

Akademia Górniczo Hutnicza
im. Stanisława Staszica w Krakowie



Wydział Inżynierii Metali i Informatyki Przemysłowej
Katedra Informatyki Stosowanej i Modelowania

Praca Dyplomowa Inżynierska

**„Interaktywne przetwarzanie obrazów mikroskopowych
związanych z oceną adhezji składników komórkowych”**

**„Interactive processing of microscopic images
related to evaluation adhesion of cellular components”**

autor: Konrad Lenart
kierunek: Inżynieria Obliczeniowa
opiekun pracy: dr hab. Magdalena Kopernik, prof. AGH

Kraków, 2021

Upředzony o odpowiedzialności karnej na podstawie art. 115 ust. 1 i 2 ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (tj. Dz.U. z 2006 r. Nr 90, poz. 631 z późn. zm.): „Kto przywłaszcza sobie autorstwo albo wprowadza w błąd co do autorstwa całości lub części cudzego utworu albo artystycznego wykonania, podlega grzywnie, karze ograniczenia wolności albo pozbawienia wolności do lat 3. Tej samej karze podlega, kto rozpowszechnia bez podania nazwiska lub pseudonimu twórcy cudzy utwór w wersji oryginalnej albo w postaci opracowania, artystyczne wykonanie albo publicznie zniekształca taki utwór, artystyczne wykonanie, fonogram, wideogram lub nadanie.”, a także upředzony o odpowiedzialności dyscyplinarnej na podstawie art. 211 ust. 1 ustawy z dnia 27 lipca 2005 r. Prawo o szkolnictwie wyższym (tj. Dz. U. z 2012 r. poz. 572, z późn. zm.) „Za naruszenie przepisów obowiązujących w uczelni oraz za czyny uchybiające godności studenta student ponosi odpowiedzialność dyscyplinarną przed komisją dyscyplinarną albo przed sądem koleżeńskim samorządu studenckiego, zwanym dalej „sądem koleżeńskim”, oświadczam, że niniejszą pracę dyplomową wykonałem osobiście, samodzielnie, i że nie korzystałem ze źródeł innych niż wymienione w pracy.

Kraków, dnia.....
(miejscowość)

.....
(podpis)

Spis treści

1	Wstęp	4
2	Cele i założenia	5
2.1	Cel główny	5
2.2	Cel poboczny	5
2.3	Założenia	5
2.4	Podstawowe funkcje biblioteki	6
2.5	Podstawowe funkcje aplikacji okienkowej	6
2.6	Rekomendacje	7
3	Część teoretyczna	7
3.1	Adhezja komórek	7
3.2	Rozszerzenie TIFF	7
3.3	Binaryzacja	7
3.4	Element Strukturalny	8
3.5	Sąsiedztwo	9
3.6	Erozja	10
3.7	Dylatacja	10
3.8	Otwarcie morfologiczne	11
3.9	Zamknięcie morfologiczne	12
3.10	Korekcja Gamma	13
4	Część Praktyczna	14
4.1	Wykorzystane oprogramowanie	14
4.1.1	Gimp 2.10.22	14
4.1.2	CMake 3.19.0	14
4.1.3	LibTIFF 4.2.0	15
4.1.4	Visual Studio 2019	15
4.1.5	L ^A T _E X	15
4.1.6	Visual Studio Code	15
4.1.7	.NET Framework 4.7.2	16
4.1.8	nlohmann.json 3.9.1	16
4.1.9	OpenMP 5.1	16
4.2	Budowa oprogramowania	16
4.3	Diagramy klas	17
4.4	Omówienie implementacji	18
4.5	Omówienie interfejsu aplikacji okienkowej	22

5	Wyniki i dyskusja	25
5.1	Przygotowanie katalogu głównego	25
5.2	Przykład Binaryzacji	26
5.3	Przykład Otwarcia	26
5.4	Przykład Zamknięcia	27
5.5	Przykład Dylatacji	27
5.6	Przykład Erozji	28
5.7	Przykład Etykietowania	28
6	Wnioski	30
7	Podsumowanie	31
8	Bibliografia	32

1 Wstęp

Każdy dzień to nowy rekord postępu dla całej cywilizacji technicznej. Szybkość z jaką pojawiają się nowe technologie stale rośnie. Nikogo nie dziwi już ile ludzkość potrafi dokonać w ciągu dwudziestu czterech godzin. Obecny trend rozwoju zawdzięczamy pojawieniu się komputerów. Komputery zastąpiły ludzi w wielu czynnościach, stworzyły także nowe miejsca pracy. Wyreczają nas w skomplikowanych obliczeniach, pozwalają testować systemy bezpieczeństwa w samochodach bez konieczności ich niszczenia, nauczyliśmy je nawet pilotować olbrzymie samoloty. Każdy z nas nosi w kieszeni większą moc obliczeniową niż komputery obecne na statkach kosmicznych, które lądowały na księżycu w ramach programu Apollo. Zaprzęgnięcie komputerów do automatyzacji pracy, którą ludzie do tej pory wykonywali ręcznie zaoszczędza nam sporo czasu. Charakter tej pracy jest analogiczny - zastąpienie ludzi w mozolej, ręcznej ocenie adhezji składników komórkowych - automatycznym procesem, który przez człowieka będzie tylko nadzorowany.

Jedną z najważniejszych dziedzin, w której komputery znalazły zastosowanie jest medycyna. Postęp w tej dziedzinie ratuje każdego dnia miliony istnień na całym świecie. W niniejszej pracy podjęto próbę oceny adhezji składników komórkowych wykorzystując do tego analizę i przetwarzanie obrazów cyfrowych pochodzących z mikroskopu. Z pomocą odpowiednich algorytmów zliczone zostają obiekty widoczne na zdjęciu. Następnie bazując na faktach o składnikach komórkowych krwi obiekty zostają zakwalifikowane jako komórka krwi lub odrzucone.

Założeniem pracy było stworzenie łatwej w obsłudze biblioteki napisanej w języku C++. Biblioteka napisana została w taki sposób, aby z łatwością można było dołączyć ją do istniejących projektów w środowisku Visual Studio. Cele poboczne pracy to stworzenie interfejsu użytkownika w celu zaprezentowania funkcjonalności biblioteki. Do tego celu wykorzystana została aplikacja na komputery osobiste z wykorzystaniem .NET Framework 4.7.2. Dokładny opis biblioteki jak i interfejsu użytkownika ma miejsce w rozdziale drugim po części teoretycznej, traktującej o wszystkich wykorzystanych w programie zagadnieniach związanych z analizą i przetwarzaniem obrazów cyfrowych.

2 Cele i założenia

2.1 Cel główny

Celem głównym było stworzenie biblioteki zajmującej się analizą i przetwarzaniem obrazów cyfrowych. Biblioteka ma za zadanie ocenę adhezji komórek krwi widocznych na obrazach pochodzących z mikroskopu, ma również ułatwić oraz usprawnić pracę ludzi. Podczas pracy nad biblioteką utrzymywany był kontakt z osobami, które będą z niej korzystać. Biblioteka napisana została w taki sposób, aby dostosować jej funkcjonalności do istniejącej już metodologii. Dzięki temu analiza obrazów niosących informacje na temat krwi pozwoli zaoszczędzić znaczącą ilość czasu.

2.2 Cel poboczny

Celem pobocznym było stworzenie interaktywnej aplikacji okienkowej, w której zaprezentowane zostało działanie biblioteki. Aplikacja służy zaprezentowaniu procesów zachodzących podczas analizy i przetwarzania obrazów cyfrowych. Pozwala również na interakcje z procesem, jego zmianę, zatrzymanie, dostosowanie parametrów oraz podgląd tego co się dzieje w czasie rzeczywistym. Do tego celu zostały wykorzystane następujące technologie: C++/CLI, C#, Windows Presentation Foundation oraz .NET Framework 4.7.2.

2.3 Założenia

Głównym założeniem tworzonego oprogramowania było stworzenie biblioteki w taki sposób, aby osoby z niej korzystające z łatwością mogły dodać ją do swoich projektów w środowisku Visual Studio 2019. Funkcjonalności jakie niesie ze sobą nie skupiają się wyłącznie na analizie i przetwarzaniu obrazów cyfrowych. Biblioteka dba również o to, aby struktura folderów była w należyтым stanie na podstawie głównego folderu. Główny folder powinien gromadzić kolejne foldery, tak zwane eksperymenty, gdzie przez eksperyment rozumiemy folder, w którym znajdują się dane pochodzące z mikroskopu. W każdym z takich folderów dodane zostają odpowiednie pliki odpowiedzialne za ustawienia. Jeżeli dany eksperyment posiada już plik ustawień, jest to jasny sygnał dla programu, że nie należy dodawać kolejnego.

2.4 Podstawowe funkcje biblioteki

1. Analiza drzewa folderów,
2. Zliczanie komórek krwi na obrazach cyfrowych,
3. Zapis wyniku pracy algorytmu zliczającego liczbę komórek krwi w ramach eksperymentu do pliku,
4. Uruchomienie procesu przygotowania katalogu głównego w sposób asynchroniczny,
5. Uruchomienie procesu przetwarzania obrazów w sposób asynchroniczny,
6. Logowanie wykonywanych operacji,
7. Personalizacja procesu przetwarzania danego obrazu przez wybranie operacji, które będą na nim wykonane.

2.5 Podstawowe funkcje aplikacji okienkowej

1. Podgląd obrazów, które powstały po podzieleniu obrazu głównego,
2. Podgląd obrazu, powstałego w procesie binaryzacji,
3. Możliwość śledzenia algorytmu przetwarzania obrazów w czasie rzeczywistym,
4. Możliwość śledzenia algorytmu przygotowania katalogu głównego w czasie rzeczywistym,
5. Możliwość zatrzymania procesów: przetwarzania obrazów oraz przygotowania katalogu głównego.

2.6 Rekomendacje

Użytkownikom oprogramowania rekomenduje się wykorzystywanie biblioteki na systemie operacyjnym Windows 10 w projektach budowanych z wykorzystaniem MSBuild (Microsoft Build Engine) w konfiguracji 32 lub 64-bitowej [1].

3 Część teoretyczna

3.1 Adhezja komórek

Adhezja komórek - to proces, w którym komórki za pomocą cząsteczek znajdujących się w błonie komórkowej, oddziałują między sobą, co umożliwia ich przyleganie. Interakcja między komórkami zachodzi dzięki białkom błony należącym do grupy cząsteczek adhezyjnych CAMs *cell-adhesion molecules* (w tym kadheryny, selektyny, integryny, nadrodzina immunoglobulin), umożliwia tworzenie połączeń międzykomórkowych i ich przyleganie [2]. Adhezja, z łac. *adhaesio* oznacza *przyleganie* [3], w kontekście tej pracy mówimy o większej adhezji, gdy więcej komórek znajduje się na obrazie. Analogicznie mniejsza adhezja oznacza mniejszą liczbę komórek.

