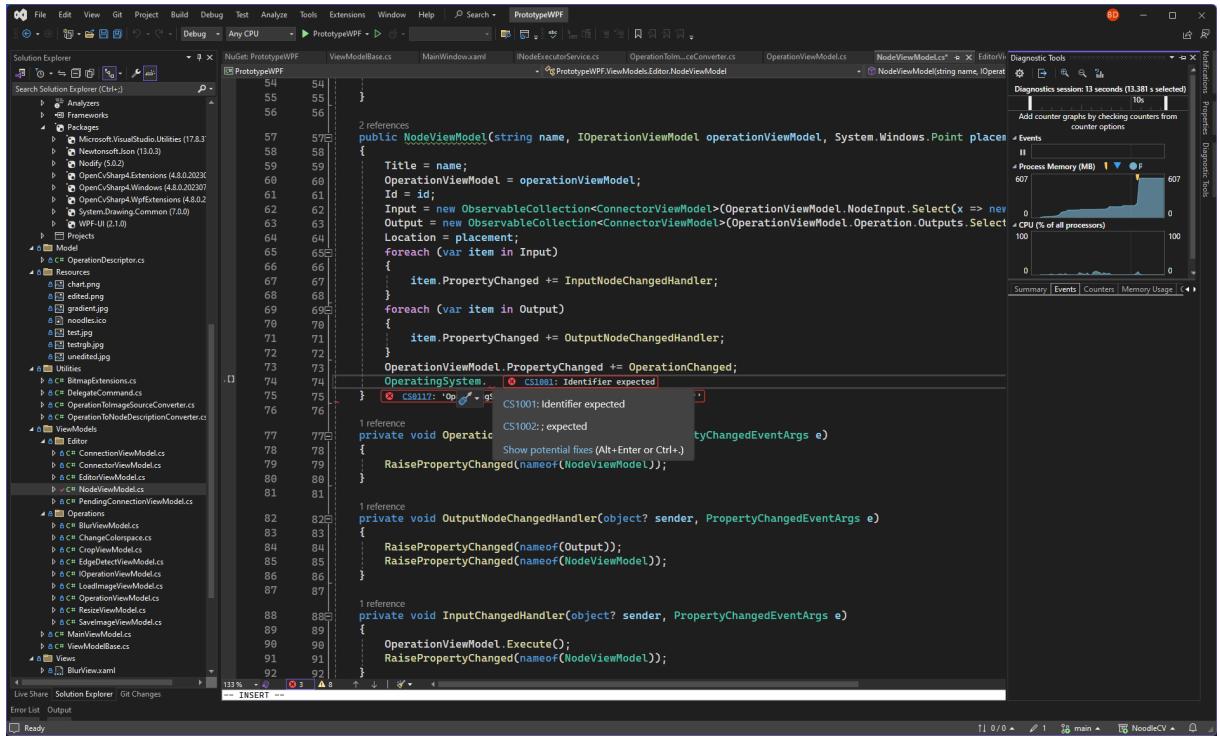
### 2.6.1 Json.NET

Newtonsoft.Json [40] to najpopularniejsza biblioteka do obsługi plików Json w galerii paczek NuGet [41]. Pozwala na serializowanie obiektów i deserializowanie Json'a do obiektów zachowując złożone struktury obiektów i kolekcje oraz posiada wiele parametrów za pomocą których można dopasować jego wyniki do większości zastosowań.

## 2.7 Środowisko deweloperskie

Odpowiednie środowisko może znacząco przyśpieszyć pracę nad projektem programistycznym. Nowoczesne IDE (ang. *Integrated Development Environment*) zamkają nawiasy, pomagają w poprawianiu błędów, refaktoryzacji i proponują automatyczne uzupełnienia zmiennych czy metod biorąc pod uwagę kontekst kodu w plikach. Ważne jest też zadbanie o kontrolę wersji w celu zapisywania stanu projektu oraz śledzeniu zmian zachodzących w nim.

## 2.7.1 Microsoft Visual Studio



Rys. 10: Interfejs programu Visual Studio. Opracowanie własne.

Microsoft Visual Studio [42] to zintegrowane środowisko programistyczne (IDE) opracowane i wydane przez firmę Microsoft. Jest przeznaczone do tworzenia aplikacji komputerowych, w tym aplikacji internetowych, aplikacji mobilnych, aplikacji desktopowych i aplikacji konsolowych. Posiada wiele gotowych zestawów narzędzi do pobrania dla specyficznych typów programów, a jego funkcjonalność dodatkowo można rozszerzać dodatkami pobieranymi z internetu.

## 2.7.2 Git

Git [43] to system kontroli wersji oparty na rozproszeniu danych (ang. *distributed version control system - DVCS*), opracowany przez Linusa Torvaldsa. Jest przeznaczony do śledzenia zmian w plikach i folderach, a także do współpracy nad projektami w czasie rzeczywistym.

## 3 Realizacja projektu

Przed rozpoczęciem pracy nad aplikacją stwierdzono możliwe zastosowania, które skorzystałyby z interfejsu graficznego do biblioteki przetwarzania obrazów. Na ich podstawie zostały zdefiniowane wymagania funkcjonalne - dotyczące możliwości programu oraz wymagania niefunkcjonalne - mówiące o tym jak użytkownik będzie korzystał z projektu.

### 3.1 Zastosowania

- **Tworzenie gotowych ciągów operacji:** Istnieje potrzeba przetwarzania małej liczby obrazów lub problem jest mało skomplikowany. Problematyczne jest wtedy szukanie programisty, który stworzy dla nas program, ale bez wiedzy o programowaniu zrobienie tego w kodzie samodzielnie może być zbyt ciężkie. Program z interfejsem graficznym pozwoli na stworzenie procesów nawet przez osoby mniej techniczne.
- **Uczenie się:** Osoby chcąc poznać techniki przetwarzania obrazów będą mogły w prosty sposób bez znajomości programowania zobaczyć na własne oczy działanie funkcji, ich interakcję ze sobą oraz łatwo dopasowywać ich parametry.
- **Komunikacja:** Dzięki temu oprogramowaniu programista może łatwiej i efektywniej porozumieć się z osobami mniej zaznajomionymi z programowaniem i dużo szybciej wprowadzać zmiany w porównaniu z pisaniem kodu i jego komplikacją przy każdej zmianie.

### 3.2 Analiza wymagań projektu

Oprogramowanie opisane w tej pracy zostało nazwane **NoodleCV**. Następne podrozdział przedstawią wymagania z jakimi trzeba było się zmierzyć w trakcie tworzenia aplikacji.

#### 3.2.1 Wymagania funkcjonalne

Stanowią onę listę funkcjonalności, które należy zaimplementować aby móc uznać aplikację za spełniającą swoje zadanie.

- **Przetwarzanie obrazów:** Pierwszym i głównym wymaganiem jest możliwość przetwarzania obrazów. Program musi być w stanie wczytać obraz, wykonać na nim operacje

i zapisać ich wynik.

- **Tworzenie ciągów operacji:** Wykonanie pojedynczej funkcji OpenCV nie jest skomplikowanym zadaniem i robią to już inne programy, a nawet aplikacje mobilne dostępne od razu w przeglądarce. Aby aplikacja spełniała swoje zadanie jako obsługę biblioteki programistycznej bez użycia kodu potrzebna jest funkcjonalność łączenia wielu operacji wykonywanych jedna po drugiej automatyczne.
- **Zapisywania pracy:** Użytkownik ma posiadać opcję zapisania do pliku ciągu operacji które stworzył. Zaletą aplikacji jest możliwość powrotu do stworzonego wcześniej systemu węzłów i zmiana ich parametrów w dowolnym miejscu. Bez zapisywania czy otwierania poprzedniego stanu programu praca włożona w efekt końcowy przepada wraz z zamknięciem aplikacji.

### 3.2.2 Wymagania niefunkcjonalne

Definiują one jakie doświadczenie powinien mieć użytkownik w trakcie korzystania z oprogramowania do przetwarzania obrazów.