3.2 Rozszerzenie TIFF

TIFF - *Tag Image File Format* jest to format plików służący do przechowywania obrazów rastrowych. TIFF ma bardzo szerokie zastosowania, od urządzeń służących do skanowania czy wysyłania faksów do analizy i przetwarzania obrazów cyfrowych. Format stworzony przez *Aldus Corporation*, nieistniejącego już dzisiaj przedsiębiorstwa przejętego w 1994 przez *Adobe*. TIFF jest elastycznym i adaptatywnym typem plików służącym do przechowywania zdjęć bez utraty ich jakości. Dzięki temu, że przechowywane dane nie tracą jakości, TIFF jest używany do archiwizacji zdjęć, ponieważ w przeciwieństwie do rozszerzenia JPEG, TIFF może być zmieniany i zapisywany wielokrotnie bez utraty jakości obrazu. Rozszerzenie to niesie ze sobą możliwość pracy zarówno wykorzystując kolory z palety RGB czy też monochromatyczne. Jest to bardzo użyteczny format, szczególnie w kontekście analizy i przetwarzania obrazów cyfrowych [4].

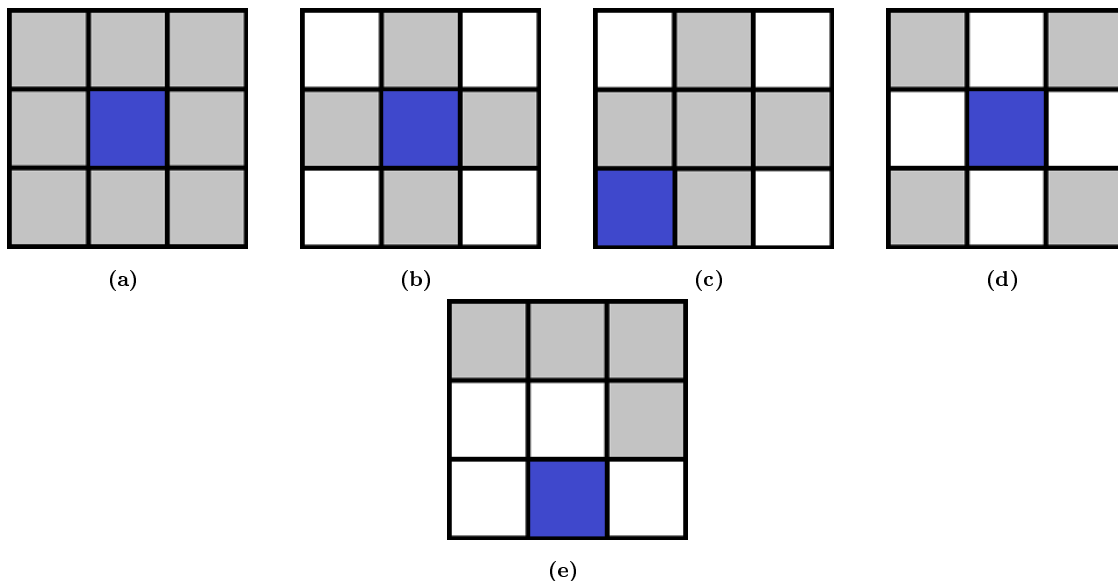
3.3 Binarizacja

Binarizacja - służy do stworzenia nowego obrazu na podstawie innego w taki sposób, aby zaznaczyć interesujący nas fragment na czarno(białe

tło) lub na biało(czarne tło). Jak sama nazwa wskazuje wartości w nowym obrazie są 0 lub 1. Dzięki binaryzacji jesteśmy w stanie przetwarzać obraz tylko dla fragmentów zaznaczonych w procesie binaryzacji. Binarizację można przeprowadzać na przykład na podstawie wartości kolorów w poszczególnych macierzach RGB. Na przykład: jeżeli *czzerwony* > 150, *zielony* > 200 i *niebieski* < 122 to wartość danego piksela w nowym obrazie równa się 1. Teraz wystarczy zastosować to dla każdego piksela w obrazie wejściowym i naszym obrazem wyjściowym staje się obraz o tym samym rozmiarze z zaznaczonym na czarno obszarem, gdzie macierze RGB miały wartości *czzerwony* > 150, *zielony* > 200 i *niebieski* < 122. Tak więc, gdy dokonamy już binaryzacji jesteśmy gotowi do kolejnych działań. Takich jak na przykład etykietowanie, erozja, dylatacja, otwarcie, zamknięcie czy innego rodzaju operacje morfologiczne [5].

3.4 Element Strukturalny

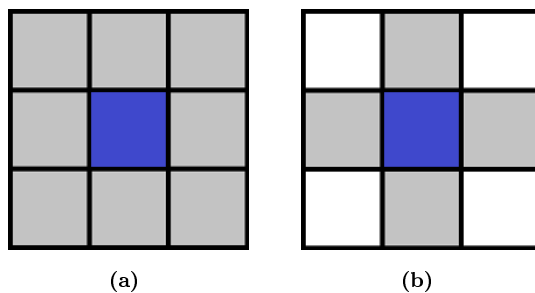
Element Strukturalny - operacje morfologiczne opierają się na elemencie strukturalnym. Przez element strukturalny należy rozumieć ruchome okno, które przykładane jest do każdego piksela w obrazie. Na podstawie wartości w sąsiedztwie wykonywane są pewne operacje. Po przyłożeniu punktu centralnego do piksela, sprawdzana jest wartość tego piksela. Jeżeli wartość ta jest równa 1 to zazwyczaj oznacza to aktywację. Na przykład zamalowanie odpowiednich sąsiadów danego piksela zgodnie z tym, jak wygląda element strukturalny. Każdy z punktów elementu strukturalnego ma odpowiednią wartość. Punkt centralny punktu strukturalnego to miejsce, które przykładane się do kolejnych pikseli w obrazie binarnym, w zależności od operacji morfologicznej, która jest przeprowadzana. Pozostałe piksele przyjmują jakąś wartość 0 lub 1. Przykłady elementów strukturalnych (granatowym kolorem zaznaczono punkty centralne, szarym wartość 1, białym wartość 0) przedstawiono na Rysunkach od 1 [5].



Rysunek 1: (*Opracowanie własne*) Element Strukturalny. (a) Przykład elementu strukturalnego o rozmiarze 3×3 z punktem centralnym w środku. Wszystkie piksele elementu strukturalnego posiadają wartość równą 1. (b) Przykład elementu strukturalnego o rozmiarze 3×3 z punktem centralnym w środku. Lewy dolny, lewy górny, prawy dolny oraz prawy górny piksel posiada wartość 0. (c) Przykład elementu strukturalnego o rozmiarze 3×3 z punktem centralnym w lewym dolnym rogu. Lewy górny, prawy górny oraz prawy dolny piksel posiada wartość 0. (d) Przykład elementu strukturalnego o rozmiarze 3×3 z punktem centralnym w środku. Lewy środkowy, prawy środkowy, dolny środkowy oraz górny środkowy piksel posiada wartość 0. (e) Przykład elementu strukturalnego o rozmiarze 3×3 z punktem centralnym w dolnym środkowym pikselu. Lewy dolny, lewy środkowy, środkowy oraz prawy dolny piksel mają wartość równą 0.

3.5 Sąsiedztwo

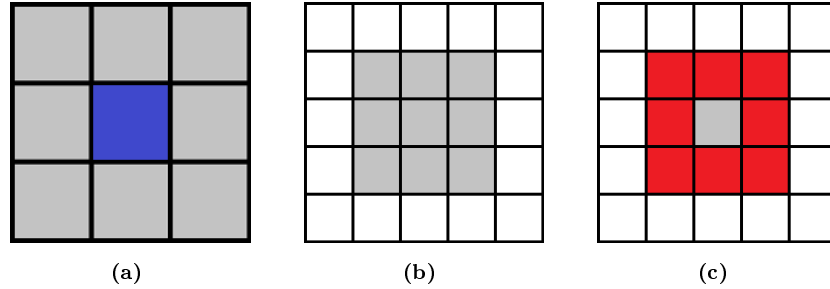
Sąsiedztwo - jest to sposób w jaki zdefiniowane jest otoczenie danego piksela. Wyróżnia się dwa główne sąsiedztwa: Moore'a (Rysunek 2(a)) oraz Von Neumann'a (Rysunek 2(b)) [5].



Rysunek 2: (*Opracowanie własne*) (a) Sąsiedztwo Moore'a. (b) Sąsiedztwo Von Neumann'a.

3.6 Erozja

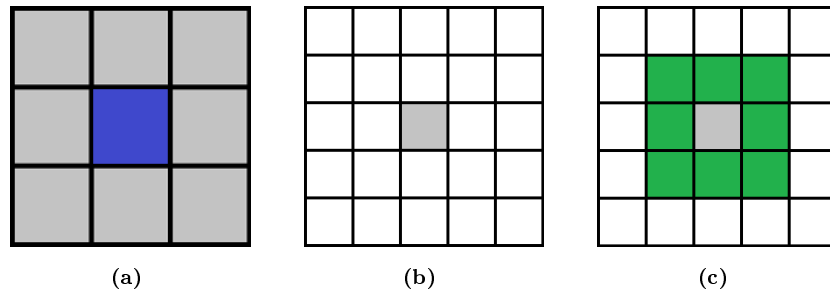
Erozja - czyli zwięzanie, to przyłożenie elementu strukturalnego do każdego piksela p_{ij} na obrazie w celu sprawdzenia czy choć jeden piksel z sąsiedztwa p_{ij} ma wartość równą zero. Jeżeli tak, to punkt centralny również przyjmuje wartość 0. W przeciwnym przypadku wartość pozostaje bez zmian [5].



Rysunek 3: (*Opracowanie własne*) Erozja. (a) Element strukturalny użyty w operacji. (b) Stan przed erozją z wykorzystaniem elementu strukturalnego z podpunktu (a). (c) Stan po erozji z wykorzystaniem elementu strukturalnego z podpunktu (a). Granatowym kolorem zaznaczono punkt centralny elementu strukturalnego, natomiast czerwonym piksele, które zmieniły wartości z 1 na 0.

3.7 Dylatacja

Dylatacja - czyli rozszerzanie, polega na przyłożeniu elementu strukturalnego do każdego piksela p_{ij} na obrazie w celu sprawdzenia czy choć jeden z pikseli z sąsiedztwa p_{ij} ma wartość równą jeden. Jeżeli tak, to punkt centralny również przyjmuje wartość 1, w przeciwnym wypadku wartość nie ulega zmianie [5].



Rysunek 4: (*Opracowanie własne*) Dylatacja. (a) Element strukturalny użyty w operacji. (b) Stan przed dylatacją z wykorzystaniem elementu strukturalnego z podpunktu (a). (c) Stan po dylatacji z wykorzystaniem elementu strukturalnego z podpunktu (a). Granatowym kolorem zaznaczono punkt centralny elementu strukturalnego, natomiast zielonym piksele, które zmieniły wartości z 0 na 1.

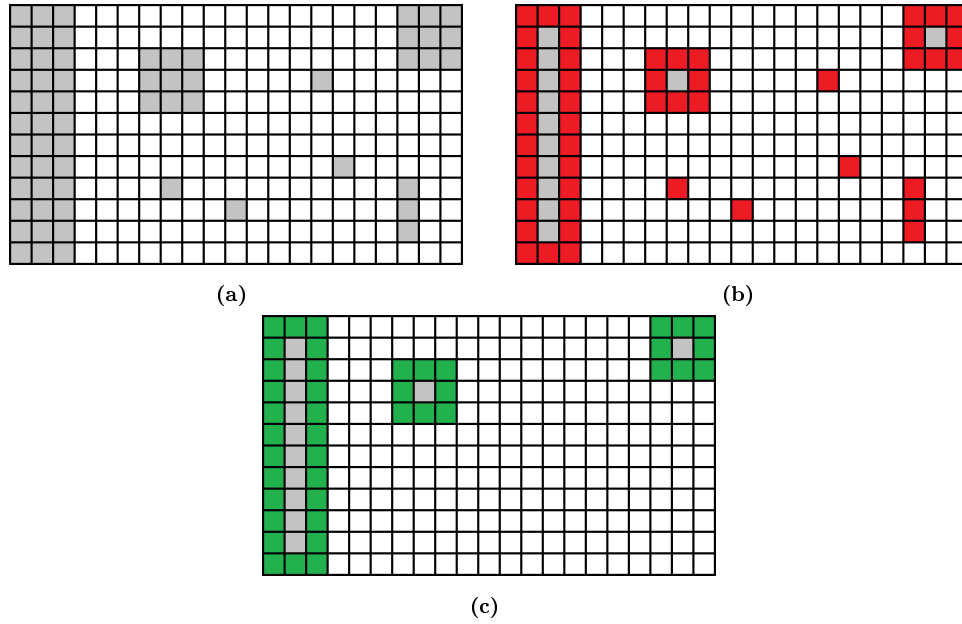
Dylatacja i erozja są operacjami dualnymi. Oznacza to, że jeżeli wykonamy negatyw obrazu, a następnie zadziałamy identycznym elementem

strukturalnym na negatyw, to skutkiem takich działań będzie odwrotność działania erozji z dylatacją. Erozja na negatywie zadziała z takim samym skutkiem, jak dylatacja na oryginale. Natomiast zastosowanie dylatacji na negatywie przyniesie takie same skutki jak wykonanie erozji na oryginale.

Należy zauważyć, że dylatacja i erozja nie są operacjami odwrotnymi. Przykłady powyżej (Rysunek 3, Rysunek 4) mogą coś takiego sugerować, jednak tak nie jest. Dzięki dylatacji możemy zamykać otwory w obiektach znajdujących się na obrazie. Po wykonaniu takiej operacji i zamknięciu obiektu, erozja nie sprawi, że obiekt na nowo się otworzy. To samo tyczy się pojedynczych punktów (piksele, których każdy sąsiad ma wartość 0). Zadziałanie na taki pojedynczy piksel erozją sprawi, że zmieni on wartość z 1 na 0. Nie da się zadziałać dylatacją w taki sposób, aby taki punkt na nowo się pojawił. Pozwala to wyciągnąć następujące wnioski: Dylatacja jest świetnym sposobem na zamykanie drobnych dziur. Erozja jest świetna w usuwaniu małych obiektów. Jednak te operacje mają swoje minusy, nie działamy erozją czy dylatacją tylko na wybrane punkty. Działamy nimi na cały obraz. Właśnie dlatego powstały dwie kolejne operacje: otwarcie oraz zamknięcie.

3.8 Otwarcie morfologiczne

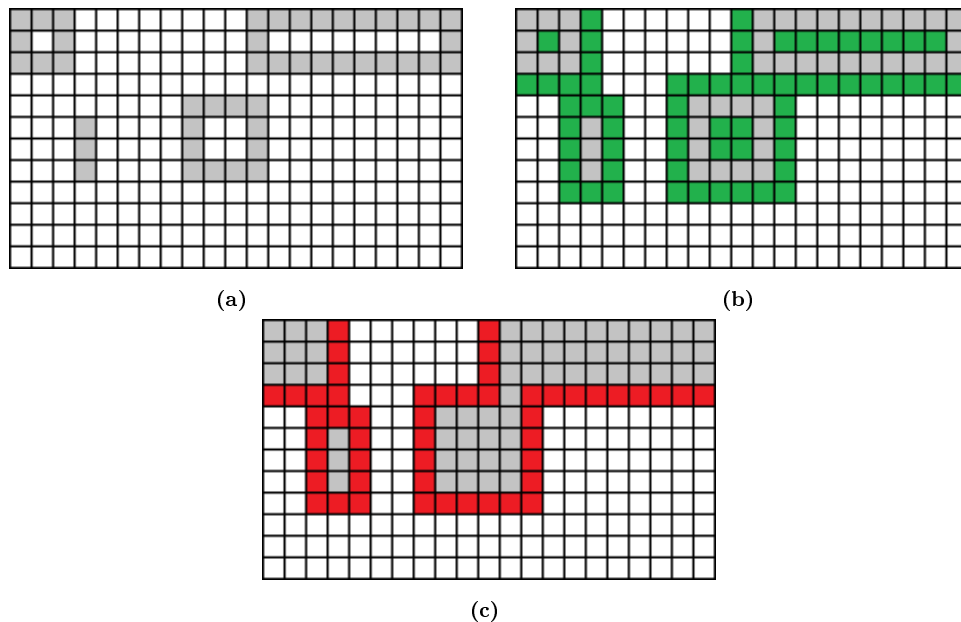
Otwarcie morfologiczne - jest równoważne dwom następującym po sobie operacjom na obrazie: erozji, a następnie dylatacji. Operacja ta pozwala usunąć małe elementy, tak zwany szum i następnie odbudować te obiekty, które ucierpiały na skutek erozji. Dzięki temu uzyskujemy czysty obraz, pozbawiony drobnych obiektów. Jak możemy zauważyć na Rysunku 5(b), pojedyncze piksele (nie posiadające sąsiadów z wartością 1 zostały usunięte), następnie reszta obiektów zostaje odbudowana [5].



Rysunek 5: (*Opracowanie własne*) Otwarcie Morfologiczne. (a) Stan początkowy. (b) Stan po erozji z wykorzystaniem elementu strukturalnego z rysunku 1(a). (c) Stan po dylatacji z wykorzystaniem elementu strukturalnego z rysunku 1(a). Szarym kolorem zaznaczono piksele, które posiadają wartość 1, czerwonym te, które zmieniły wartość z 1 na 0. Zielonym natomiast zaznaczone te piksele, które zmieniły wartość z 0 na 1.

3.9 Zamknięcie morfologiczne

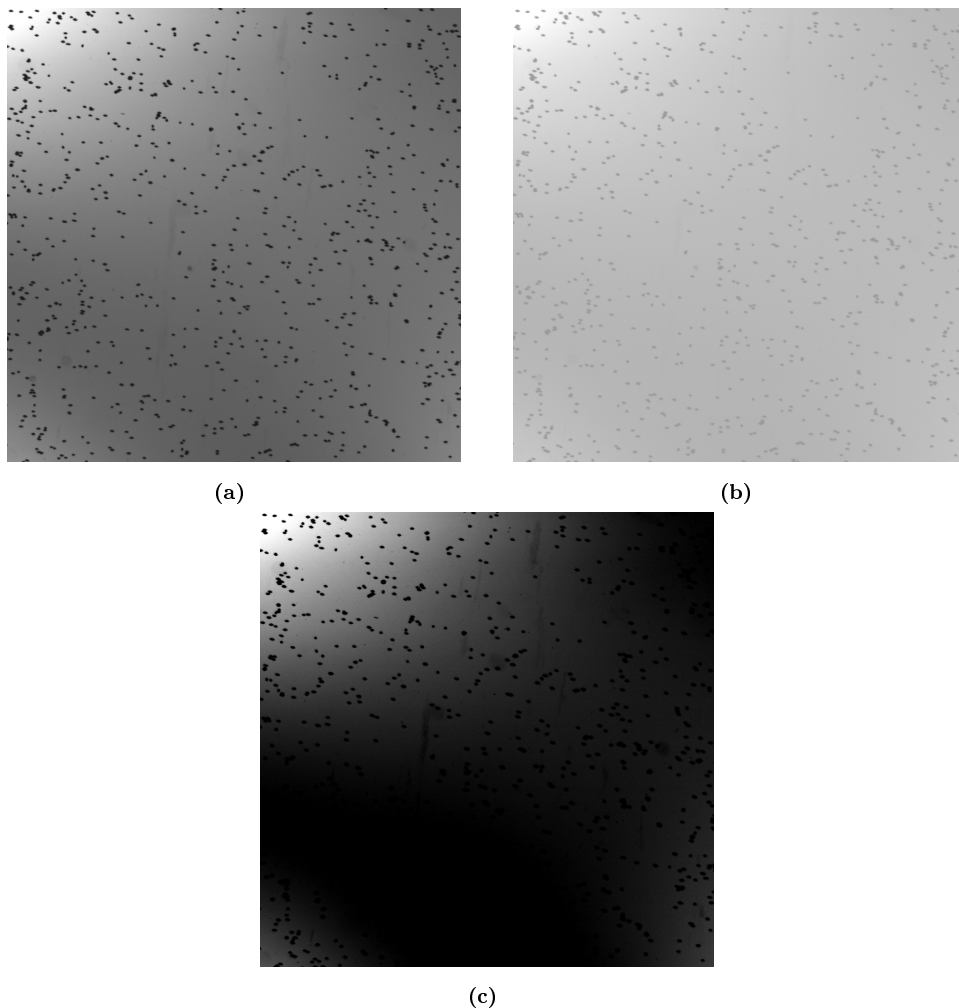
Zamknięcie morfologiczne - jest równoważne dwom następującym po sobie operacjom na obrazie: dylatacji, a następnie erozji. Wykonanie dylatacji, a następnie erozji pozwala na zamknięcie (przez zamknięcie należy rozumieć wypełnienie otworu wewnątrz obiektu, tak jak pokazano na Rysunku 6(b)) niektórych obiektów, a następnie przywrócenie pozostałych obiektów do stanu sprzed dylatacji. Jak możemy zauważyć na Rysunku 6(c), piksele, które wcześniej były odłączone od większego obiektu ale leżały niedaleko teraz stanowią integralną jego część, a dwa większe obiekty połączyły się [5].