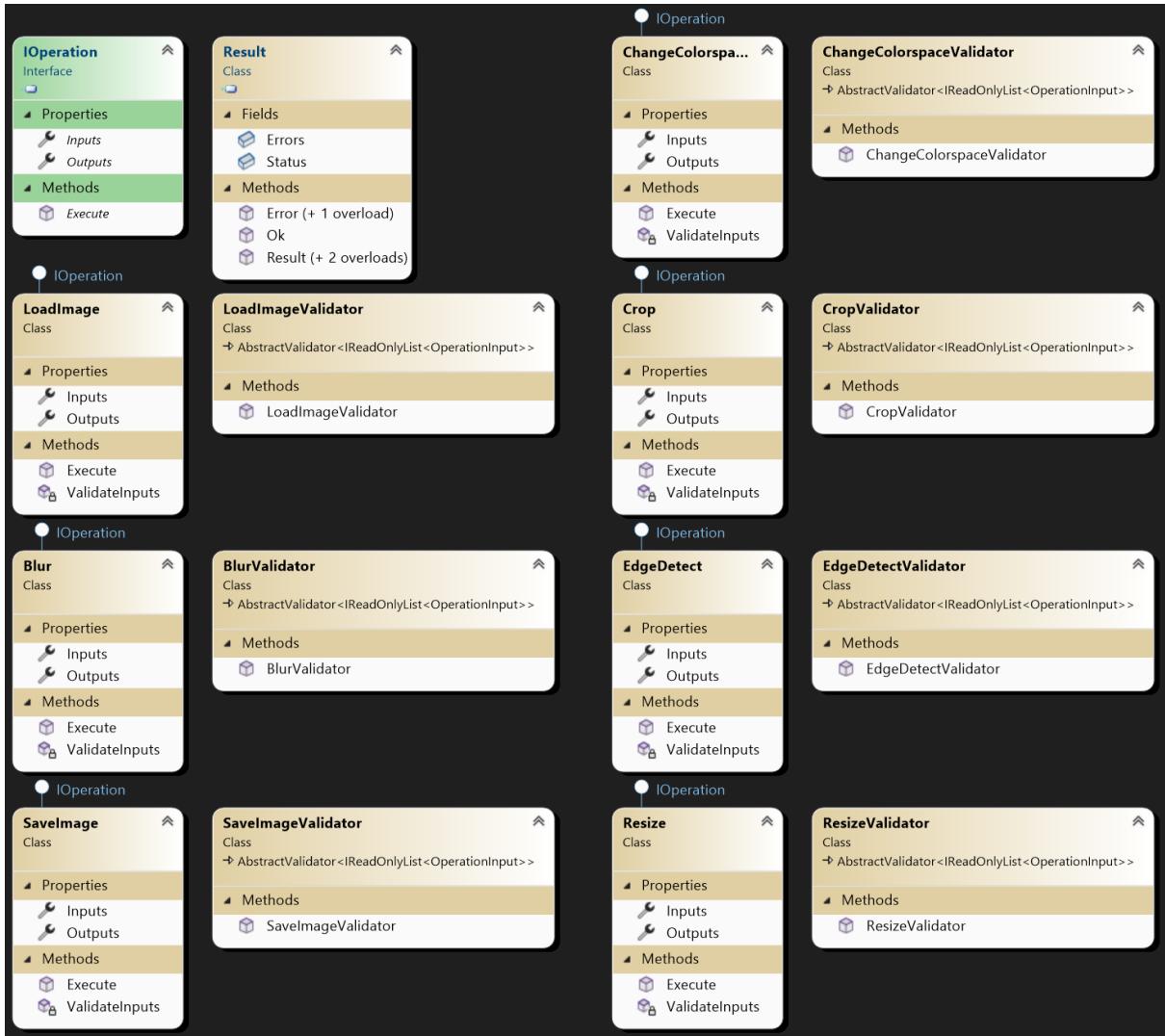
- **Proste w obsłudze:** Aplikacja powina być zbudowana w taki sposób, by osoba potrafiąca obsługiwać komputer mogła z minimalną ilością lub brakiem instrukcji zacząć korzystać z programu.
- **Natychmiastowe wyniki:** Przetwarzanie obrazów powinno odbywać się na tyle szybko by użytkownik widział zmiany które wprowadza na bieżąco. Otrzymanie rezultatu ma odbywać się od razu po wprowadzeniu nowych parametrów operacji.
- **Zastąpienie programowania do przetwarzania obrazów:** Program powinien być na tyle użyteczny żeby korzystać z niego zamiast obsługiwać OpenCV za pomocą kodu.

## 3.3 Architektura aplikacji

Projekt ten powstał w technologii Windows Presentation Foundation. Jedną z metod rozdzielenia widoku od logiki biznesowej w interfejsie tworzonym w WPF jest wzorzec MVVM (ang. *Model-View-ViewModel*). Jego cel to jasny podział aplikacji na *Model* - dane naszej aplikacji, ich interakcja ze sobą i implementacja mechanizmów działania oprogramowania. *View*

odpowiada tylko za wyświetlanie danych użytkownikowi oraz przyjmowanie jego interakcji jak kliknięcia czy wprowadzanie informacji. Za to klasy *ViewModel* zajmują się na połączeniu modelu i widoku konwertując obiekty z częścią biznesowej na dane które mają dotrzeć do użytkownika. Ich zadaniem jest też obsługa danych wprowadzanych przez interfejs aplikacji.

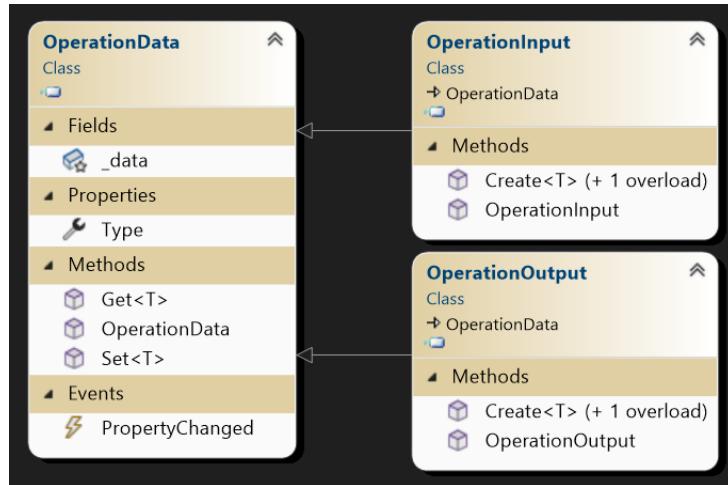
### 3.3.1 Model



Rys. 11: Diagram operacji. Opracowanie własne.

W NoodleCV warstwa *Model* jest minimalna Rys. 11. Posiada interfejs **IOperation** odpowiadający za formę wszystkich implementacji operacji. Posiadają generyczna kolekcje wejść *Inputs* oraz wyjść *Outputs*. Jedyna metoda zdefiniowana w nich to wykonanie operacji - zwraca

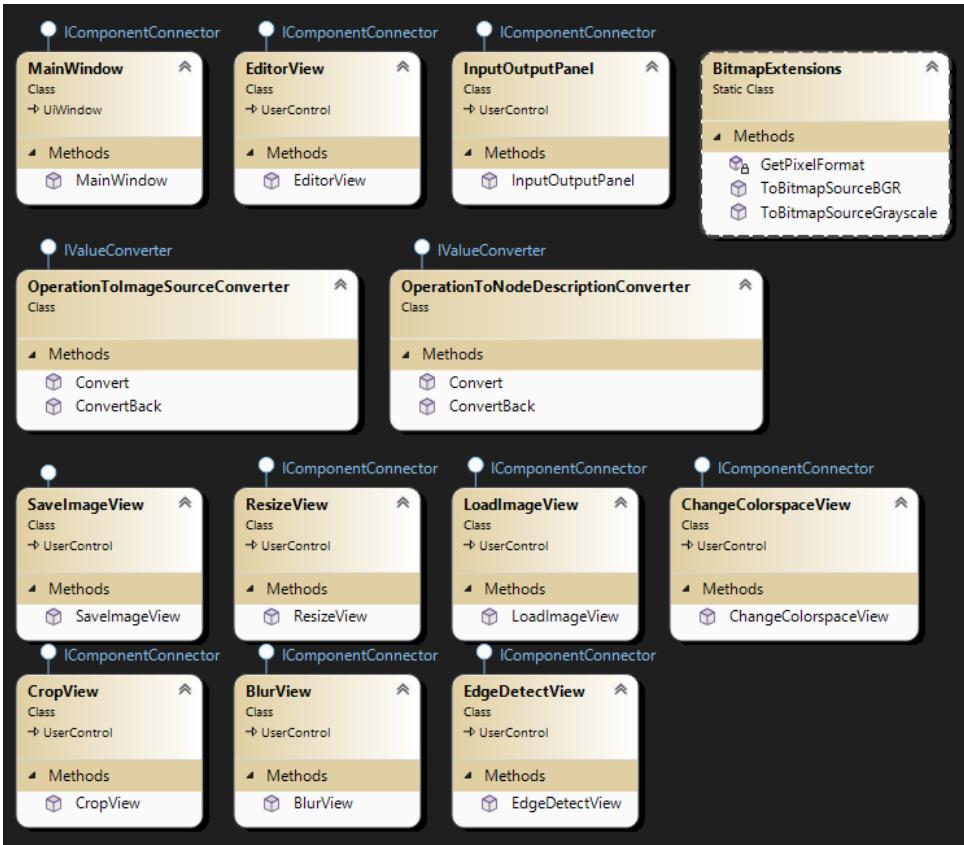
ona obiekt *Result*. Informuje ona następne warstwy czy operacja się powiodła, jeżeli nie to dla czego. Listę błędów otrzymujemy dzięki sprawdzeniu wejść za pomocą biblioteki Fluent Validation. Każda operacja wymaga swojego własnego validatora danych i zapewnia tym bezpieczne działanie aplikacji - użytkownik nie powinien być w stanie doprowadzić programu do błędu złym parametrem operacji.



Rys. 12: Diagram generycznych typów przechowujących dane operacji. Opracowanie własne.