Rysunek 6: (*Opracowanie własne*) Zamknięcie Morfologiczne. (a) Stan początkowy. (b) Stan po dylatacji z wykorzystaniem elementu strukturalnego z rysunku 1(a). (c) Stan po erozji z wykorzystaniem elementu strukturalnego z rysunku 1(a). Szarym kolorem zaznaczono piksele, które posiadają wartość 1, czerwonym te, które zmieniły wartość z 1 na 0. Zielonym natomiast zaznaczone te piksele, które zmieniły wartość z 0 na 1.

3.10 Korekcja Gamma

Korekcja Gamma - operacja punktowa wykonywana na obrazie monochromatycznym, która bazuje na krzywej gamma i wyraża się ją za pomocą wzoru $L'(x, y) = L(x, y)^\gamma$. Zastosowanie $\gamma = 1$ sprawi, że obraz pozostanie bez zmian. Natomiast w przypadku, gdy $\gamma > 1$ obraz zostanie przyciemniony. Analogicznie jest, gdy $\gamma < 1$, to obraz zostanie rozjaśniony [5].



Rysunek 7: (*Opracowanie własne*) Korekcja Gamma. (a) Stan początkowy. (b) Stan po korekcji gamma z parametrem 0.85 . (c) Stan po korekcji gamma z parametrem 1.10 .

4 Część Praktyczna

4.1 Wykorzystane oprogramowanie

4.1.1 Gimp 2.10.22

Gimp 2.10.22 - program graficzny służący do tworzenia i edycji grafiki rastrowej [8]. Program został wykorzystany do przygotowania wszystkich grafik znajdujących się w pracy oznaczonych słowami “*opracowanie własne*”.

4.1.2 CMake 3.19.0

CMake 3.19.0 - oprogramowanie, które służy do testowania, budowania oraz upakowania źródeł programu. CMake został wykorzystany do wygenerowania projektów na podstawie źródeł biblioteki *LibTIFF*. Wyżej wy-

mienione projekty mogą można otworzyć, a następnie zbudować za pomocą programu Visual Studio 2019 [9].

4.1.3 LibTIFF 4.2.0

LibTIFF 4.2.0 - biblioteka służąca do manipulacji obrazami typu TIFF, typ ten dokładniej został opisany w rozdziale 3.2. Biblioteka została wybrana, ponieważ pozwala w łatwy sposób odczytywać informacje zapisane w obrazach. Napisana jest w języku C, więc w łatwy sposób można wykorzystać ją w programach pisanych w języku C++ [10].

4.1.4 Visual Studio 2019

Visual Studio 2019 - doskonały program do pisania oprogramowania wykorzystując technologie pochodzące od firmy *Microsoft* [11]. W Visual Studio 2019 jak nigdzie indziej można z powodzeniem łączyć języki C++ oraz C# za pomocą C++/CLI. Dzięki takim możliwościom, można z powodzeniem tworzyć oprogramowanie składające się z wielu mniejszych programów, które napisane są w różnych językach. Dla przykładu: tworzymy bibliotekę napisaną w C++, ponieważ zależy nam na tym, aby operacje wykonywały się bardzo szybko. Interfejs użytkownika jest napisany w C#, ponieważ robi się to w prosty i szybki sposób. Następnie całość łączymy wykorzystując C++/CLI. Wynikiem takiego działania będzie program posiadający interfejs graficzny i łączący w sobie C++ oraz C#.

4.1.5 L^AT_EX

L^AT_EX- wysokiej jakości system służący do tworzenia dokumentów w przystępny sposób [12]. Dzięki L^AT_EX, proces tworzenia spisu treści, spisu grafik czy bibliografii jest automatyzowany. Odwołania do konkretnych rozdziałów, podrozdziałów czy źródeł to kwestia wywołania funkcji *cite* lub *ref*. Jest standardem publikacji dokumentów naukowych i technicznych.

4.1.6 Visual Studio Code

Visual Studio Code - uniwersalny edytor z ogromną bazą rozszerzeń, dostępny na każdej z platform *Windows*, *Linux* oraz *MacOS*, umożliwiający tworzenie oprogramowania niemal w każdym języku [13]. W swojej pracy wykorzystałem *Visual Studio Code* jako edytor tekstowy plików z rozszerzeniem *tex*, które następnie były wykorzystywane przez program L^AT_EX(4.1.5).

4.1.7 .NET Framework 4.7.2

.NET Framework 4.7.2 - oprogramowanie stworzone przez firmę *Microsoft* pozwalające na tworzenie szerokiej gamy aplikacji, które działają głównie na systemach *Windows*. W kontekście tej pracy .NET Framework 4.7.2 został wykorzystany w celu uruchomienia zarówno kodu zarządzalnego (napisanego w języku C#) oraz natywnego (napisanego w języku C++) [14].

4.1.8 nlohmann.json 3.9.1

nlohmann.json 3.9.1 - prosta w obsłudze biblioteka pozwalająca na tworzenie, modyfikowanie oraz odczytywanie informacji z obiektów typu *json* [15].

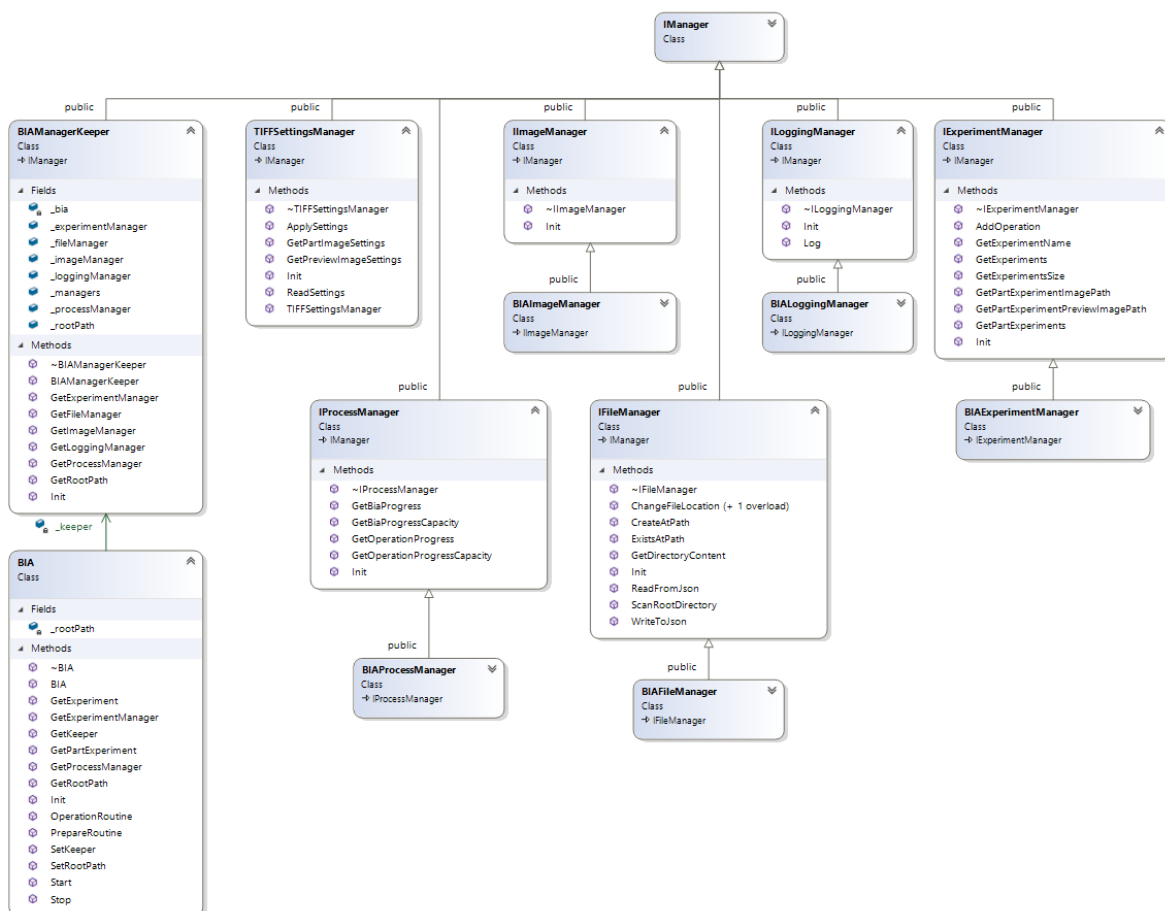
4.1.9 OpenMP 5.1

OpenMP 5.1 - oprogramowanie służące do programowania równoległego [16]. W bibliotece, o której traktuje ta praca, *OpenMP 5.1* zostało wielokrotnie wykorzystane do zrównoleglania pętli *for*. Dzięki temu niektóre operacje wykonują się kilka razy szybciej.

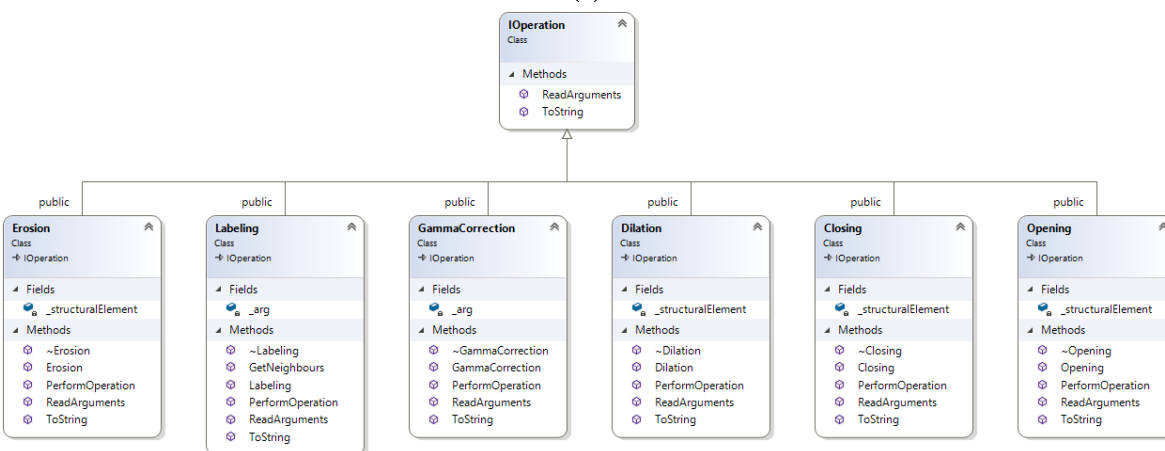
4.2 Budowa oprogramowania

Aby wykorzystać bibliotekę BIA.Core w swoim projekcie, należy najpierw zbudować tę bibliotekę. Kolejnym etapem jest wykorzystanie plików zawartych w katalogu, do którego biblioteka zostanie zbudowana, oraz dodanie pliku "BIA.h" za pomocą dyrektywy *#include*. Wykorzystanie biblioteki LibTIFF 4.2.0 w programie pisanym w języku C++ opartym o MSBuild polega na tym, że należy najpierw ręcznie zbudować tę bibliotekę przy użyciu narzędzia *nmake* oraz *Command Prompt for Visual Studio 2019* [10]. W związku z powyższym, w programie *Visual Studio 2019* trzeba dodać ścieżkę do miejsca, w którym znajduje się zbudowana przez nas biblioteka. Przykładowo "D:/tiff-4.2.0/libtiff".

4.3 Diagramy klas



(a)



(b)

Rysunek 8: (*Opracowanie własne*) Diagramy klas biblioteki BIA.Core.

(a) Główny diagram klas. Przedstawione na nim klasy odpowiadają za najważniejsze funkcje biblioteki.

(b) Diagram klas przedstawiający operacje, których wykonanie na obrazach umożliwia biblioteka.

4.4 Omówienie implementacji

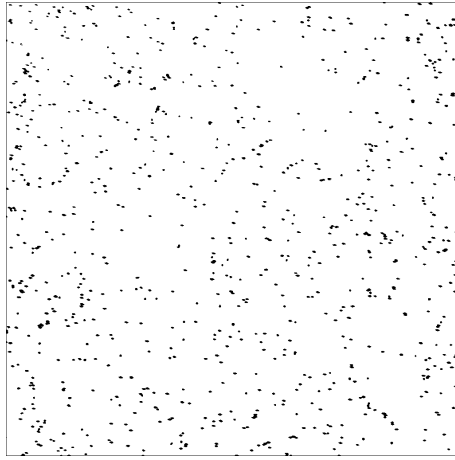
Biblioteka, która opisywana jest w tej pracy nosi nazwę “BIA.Core”. Całe rozwiązanie składa się jeszcze z dwóch innych projektów: “BIA.UI” oraz “BIA.Bridge”. “BIA.UI” to biblioteka napisana w języku C#, stanowiąca interfejs użytkownika. Służy do pokazania funkcjonalności biblioteki “BIA.Core”. Wspomniany wcześniej projekt “BIA.Bridge” stanowi połączenie (*bridge* z ang. *most*) między interfejsem użytkownika, a biblioteką “BIA.Core”. Główną klasą biblioteki “BIA.Core” jest klasa “BIA” (Rysunek 8 podpunkt (a)). Nazwa jej pochodzi od angielskich słów “Blood Image Analyzer”, czyli “Analizator Obrazów Krwi”. Głównym zadaniem klasy “BIA” jest zebranie wszystkich funkcjonalności biblioteki w wygodny do użycia sposób. Zawiera ona prywatne pole o nazwie “_rootPath”, które jest odpowiedzialne za przetrzymywanie ścieżki do wybranego przez nas katalogu. W bibliotece “BIA.Core” nazwy wszystkich interfejsów zaczynają się od litery “I”. “BIA” posiada wskaźnik typu “BIAManagerKeeper”, który zawiera w sobie klasy implementujące odpowiednie interfejsy. Jest odpowiedzialny za ich inicjalizację oraz późniejsze wykorzystanie. Każdy z interfejsów niesie ze sobą funkcjonalności w zależności od nazwy: “IImageManager” - zarządzanie obrazami, “IExperimentManager” - zarządzanie eksperymentami, “IFileManager” - zarządzanie plikami, “IProcessManager” - zarządzanie asynchronicznymi procesami, “ILoggingManager” - logowanie do pliku. Każdy z menadżerów implementuje odpowiedni interfejs: “BIAImageManager” implementuje “IImageManager”, “BIAExperimentManager” implementuje “IExperimentManager”, “BIAFileManager” implementuje “IFileManager”, “BIAProcessManager” implementuje “IProcessManager”, “BIALoggingManager” implementuje “ILoggingManager”. Klasa “BIA” pozwala na wykorzystanie niektórych funkcjonalności menadżerów poza biblioteką “BIA.Core”. Służą do tego funkcje: “GetExperimentManager”, która zwraca wskaźnik typu “IExperimentManager”, “GetProcessManager”, która zwraca wskaźnik typu “IProcessManager”. Z punktu widzenia użytkownika biblioteki wykorzystanie pozostałych menadżerów nie jest konieczne, ponieważ wykonują swoje operacje automatycznie - w zależności od kontekstu. Aby poprawnie skorzystać z funkcjonalności klasy “BIA” należy po stworzeniu obiektu tej klasy wywołać funkcję “Init”. Funkcja ta odpowiedzialna jest za utworzenie wszystkich potrzebnych obiektów, w tym inicjalizację “BIAManagerKeeper”. Gdy “BIAManagerKeeper” zostanie zainicjalizowany, utworzeni zostają pozostali menadżerowie. Domyślny konstruktor klasy “BIA” przyjmuje jako parametr ścieżkę do katalogu, która następnie zostaje zapisana do zmiennej “_rootPath”. Jest to bardzo ważna rzecz, gdyż od

tego zależy poprawne funkcjonowanie całej biblioteki. Destraktor “~BIA” działa w taki sposób, że gdy zostanie wywołany to usunięte zostają wszystkie obiekty, których używa biblioteka. Zapobiega to wyciekaniu pamięci, a także jej niekontrolowanemu wykorzystaniu. Funkcja “GetKeeper” zwraca wskaźnik do obiektu “BIAManagerKeeper”. “GetRootPath” pozwala na otrzymanie ścieżki, do głównego katalogu. “OperationRoutine” wykonuje operacje (operacje zapisywane są w pliku “recipe.json”) na każdym z obrazów. “PrepareRoutine” dzieli główne obrazy na mniejsze i przygotowuje wszystkie potrzebne do późniejszej fazy pliki. Obydwie powyższe metody można uruchomić w sposób asynchroniczny za pomocą “BIAProcessManager”, nie jest to jednak przymus. “SetKeeper” pozwala na ustawienie obiektu “BIAManagerKeeper”. “SetRootPath” pozwala na zmianę ścieżki do katalogu głównego. “Start” odpowiada za uruchomienie odpowiedniego procesu. Jako argument przyjmuje typ wyliczeniowy o nazwie “EProcess”, jeżeli ustalimy jego wartość na “EProcess::BIAPROCESS” to “BIAProcessManager” uruchomi asynchronicznie funkcję “PrepareRoutine”. W przypadku wartości “EProcess::BIAOPERATIONS” uruchomiona zostanie funkcja “OperationRoutine”, również w sposób asynchroniczny. Funkcja “OperationsRoutine” wykonuje operacje zapisane w pliku “recipe.json” dla każdego obrazu. Funkcja “Stop” pozwala na zatrzymanie operacji niezależnie od tego, którą z nich uruchomiliśmy wcześniej. Zatrzymanie procesów pozwala na oszczędzenie sporej ilości czasu w przypadku, gdy przewidujemy, że proces nie przyniesie pożądanych efektów i chcemy nanieść korekty. Zapisana do zmiennej “_rootPath” ścieżka będzie postrzegana jako źródło tak zwanych “eksperymentów”. Przez “eksperyment” należy rozumieć katalog, w którym znajdują się dwa zdjęcia pochodzące z mikroskopu. Zdjęcia te powinny wcześniej zostać wyeksportowane do plików o typie “.tif” oraz nazwane w odpowiedni sposób. Zdjęcie zorientowane horyzontalnie powinno zawierać w swojej nazwie słowo “horizontal”, natomiast te zorientowane wertykalnie - “vertical”. Dzięki temu obrazy zostaną zlokalizowane, odpowiednie nazewnictwo pozwoli na zastosowanie odpowiednich ustawień do danego ułożenia obrazu. Wewnątrz eksperymentu utworzone zostaną dwa katalogi: “Horizontal” oraz “Vertical”. W kolejnym kroku utworzone zostaje po czterdzieści katalogów wewnątrz każdego z podkatalogów eksperymentu. Do każdego z tych podkatalogów trafia fragment obrazu głównego, który będzie podlegał różnym operacjom. Na potrzeby biblioteki, dla każdego z wydzielonych obrazów utworzone zostają też pliki: “preview.tif”, “recipe.json” oraz “results.json”. Plik “preview.tif” to zbinaryzowany obraz wynikowy, “recipe.json” zawiera spis operacji, które mają

zostać wykonane na obrazie binarnym, “results.json” to plik, do którego zapisywane są wyniki operacji etykietowania (ang *labeling*). Etykietowanie to operacja, która pozwala na zliczanie obiektów znajdujących się na obrazie binarnym w zależności od tego jaką wartość przyjmujemy jako tło. W przypadku tej biblioteki tło jest zaznaczane białym kolorem. Kod źródłowy z racji swojej znaczącej objętości został umieszczony jako załącznik do tej pracy. Wszystkie algorytmy są autorskie.

IProcessManager - zaimplementowany przez klasę “BIAProcessManager” pozwala na śledzenie postępu wykonywanych operacji. Opis funkcji: “GetBiaProgress” pozwala na otrzymanie aktualnego postępu w przygotowywaniu głównego katalogu. “GetBiaProgressCapacity” zwraca całkowitą ilość operacji przygotowywania katalogu głównego. “GetOperationProgress” zwraca aktualny postęp procesu przetwarzania obrazów. “GetOperationProgressCapacity” zwraca całkowitą ilość operacji przetwarzania obrazów.

IExperimentManager - zaimplementowany przez klasę “BIAExperimentManager” pozwala na wykonywanie pewnych funkcji na eksperymentach. Opis funkcji: “AddOperation” pozwala na dodanie operacji do danego eksperymentu częściowego. “GetExperimentName” zwraca nazwę eksperymentu na podstawie identyfikatora. “GetExperiments” zwraca wszystkie istniejące eksperymenty. “GetExperimentsSize” zwraca ilość eksperymentów. “GetPartExperimentImagePath” zwraca ścieżkę do danego obrazu na podstawie nazwy eksperymentu, identyfikatora eksperymentu częściowego oraz tego, czy obraz ten należy do folder “Vertical” czy “Horizontal”. “GetPartExperimentPreviewImagePath” podobnie jak w przypadku poprzedniej funkcji na podstawie takich samych argumentów zwraca ścieżkę do obrazu wyjściowego (powstałego po podzieleniu obrazu głównego). “GetPartExperiments” zwraca eksperymenty częściowe na podstawie identyfikatora eksperymentu oraz przynależności do folder “Vertical” lub “Horizontal”.