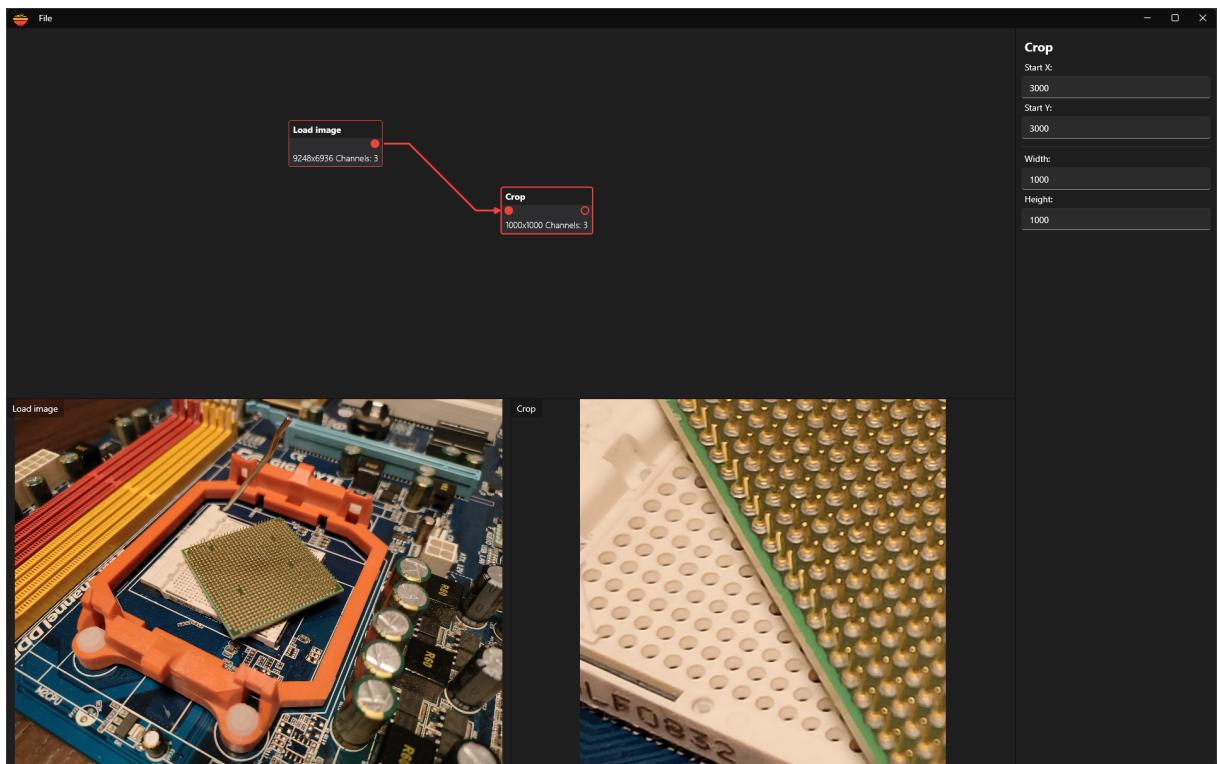
Wejścia i wyjścia parametrów operacji mogą być w wielu różnych typach więc kolekcje przyjmujące te dane musiały być generyczne. Daje to dużo większą elastyczność ale może utrudnić implementację. Tworząc operacje i ich późniejsze *ViewModel*'e trzeba zachować szczególną ostrożność. Przy tworzeniu nowych modeli zakładamy że obiekt w tym miejscu kolekcji będzie miał odpowiedni typ. Przy odczytywaniu danych z tej kolekcji nie ma informacji co to za klasa, należy poprawnie narzucić jej rodzaj inaczej napotkamy błąd i aplikacja wyłącza się. Pomimo tych problemów warto korzystać z tej struktury ponieważ niektóre operacje przyjmują tylko ścieżkę do pliku w formie tekstu, inne mogą nie mieć żadnego wyjścia ale wszystkie pasują do jednego wspólnego interfejsu.

### 3.3.2 View



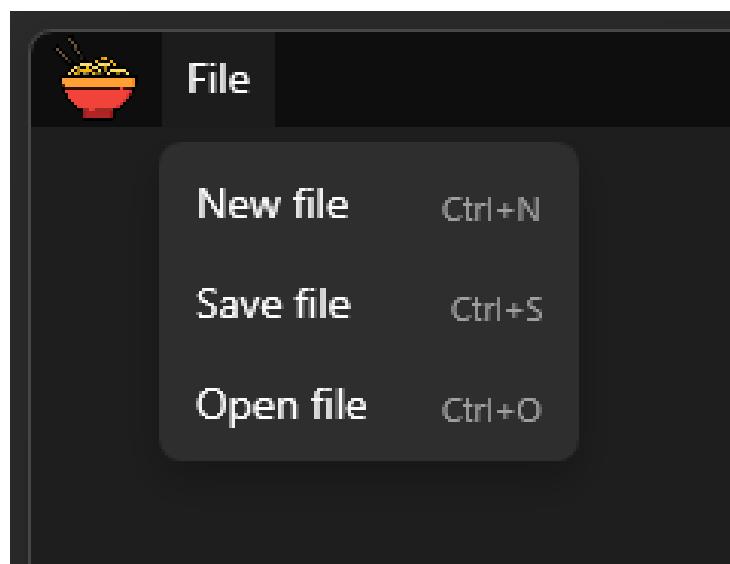
Rys. 13: Diagram klas używanych do tworzenia interfejsu. Opracowanie własne.

Warstwa widoku składa się z głównego okna, edytora i jego elementów. Dodatkowo użyto klasy statycznej posiadającej metody pozwalające na konwersję obrazów z formatu używanego przez operacje na format akceptowany przez wyświetlanie obrazu w WPF. Drugi konwerter zajmuje się pobieraniem danych z obrazu przetwarzanego przez operację i stworzenie opisu zawierający jego rozmiar oraz ilość kanałów koloru.



Rys. 14: Główne okno aplikacji z przykładowymi operacjami. Opracowanie własne.

Aplikacja posiada jedno okno Rys. 14. Użyta została klasa *UiWindow* z biblioteki WPF UI [35] pozwalająca łatwo zmodyfikować górny pasek by pasował do nowoczesnych systemów operacyjnych.



Rys. 15: Menu aplikacji. Opracowanie własne.

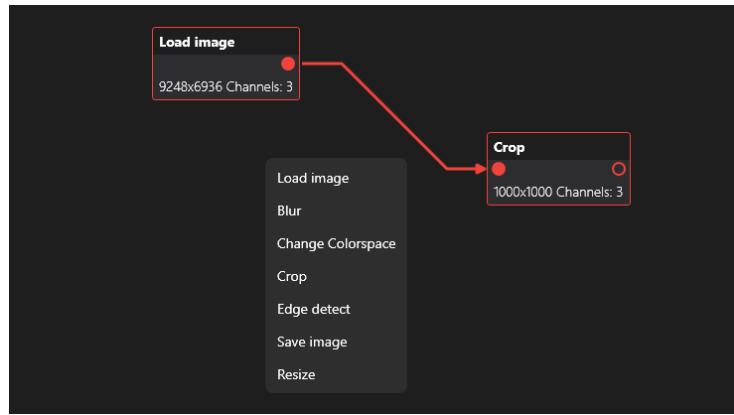
Menu pozwala na zapisywanie odczytywanie lub czyszczenie stanu aplikacji. Opcje te są też przypisane znany skrótom klawiszowym w celu ułatwienia ich używania.

Zawartość okna znajduje się w osobnym widoku edytora. Jest on podzielony na 3 główne elementy. Edytor (Rys. 16) gdzie dodaje się nowe operacje, podgląd (Rys. 18) gdzie pojawiają się wyniki operacji i panel od modyfikacji parametru (Rys. 19).



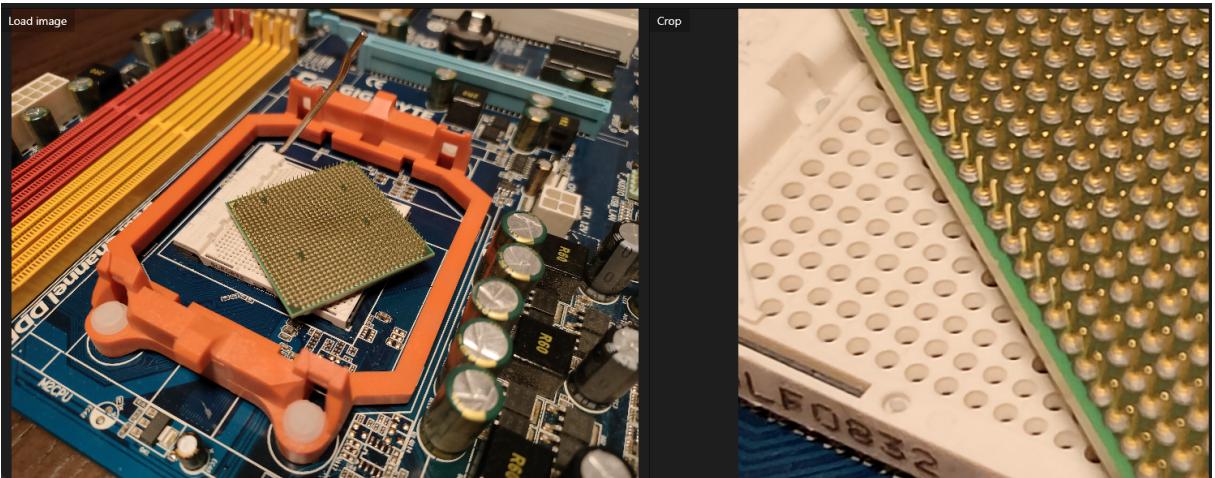
Rys. 16: Widok edytora. Opracowanie własne.

W tej części widoku użytkownik może dodawać nowe operacje, łączyć je, przeciągać oraz wybierać i zmieniać parametry wybranych bloczków. Operacje dodajemy za pomocą menu kontekstowego (Rys. 17) umieszczonego w edytorze. Otwiera się je za pomocą kliknięcia prawa przycisku myszy na jego obszarze. Węzły stworzone w ten sposób posiadają nagłówek odpowiadający nazwie operacji którą reprezentują oraz połączenia wejściowe i wyjściowe reprezentujące możliwe połączenia. Po uzyskaniu rezultatu pojawia się też informacja jakiego rozmiaru jest obraz znajdujący się na wyjściu i ile kanałów posiada - czy to obraz biało-czarny lub kolorowy. Są to elementy biblioteki Nodify [34] ze zmodyfikowanymi stylami w celu utrzymania jednolitej stylistyki projektu.