(a)

```
{
  "OPERATIONS": [
    {
      "OPERATION": {
        "NAME": "OPENING",
        "ARGS": "3,3"
      }
    },
    {
      "OPERATION": {
        "NAME": "LABELING",
        "ARGS": "VONNEUMANN"
      }
    }
  ],
  "THRESHOLD": 200
}
```

(b)

```
"265": {
  "698367": {
    "x": 1023,
    "y": 681
  },
  "699390": {
    "x": 1022,
    "y": 682
  },
  "699391": {
    "x": 1023,
    "y": 682
  },
  "700414": {
    "x": 1022,
    "y": 683
  },
  "700415": {
    "x": 1023,
    "y": 683
  },
  "size": "5px"
},
```

(c)

```
},
  "size": "23px"
},
  "96": { ...
},
  "97": { ...
},
  "98": { ...
},
  "99": { ...
},
  "area": "21825px",
  "cells": 772
}
```

(d)

Rysunek 9: (Opracowanie własne) Przykłady plików.

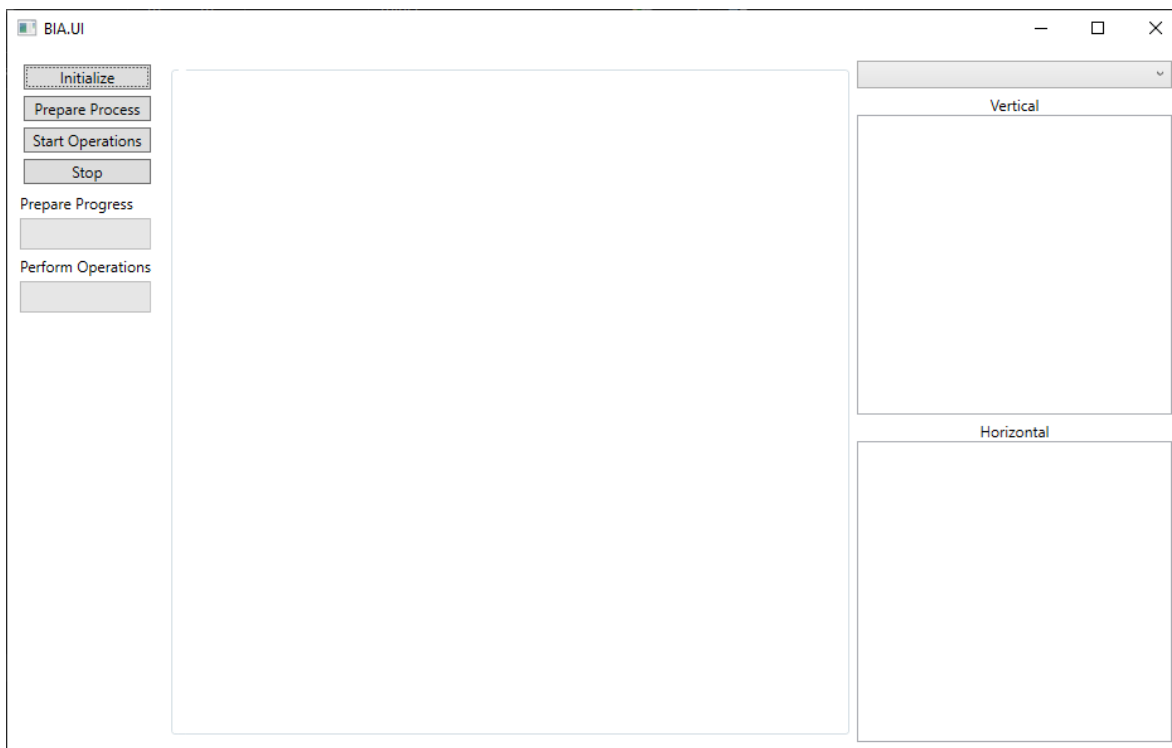
(a) Zbinaryzowany obraz wynikowy.

(b) Przykład pliku “recipe.json”. Tablica “OPERATIONS” zawiera listę operacji, które mają zostać wykonane na danym obrazie. W tym przypadku zostaną wykonane dwie operacje: Otwarcie morfologiczne (rozdział 3.8) elementem strukturalnym o rozmiarach 3 na 3. Drugą operacją jest etykietowanie wykorzystując sąsiedztwo Vonneumann’a (rozdział 3.5). Zapisana w polu “THRESHOLD” wartość stanowi argument operacji binaryzacji.

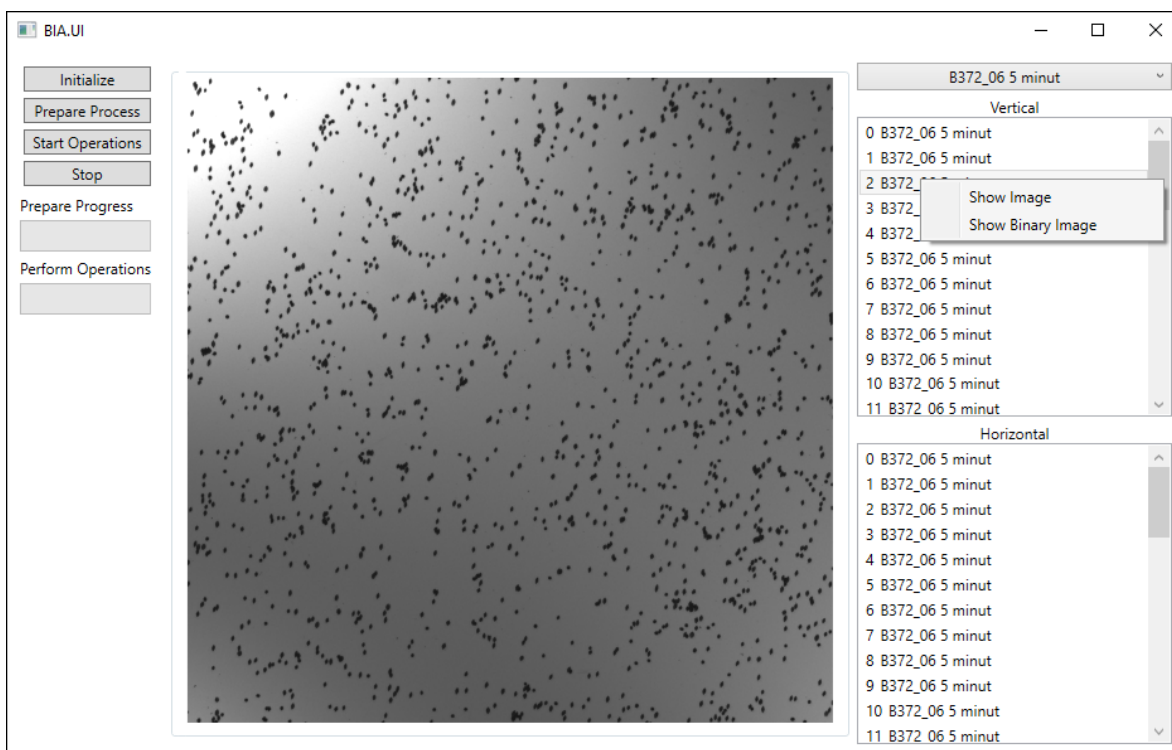
(c) Przykład pliku “results.json”. Na rysunku przedstawiony jest obiekt, któremu nadano etykietę “265”. Piksele, z których dany obiekt się składa mają odpowiednio indeksy “698367”, “699390”, “699391”, “700414”, “700415”. Każdy z nich posiada współrzędne zapisane w polach “x” oraz “y”. Pole “size” zawiera informacje o tym jaki rozmiar ma dany obiekt.

(d) Przykład zakończenia pliku “results.json”. Pole “area” zawiera sumę powierzchni obiektów znalezionych na obrazie, Natomiast pole “cells” to liczba znalezionych obiektów (obiekty o etykietach “96”, “97”, “98”, “99” mogą sugerować, że obiektów jest tylko 99 pomimo wartości “cells” 772. Należy jednak pamiętać o tym, że obiekty są zapisywane do pliku w kolejności losowej).

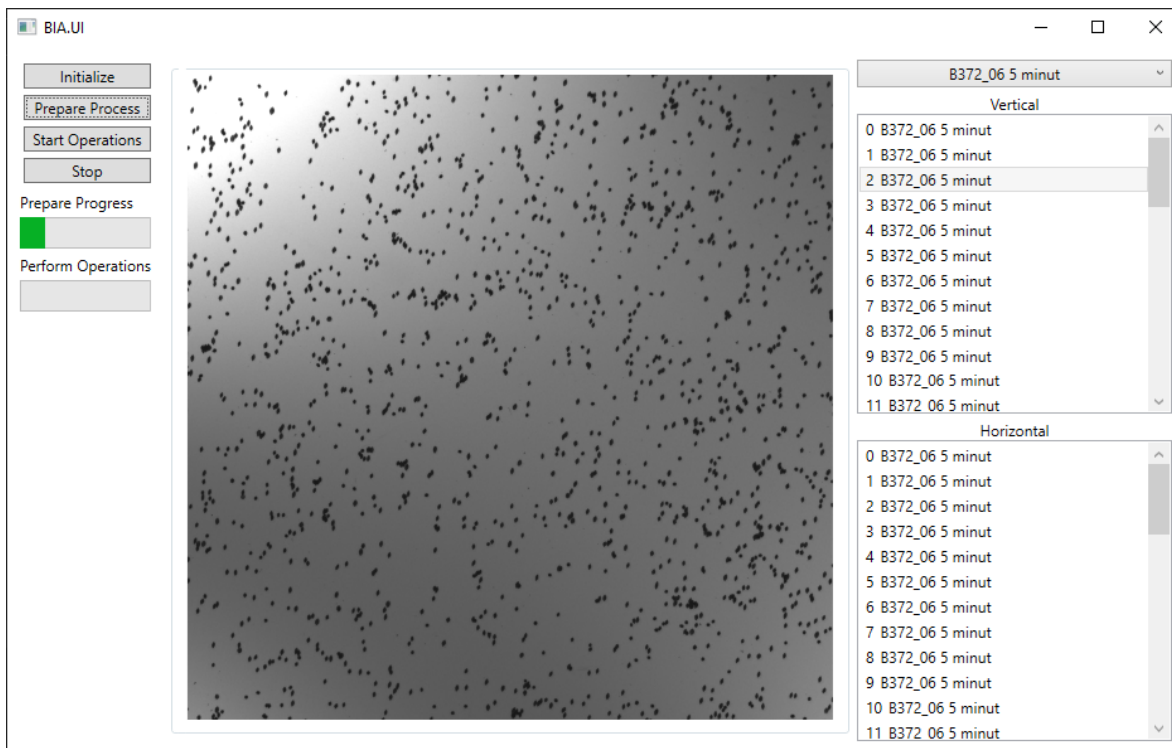
4.5 Omówienie interfejsu aplikacji okienkowej