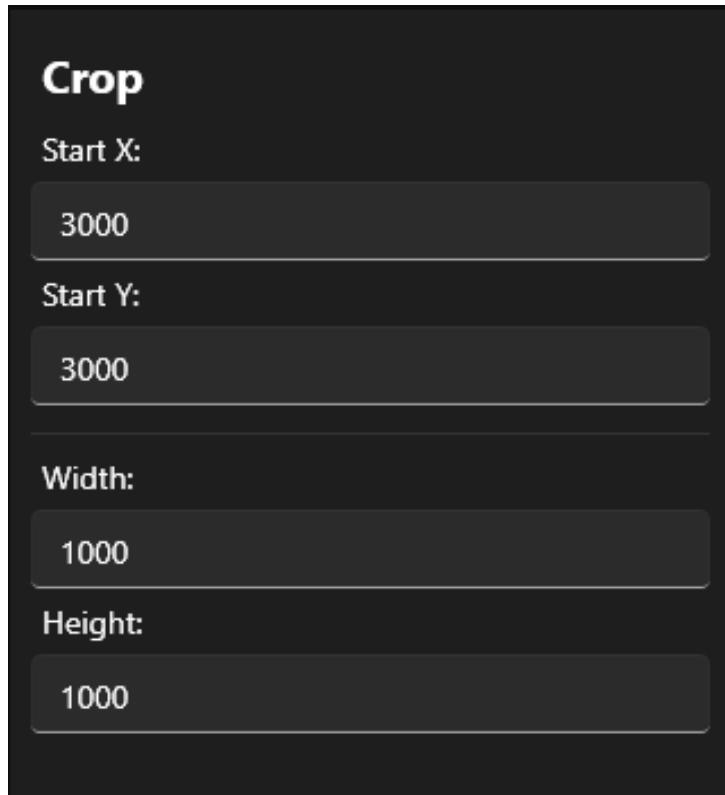
Rys. 17: Menu kontekstowe edytora. Opracowanie własne.

Po dodaniu operacji zostaje ona automatycznie przypisana do podglądu obrazu Rys. 18 umieszczonego pod edytorem. Użytkownik może później zmienić i wybrać które elementy są wyświetlane za pomocą klawiszy ‘1’ i ‘2’.



Rys. 18: Podgląd wyników operacji. Opracowanie własne.

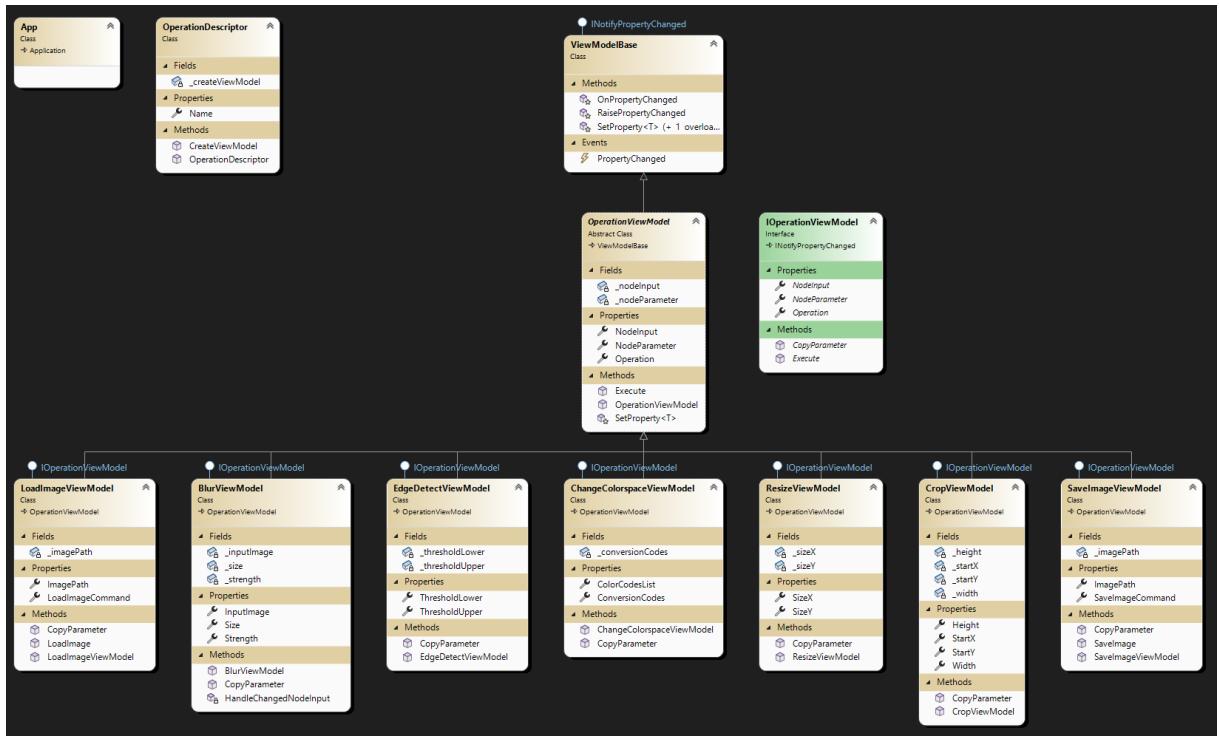
Ostatni element widoku aplikacji to panel gdzie można edytować parametry wybranej operacji. Każdy węzeł potrzebuje swojego osobnego widoku który poprawnie odwzorowuje parametry potrzebne do wykonania operacji. Są one przypisane do pól w odpowiednich View-Modeli i każda zmiana użytkownika jest przesyłana do niego w celu zaktualizowania wyników operacji.



Rys. 19: Edycja parametrów. Opracowanie własne.

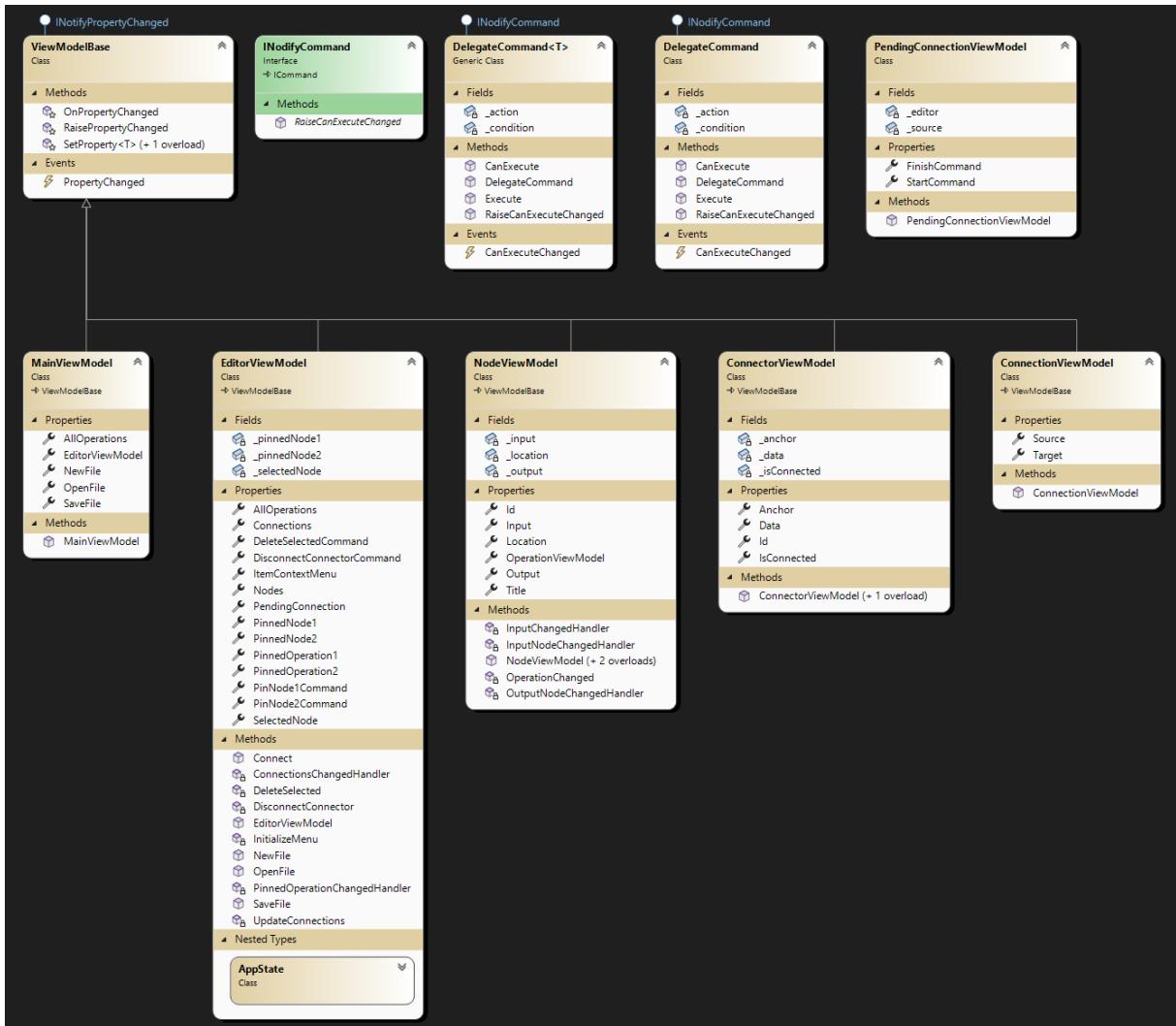
### 3.3.3 ViewModel

W przypadku tego projektu warstwa komunikacji między modelem i widokiem jest najbardziej obszerna. Większość z tych klas dziedziczy po ViewModelBase wziętym z *BindableBase* [44] z biblioteki *Prism* [45]. Znajdują się w nim metody pomagające z obsługą interfejsu *IPropertyChanged*. Jest on kluczowy w tworzeniu aplikacji reagującej natychmiastowo na dane wejściowe od użytkownika w technologii WPF. Dają one znać komponentom do których przypisane są wartości, że się zmieniły i trzeba je zaktualizować.



Rys. 20: Diagram ViewModeli operacji. Opracowanie własne.

Każda operacja potrzebuje swój ViewModel. Dbają one o poprawne wyświetlanie i przyznanie wartości z modelu. Zajmują się też uruchomieniem obliczeń i zwróceniem ich rezultatu. Każdy z nich posiada też metodę pozwalającą na skopiowanie parametrów z innej operacji. Jest to wykorzystane przy odczytywaniu danych z zapisanego stanu.



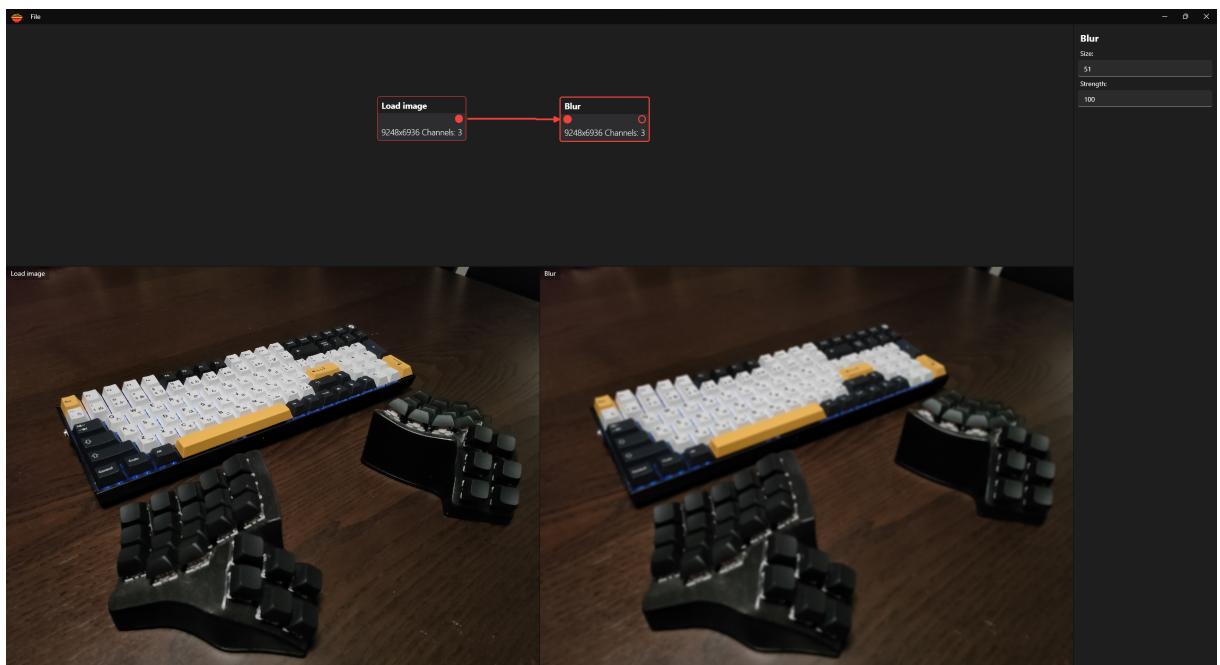
Rys. 21: Diagram ViewModelów aplikacji. Opracowanie własne.

Pozostałe ViewModele dalej korzystają z tej samej klasy bazowej. MainViewModel tworzy jedynie komendy do których odnosi się menu główne, większość logiki programu jest umieszczona w EditorViewModelu. Jest tam umieszczona kolekcja przechowująca stworzone węzły i ich stany oraz połączenia między nimi. Klasa NodeViewModel odpowiada bloczkom w edytorze. Przechowując one ViewModel odpowiedniej operacji, ale też pozycje na ekranie wraz z wejściami i wyjściami. Są to obiekty typu ConnectorViewModel między którymi tworzą się połączenia przesyłające dane między węzłami.

## 3.4 Zaimplementowane operacje

Dla każdej operacji zostanie wczytany ten sam obraz i zamieszczone poniżej zrzuty ekranu będą prezentowały podgląd zdjęcia przed i po, a także panel z parametrami. Zaimplementowano najpopularniejsze [46] metody, które pasowały do obecnych założeń projektu - operacja powinna przetwarzać obraz i zwracać obraz jako wynik.

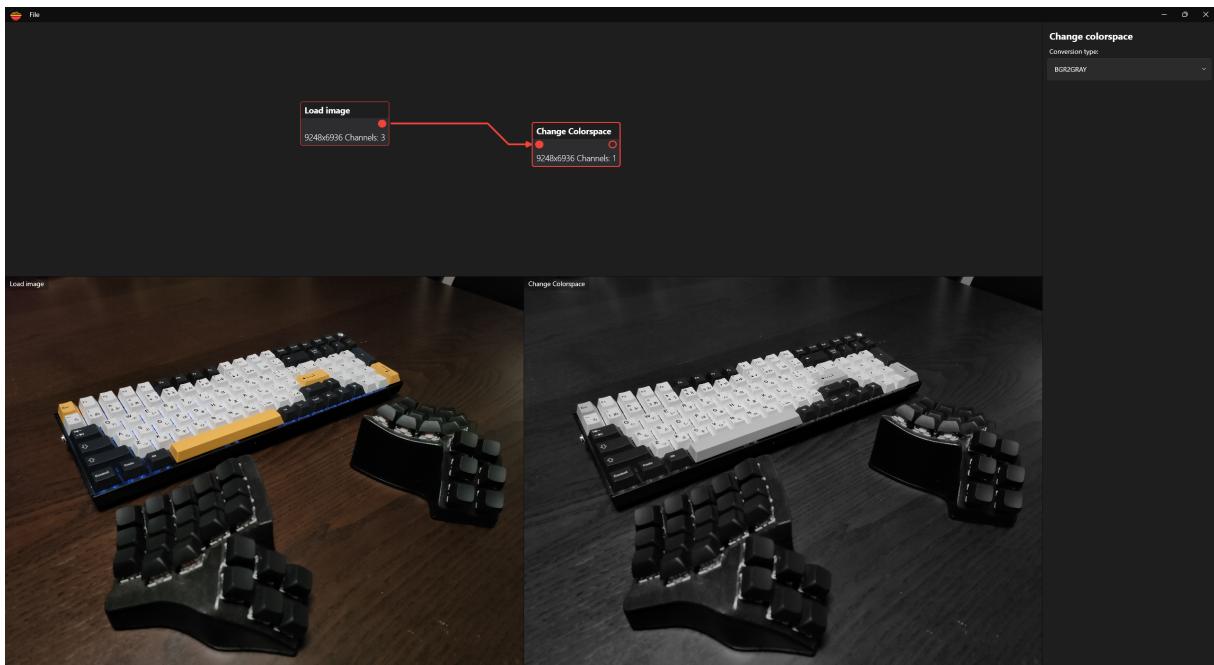
### 3.4.1 Blur



Rys. 22: Operacja *Blur*. Opracowanie własne.

Zaimplementowana w oparciu o *Gaussian Blur* [47]. Parametr *Size* musi być nieparzysty ponieważ opisuje on rozmiar maski na podstawie której wartość piksela jest uśredniana - potrzeby jest środek. *Strength* opisuje natomiast odchylenie standardowe jakie będzie zastosowane przy przeprowadzaniu operacji.

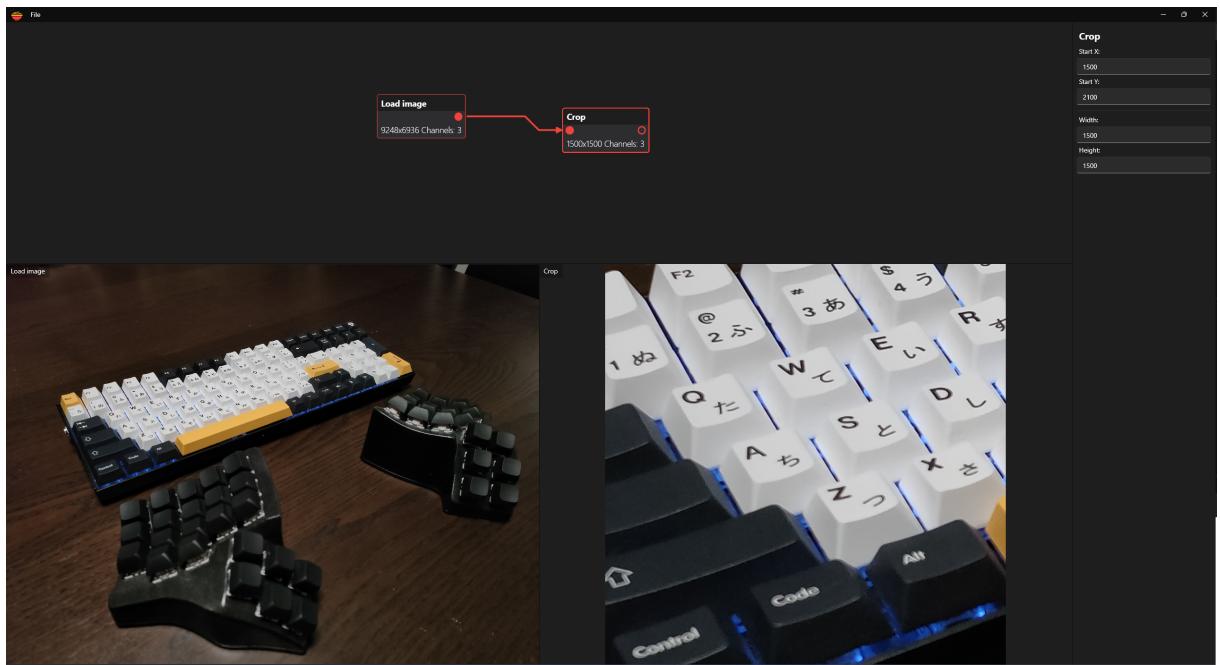
### 3.4.2 ChangeColorspace



Rys. 23: Operacja *ChangeColorspace*. Opracowanie własne.