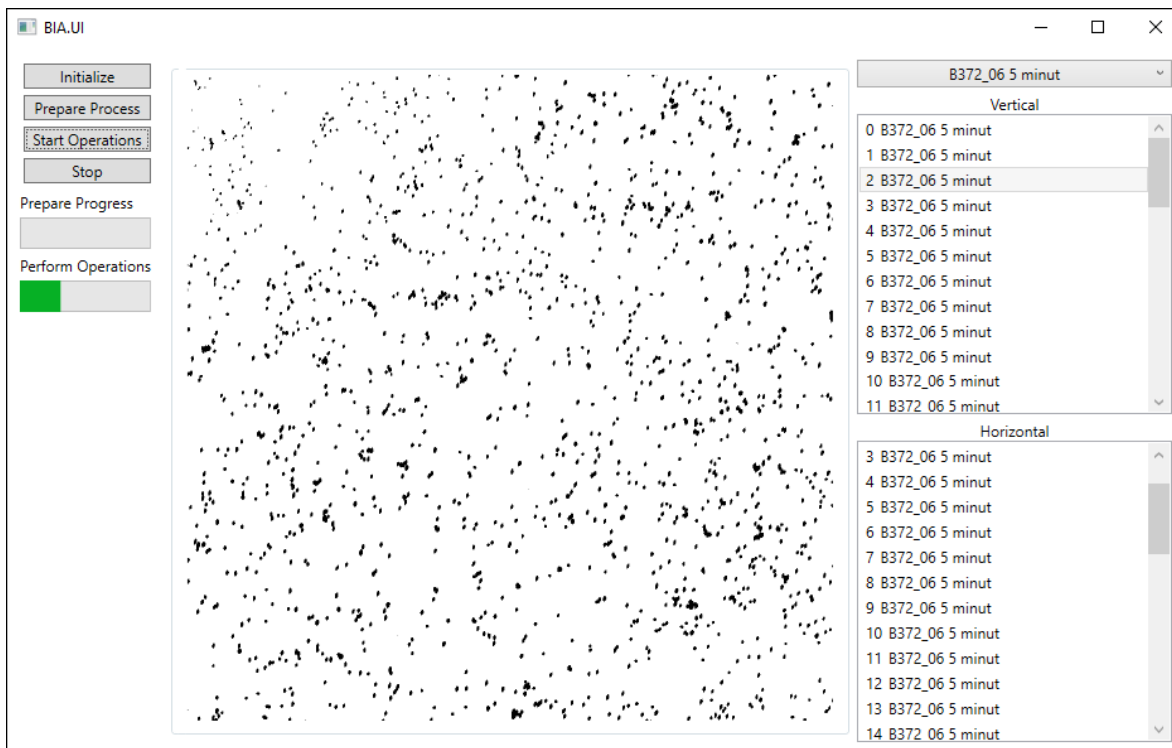
Rysunek 10: (Opracowanie własne) Widok aplikacji okienkowej tuż po jej uruchomieniu.



Rysunek 11: (Opracowanie własne) Widok aplikacji okienkowej po kliknięciu przycisku “Initialize” oraz wybraniu eksperymentu “B372_06 5 minut” oraz kliknięciu prawym przyciskiem myszy na jeden z eksperymentów częściowych.



Rysunek 12: (Opracowanie własne) Widok aplikacji okienkowej po kliknięciu przycisku “Prepare Process”. Poniżej napisu “Prepare Process” można obserwować postęp tego procesu.

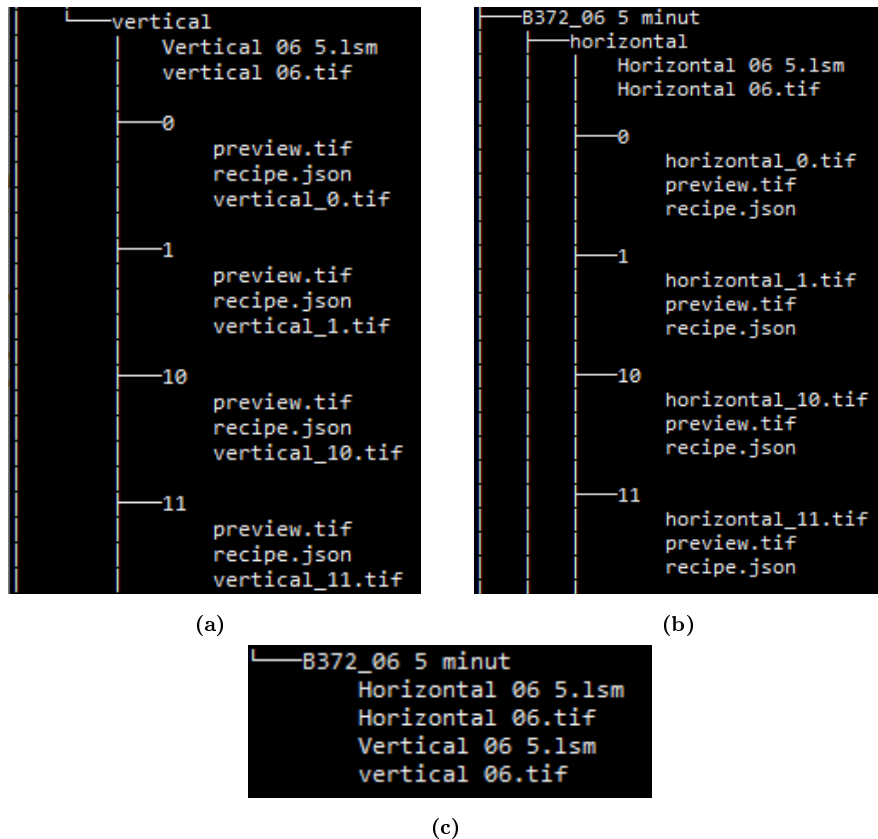


Rysunek 13: (Opracowanie własne) Widok aplikacji okienkowej po kliknięciu przycisku “Start Operations”. Poniżej napisu “Perform Operations” można obserwować postęp tego procesu.

Jak można zauważyć na rysunkach 10, 11, 12 oraz 13 aplikacja okienkowa składa się z kilku elementów. Cztery przyciski po lewej stronie odpowiadają za: “Initialize” inicjalizację obiektu klasy “BIA” pochodzącej z biblioteki “Bia.Core”, “Perpare Process” uruchamia asynchronicznie proces przygotowania katalogu głównego. “Start Operations” uruchamia asynchronicznie proces przetwarzania obrazów, “Stop” umożliwia zatrzymanie wyżej wymienionych procesów. Dwa paski postępu poniżej informują nas o postępie wykonania odpowiedniego procesu. Na rysunku 12 widać postęp w wykonaniu procesu “Prepare Process”, z kolei na rysunku 13 postęp wykonania procesu przetwarzania obrazów. Środek okna zajmuje kontrolka służąca do wyświetlania obrazów. Obraz ładowany jest w momencie gdy wybierzemy za pomocą menu kontekstowego opcje “Show Image” lub “Show Binary Image” tak jak na rysunku 11. Po prawej stronie aplikacji znajduje się kontrolka o nazwie “ComboBox”, która służy do wyboru danego eksperymentu za pomocą jego nazwy. Poniżej można znaleźć dwie kontrolki o nazwie “ListBox”, które służą do wyświetlania list elementów. W kontekście tej aplikacji wyświetlane są eksperymenty częściowe, które należą do wybranego wcześniej eksperymentu.

5 Wyniki i dyskusja

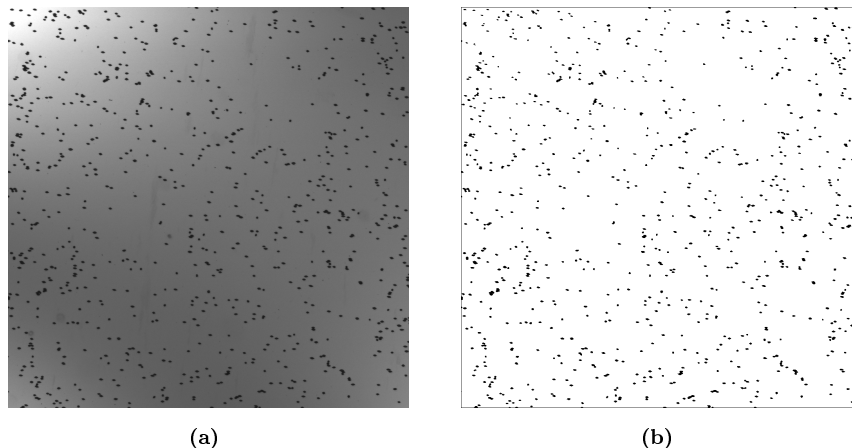
5.1 Przygotowanie katalogu głównego



Rysunek 14: (*Opracowanie własne*) Przykłady przygotowania katalogu głównego.
(a) Folder “vertical” po przeprowadzeniu procesu przygotowania katalogu głównego.
(b) Folder “horizontal” po przeprowadzeniu procesu przygotowania katalogu głównego.
(c) Folder główny przed uruchomienie procesu przygotowania katalogu głównego.

Na rysunku 14 możemy zauważyć jak wygląda drzewo folderów po przeprowadzeniu procesu przygotowania katalogu głównego. Nazwa eksperymentu to “B372_06 5 minut” co możemy zauważyć na rysunku 14 (b). W tym eksperymencie zostały stworzone dwa foldery “vertical” oraz “horizontal”. W każdym z tych folderów znajduje się czterdzieści eksperymentów częściowych noszących nazwy od 0 do 39. Każdy z eksperymentów częściowych posiada pliki “preview.tif”, “recipe.json” oraz obraz wynikowy (powstały z podzielenia obrazu głównego). W ramach eksperymentu wyróżniamy dwa obrazy główne, w tym przypadku - “Vertical 06.tif” oraz “Horizontal 06.tif”. Powstają z nich obrazy wynikowe, które trafiają do poszczególnych eksperymentów częściowych. Ich przykładowe nazwy to “vertical_0.tif” czy “horizontal_0.tif”.