Jest to funkcja *CvtColor* [48]. Konwertuje ona przestrzeń kolorów obrazu i dopasowuje ilość kanałów. Jej jedynym parametrem jest kod z rodzajem konwersji np. *BGR2GRAY* oznaczający przejście ze zwykłego kolorowego zdjęcia na biało-czarne.

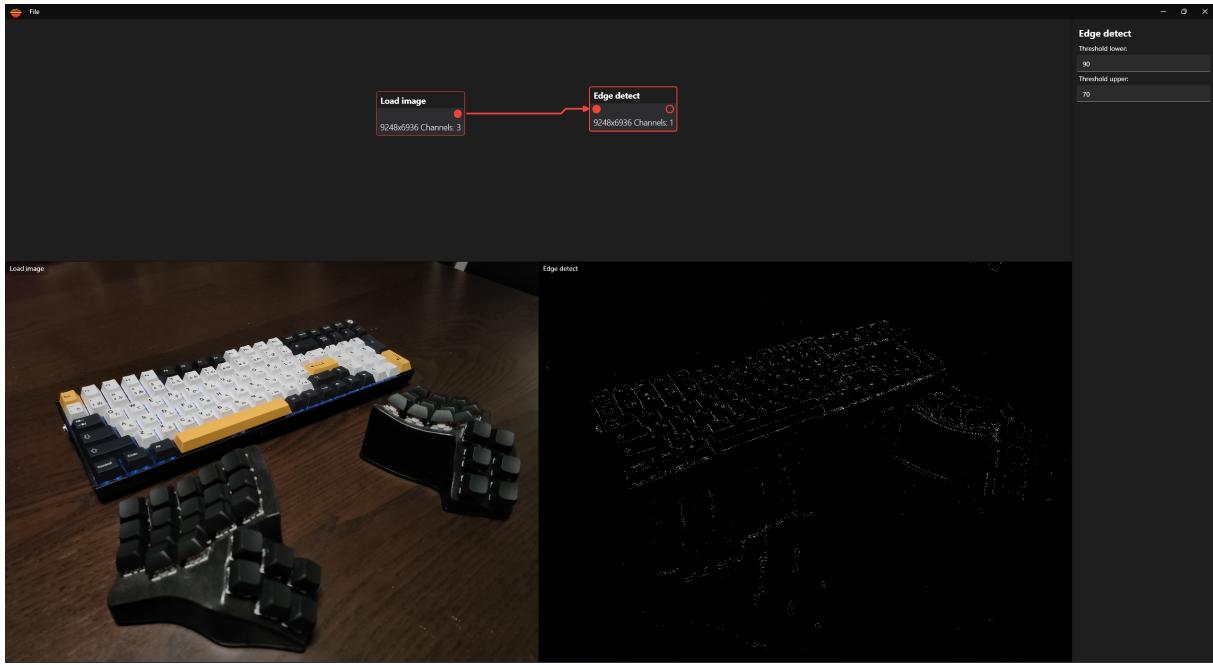
### 3.4.3 Crop



Rys. 24: Operacja *Crop*. Opracowanie własne.

Tutaj implementacja to nie metoda ale stworzenie nowego *Mat'a* [49]. Do jego konstruktora zostają dodane koordynaty startowe oraz rozmiar nowego prostokąta który wyznacza co będzie znajdowało się w nowym obrazie.

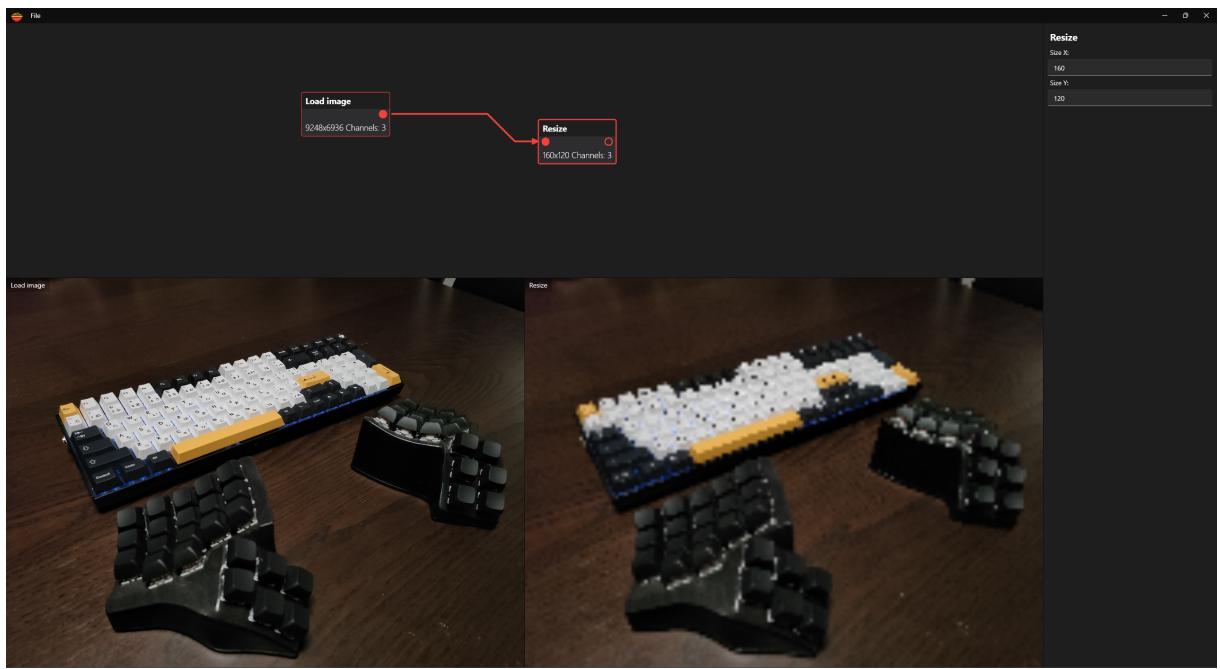
### 3.4.4 EdgeDetect



Rys. 25: Operacja *EdgeDetect*. Opracowanie własne.

Metoda *Canny* [50] to jeden z kilku dostępnych algorytmów wykrywania krawędzi. Jako parametry wejściowe należy podać niższy i wyższy próg akceptacji. Jeżeli krawędź jest mało wyraźna zostaje odrzucona przez algorytm. Wartości powyżej drugiego parametru są uznawane za krawędź, a te które są pomiędzy zostają włączone do wyniku jako element łączący krawędzie jeżeli taki jest.

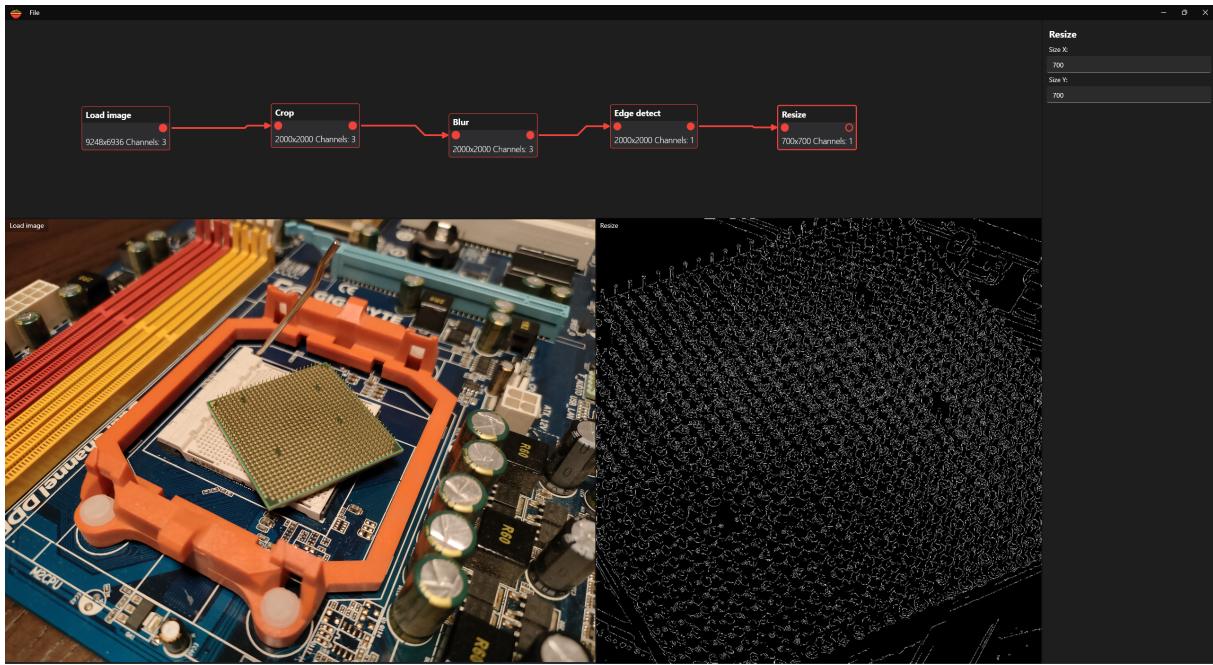
### 3.4.5 Resize



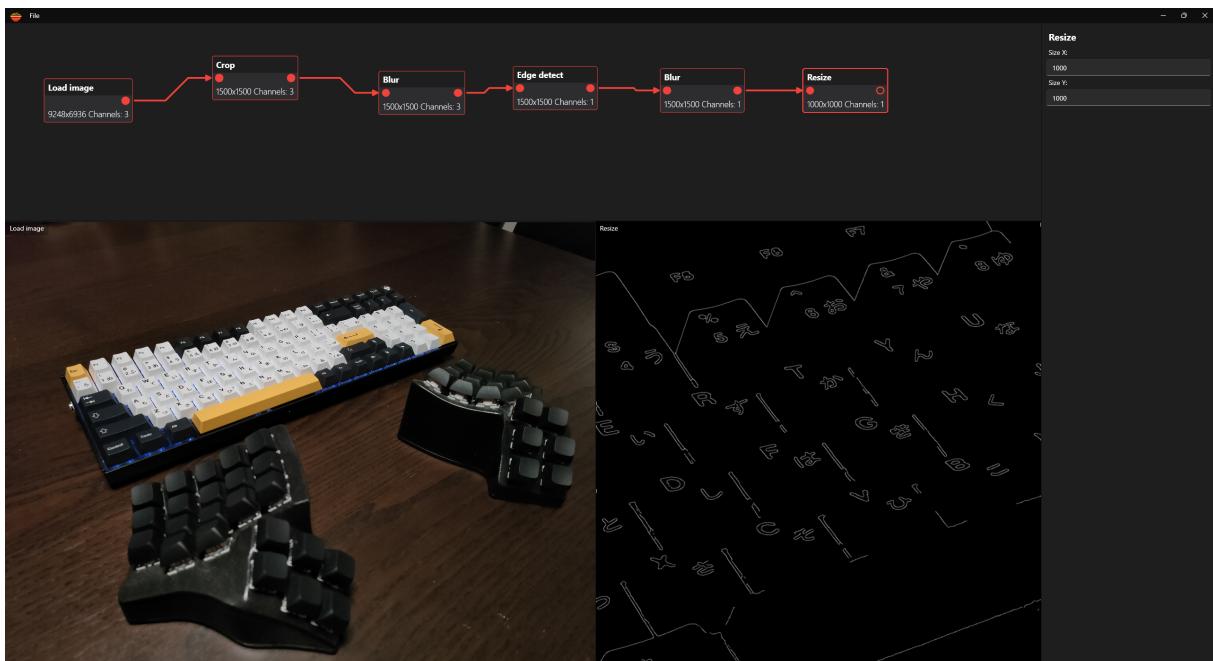
Rys. 26: Operacja *Resize*. Opracowanie własne.

*Resize* jest wykonywane przez metodę o tej samej nazwie z biblioteki OpenCV [51]. Po podaniu nowych wymiarów otrzymujemy obraz stworzony na podstawie oryginalnego ze zmienioną rozdzielczością.

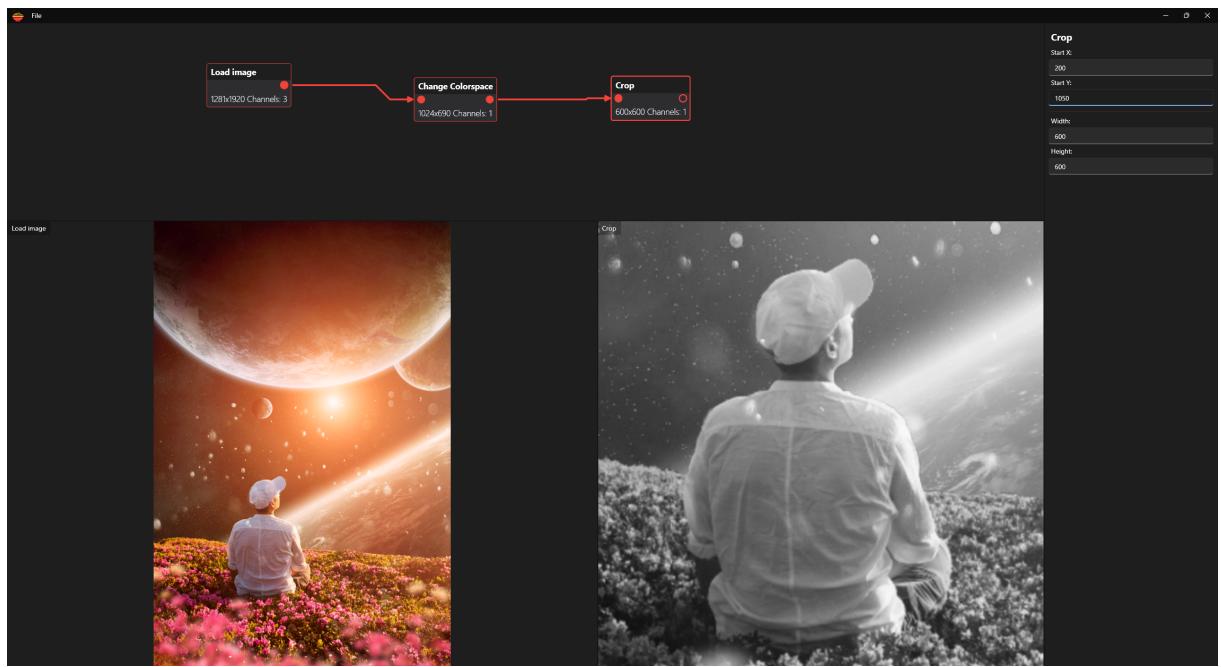
### 3.5 Przykładowe ciągi operacji



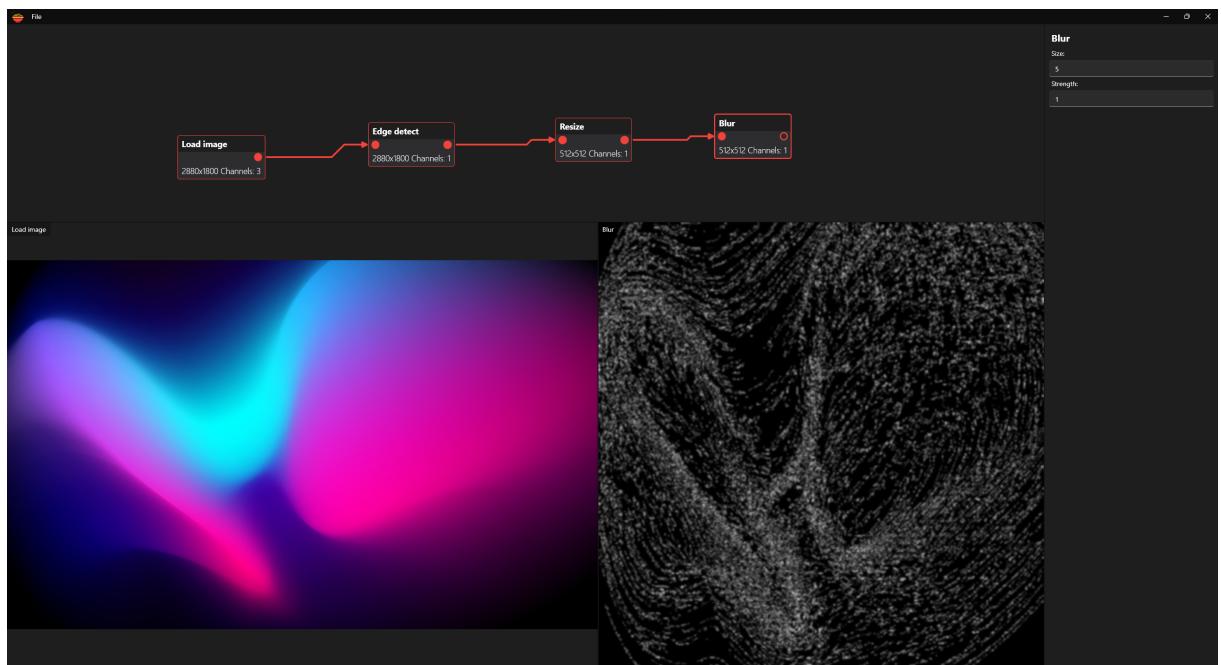
Rys. 27: Wykrywanie krawędzi na wybranym fragmencie obrazu. Opracowanie własne.



Rys. 28: Wykrywanie krawędzi dla uwydatnienia liter. Opracowanie własne.



Rys. 29: Obcięcie zdjęcia i zmiana na odcień szarości. Opracowanie własne.



Rys. 30: Abstrakcyjne edytowanie. Opracowanie własne.

## **4 Podsumowanie**

*Na koniec należy podsumować, co było celem pracy. Czy cel ten został osiągnięty. Jakie są możliwe wnioski. Czy są dalsze możliwości rozwoju.*

## Bibliografia

1. Chakravorty, P. What Is a Signal? [Lecture Notes]. *IEEE Signal Processing Magazine* **35**, 175–177. <https://api.semanticscholar.org/CorpusID:52164353> (2018).
2. The Televisor: Successful Test of New Apparatus. *The Times (London)*, 9 column C. <https://www.bairdtelevision.com/the-televisor-successful-test-of-new-apparatus-1926.html> (1926).
3. Gonzalez, R. C. & Woods, R. E. *Digital image processing* 3rd, 23–28. ISBN: 978-0-13-168728-8 (Prentice Hall, Upper Saddle River, N.J., 2008).
4. *OpenCV: cv::Mat Class Reference* OpenCV. [https://docs.opencv.org/3.4/d3/d63/classcv\\_1\\_1Mat.html#details](https://docs.opencv.org/3.4/d3/d63/classcv_1_1Mat.html#details).
5. Szymon, K. *Przetwarzanie obrazów* <https://skszymon.eu/blog/2023/cv-przetw-obrazow/>.
6. *DICOM Part 1: Introduction and Overview* National Electrical Manufacturers Association. [https://dicom.nema.org/medical/dicom/current/output/chtml/part01/chapter\\_1.html#sect\\_1.1](https://dicom.nema.org/medical/dicom/current/output/chtml/part01/chapter_1.html#sect_1.1).
7. Bokhan, K. *Computer Vision in Manufacturing: A Guide to Integration* N-iX. <https://www.n-ix.com/computer-vision-manufacturing/>.
8. Enterprise, D. *How Can Police, Firefighters and Search and Rescue Professionals Use Drones to Keep the Public Safe?* <https://enterprise-insights.dji.com/blog/droneshelp-how-can-police-firefighters-and-search-and-rescue-professionals-use-drones-to-keep-the-public-safe>.
9. Bertalmio, M. *Image Processing for Cinema* ISBN: 9780429067501 (lut. 2014).
10. Rosenfeld, A. Picture Processing by Computer. *ACM Comput. Surv.* **1**, 147–176. ISSN: 0360-0300. <https://doi.org/10.1145/356551.356554> (1969).
11. OpenCV. *OpenCV CPU optimizations* <https://github.com/opencv/opencv/wiki/CPU-optimizations-build-options>. (dostęp: 07.01.2024).
12. Keller, K. *i in. w Ullmann's Encyclopedia of Industrial Chemistry* (John Wiley & Sons, Ltd, 2000). ISBN: 9783527306732.

13. Dungarwal, S. *Understanding black & white film photography* <https://www.shreyansdungarwal.com/blog/2019/4/15/understanding-black-amp-white-film-photography>.
14. Dalladay, A. J. *The British Journal Photographic Almanac* 149–155 (Henry Greenwood & Co Ltd, London, 1956).
15. Blaker, A. *Photography: Art and Technique* ISBN: 9780716711162. <https://books.google.pl/books?id=bWdgQgAACAAJ> (W. H. Freeman, 1980).
16. BBC. *John Logie Baird - Historic figure* [https://www.bbc.co.uk/history/historic\\_figures/baird\\_logie.shtml](https://www.bbc.co.uk/history/historic_figures/baird_logie.shtml).
17. Analog. *Video Basics* <https://www.analog.com/en/technical-articles/basics-of-analog-video.html>.
18. Wigginton, C., Curran, M. & Brodeur, C. *Global mobile consumer trends* Deloitte. <https://www2.deloitte.com/content/dam/Deloitte/us/Documents/technology-media-telecommunications/us-global-mobile-consumer-survey-second-edition.pdf>.
19. McHugh-Johnson, M. *How Pixel's Super Res Zoom works* Google. <https://blog.google/products/pixel/super-res-zoom-google-pixel/>.
20. Cervantes, E. *What is Night Mode and how does it work* Android Authority. <https://www.androidauthority.com/what-is-night-mode-and-how-does-it-work-979590/>.
21. *Zrzut ekranu Adobe Photoshop* VisArts Center. <https://www.visartscenter.org/event/introduction-to-adobe-photoshop-2/>.
22. *Zrzut ekranu ImageJ* NC State University Libraries. <https://www.lib.ncsu.edu/workshops/introduction-to-imagej-for-scientific-research>.
23. *Batch Processing ImageJ*. <https://imagej.net/scripting/batch>.
24. Commons, W. *File:20221126\_17\_58\_02-Fiji (software) - Wikipedia.png* — Wikimedia Commons, the free media repository 2023. [https://commons.wikimedia.org/w/index.php?title=File:20221126\\_17\\_58\\_02-Fiji\\_\(software\)\\_-\\_Wikipedia.png&oldid=763353557](https://commons.wikimedia.org/w/index.php?title=File:20221126_17_58_02-Fiji_(software)_-_Wikipedia.png&oldid=763353557).
25. *Zrzut ekranu GIMP* GIMP. <https://www.gimp.org/release-notes/gimp-2.10.html>.

26. *Substance 3D Designer* Adobe. <https://helpx.adobe.com/substance-3d-designer.html>.
27. Dietrich, E. & Smacchia, P. *C# Version History: Examining the Language Past and Present* <https://blog.ndepend.com/c-versions-look-language-history/>.
28. Microsoft. *What is NuGet?* <https://learn.microsoft.com/en-us/nuget/what-is-nuget>.
29. Microsoft. *Introduction to .NET Core* <https://learn.microsoft.com/en-us/dotnet/core/introduction>.
30. *Avalonia UI* <https://avaloniaui.net/>.
31. *What is Windows Presentation Foundation (WPF)?* <https://docs.microsoft.com/en-us/dotnet/desktop/wpf/overview/?view=netdesktop-8.0>.
32. Gaikwad, Y. *Windows Presentation Foundation History* <https://www.linkedin.com/pulse/windows-presentation-foundation-history-yash-gaikwad-1yfbf/>.
33. Miroiu. *Image of example node network* <https://github.com/miroiu/nodify/wiki/Getting-Started>.
34. Miroiu. *nodify* <https://github.com/miroiu/nodify>.
35. Lepoco. *WPF UI* lepo.co. <https://github.com/lepoloco/wpfui>.
36. *OpenCV History and Background* Computer Vision Project. <https://computervisionproject.com/opencv/intro/opencvbackground.html>.
37. Corporation, E. *EmguCV* <https://github.com/emgucv/emgucv>.
38. shimat. *opencvsharp* <https://github.com/shimat/opencvsharp>.
39. FluentValidation. *FluentValidation* <https://github.com/FluentValidation/FluentValidation>.
40. JamesNK. *Newtonsoft.Json* <https://github.com/JamesNK/Newtonsoft.Json>.
41. *Most downloaded NuGet package for Json* NuGet. <https://www.nuget.org/packages?q=json>.
42. Microsoft. *Visual Studio* <https://visualstudio.microsoft.com/pl/>.
43. Torvalds, L. *Git* <https://git-scm.com/>.
44. Library, P. *Prism BindableBase* <https://github.com/PrismLibrary/Prism/blob/master/src/Prism.Core/Mvvm/BindableBase.cs>.

45. Library, P. *Prism* <https://github.com/PrismLibrary/Prism>.
46. Deepak\_Raj. *9 Most important inbuilt functions in OpenCV for Computer Vision* <https://medium.com/analytics-vidhya/9-most-important-inbuilt-functions-in-opencv-for-computer-vision-f930cba28b14>.
47. *Gaussian Blur documentation* OpenCV. <http://bit.ly/48VaSG9>.
48. *CvtColor documentation* OpenCV. <https://bit.ly/306a1uq>.
49. *Mat documentation* OpenCV. <https://bit.ly/3tYVkT3>.
50. *Canny documentation* OpenCV. <https://bit.ly/3tWcPU2>.
51. *Resize documentation* OpenCV. <https://bit.ly/3HnUMcx>.

# Spis rysunków

1	Wywoływanie biało czarnej kliszy [13]. . . . .	8
2	Adobe Photoshop [21]. . . . .	10
3	ImageJ [22]. . . . .	11
4	Fiji - sukcesor ImageJ [24]. . . . .	12
5	GIMP wersja 2.10 [25]. . . . .	13
6	Adobe Substance Designer [26]. . . . .	14
7	Opis elementów edytora węzłów [33]. . . . .	17
8	Galeria elementów WPF UI. Opracowanie własne. . . . .	18
9	Galeria elementów WinUI 3. Opracowanie własne. . . . .	18
10	Interfejs programu Visual Studio. Opracowanie własne. . . . .	21
11	Diagram operacji. Opracowanie własne. . . . .	24
12	Diagram generycznych typów przechowujących dane operacji. Opracowanie własne. . . . .	25
13	Diagram klas używanych do tworzenia interfejsu. Opracowanie własne. . . . .	26
14	Główne okno aplikacji z przykładowymi operacjami. Opracowanie własne. . . . .	27
15	Menu aplikacji. Opracowanie własne. . . . .	27
16	Widok edytora. Opracowanie własne. . . . .	28
17	Menu kontekstowe edytora. Opracowanie własne. . . . .	29
18	Podgląd wyników operacji. Opracowanie własne. . . . .	29
19	Edycja parametrów. Opracowanie własne. . . . .	30
20	Diagram ViewModeli operacji. Opracowanie własne. . . . .	31
21	Diagram ViewModeli aplikacji. Opracowanie własne. . . . .	32
22	Operacja <i>Blur</i> . Opracowanie własne. . . . .	33
23	Operacja <i>ChangeColorspace</i> . Opracowanie własne. . . . .	34

24	Operacja <i>Crop</i> . Opracowanie własne. . . . .	35
25	Operacja <i>EdgeDetect</i> . Opracowanie własne. . . . .	36
26	Operacja <i>Resize</i> . Opracowanie własne. . . . .	37
27	Wykrywanie krawędzi na wybranym fragmencie obrazu. Opracowanie własne. . . . .	38
28	Wykrywanie krawędzi dla uwydatnienia liter. Opracowanie własne. . . . .	38
29	Obcięcie zdjęcia i zmiana na odcień szarości. Opracowanie własne. . . . .	39
30	Abstrakcyjne edytowanie. Opracowanie własne. . . . .	39

## **Spis tabel**