5.2 Przykład Binaryzacji

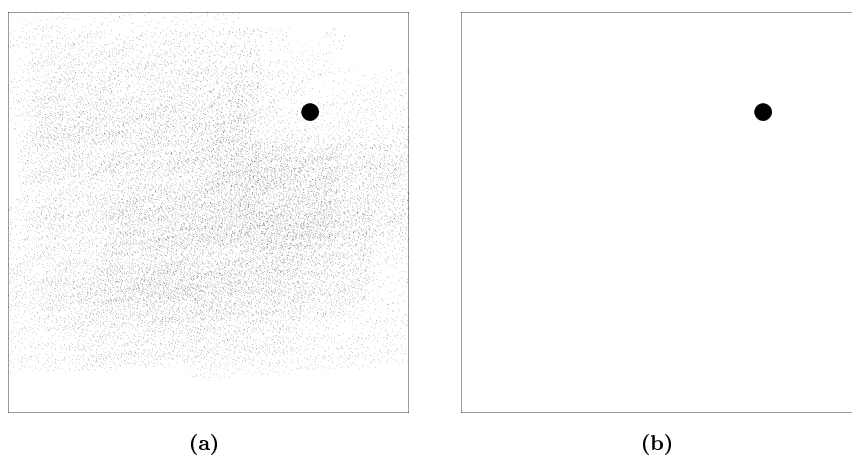


Rysunek 15: (*Opracowanie własne*) Przykład operacji binaryzacji.

(a) Obraz wynikowy (powstały po podzieleniu obrazu głównego)

(b) Obraz powstały w wyniku procesu binaryzacji obrazu wynikowego. Parametr binaryzacji odczytywany jest z pliku “recipe.json”, który zapisany jest w sposób jaki pokazano na rysunku 9(b).

5.3 Przykład Otwarcia

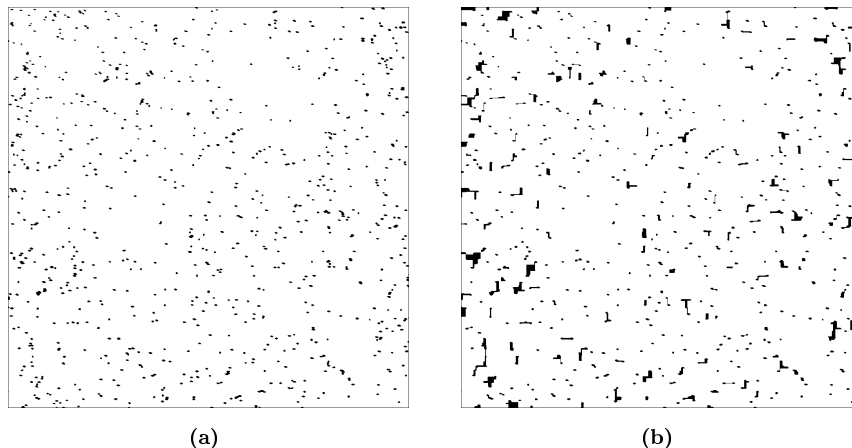


Rysunek 16: (*Opracowanie własne*) Przykład operacji otwarcia.

(a) Przykładowy obraz zawierający jeden duży obiekt oraz szum.

(b) Obraz powstały w wyniku operacji otwarcia na obrazie przykładowym elementem strukturalnym o rozmiarach 3 na 3. Szum został usunięty.

5.4 Przykład Zamknięcia

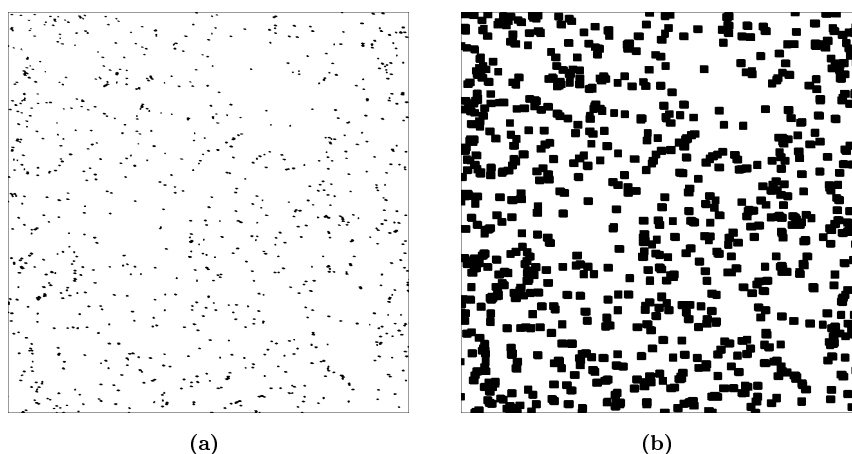


Rysunek 17: (*Opracowanie własne*) Przykład operacji zamknięcia.

(a) Przykładowy obraz.

(b) Obraz powstały w wyniku operacji zamknięcia na obrazie przykładowym elementem strukturalnym o rozmiarach 9 na 9. Niektóre obiekty zostały połączone i zwiększyły swoją powierzchnię.

5.5 Przykład Dylatacji

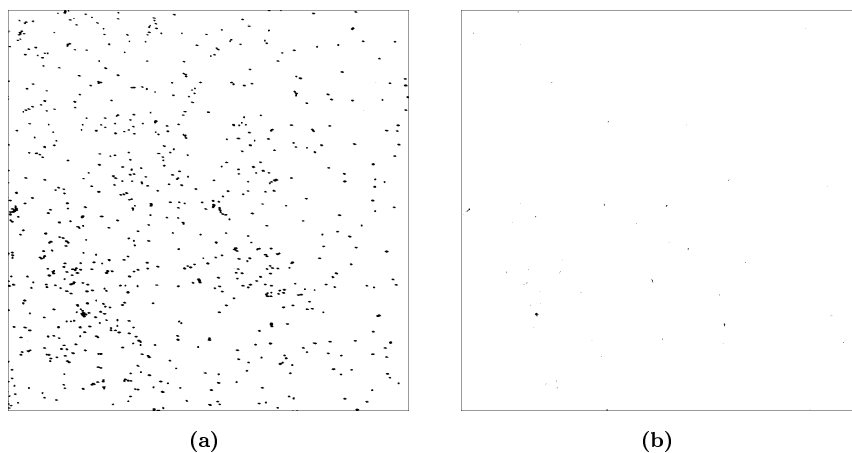


Rysunek 18: (*Opracowanie własne*) Przykład operacji dylatacji.

(a) Przykładowy obraz.

(b) Obraz powstały w wyniku operacji dylatacji na obrazie przykładowym elementem strukturalnym o rozmiarach 9 na 9. Wszystkie obiekty wyraźnie zwiększyły swoją powierzchnię.

5.6 Przykład Erozji

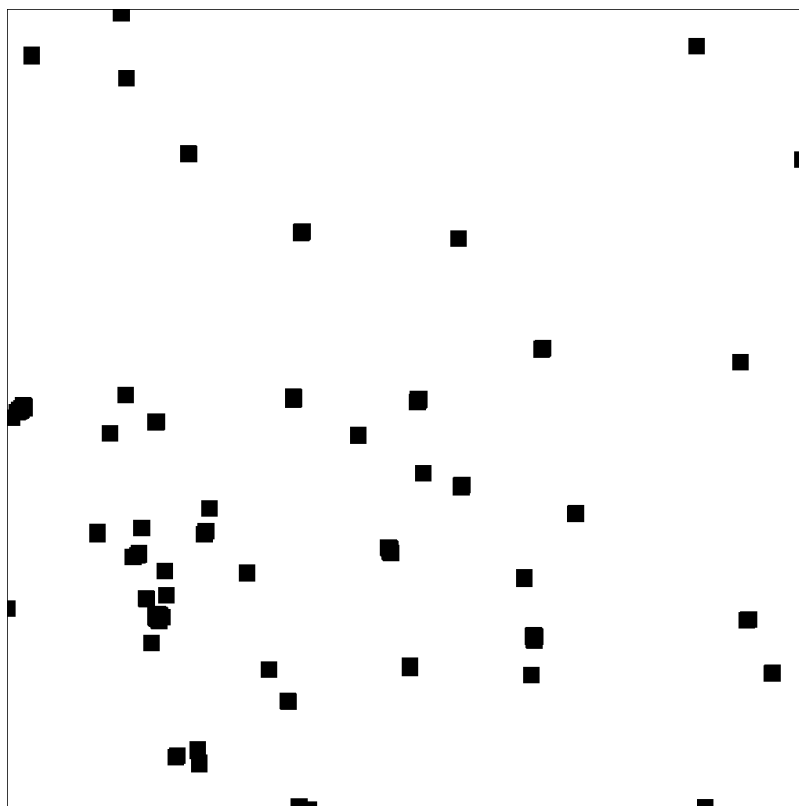


Rysunek 19: (*Opracowanie własne*) Przykład operacji erozji.

(a) Przykładowy obraz.

(b) Obraz powstały w wyniku operacji erozji na obrazie przykładowym elementem strukturalnym o rozmiarach 3 na 3. Wszystkie obiekty wyraźnie zmniejszyły swoją powierzchnię, niektóre całkiem zniknęły.

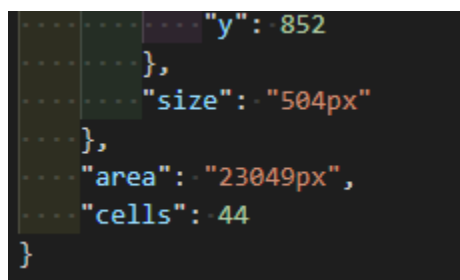
5.7 Przykład Etykietowania



Rysunek 20: (*Opracowanie własne*) Przykładowy obraz.

Przykładowy obraz zawiera 44 elementy, co zostało policzone ręcznie. Za pomocą matematycznych funkcji na pole prostokąta przybliżona suma pól

powierzchni obiektów została wyestymowana na 26000 pikseli.

A screenshot of a code editor showing a JSON object. The text is as follows:

```
{
  "y": 852
},
{
  "size": "504px"
},
{
  "area": "23049px",
  "cells": 44
}
```

Rysunek 21: *(Opracowanie własne)* Zapis z pliku “results.json”.

Zapis z pliku “results.json” wskazuje dużo dokładniejszą sumę pól powierzchni wszystkich obiektów. Błąd wynosi mniej niż 12%, przez co obydwa wyniki dałyby miarodajną ocenę adhezji składników komórkowych. Z prostych przyczyn lepiej jednak skorzystać z dokładniejszego wyniku. Aby obliczyć adhezję potrzebujemy jeszcze ilości wszystkich pikseli na obrazie. Wszystkie obrazy wynikowe (powstałe w wyniku podzielenia obrazu głównego) mają rozmiar 1024×1024 pikseli, stąd ich suma wynosi 1048576 pikseli. Ocena adhezji: $23049/1048576 = 0,022$. Porównanie wyników z kolejnych obrazów ukazuje zmianę adhezji. Ocena adhezji to jasna informacja na temat tego jak wiele komórek znalazło się na obrazie.

6 Wnioski

Opracowana biblioteka pozwala w łatwy sposób tworzyć spersonalizowane procesy przetwarzania obrazów cyfrowych. Algorytmy działają szybko i poprawnie. Istniejące mechanizmy wewnątrz biblioteki pozwalają na podgląd postępu zachodzących procesów. Dzięki możliwości personalizacji procesu przetwarzania, poszczególne obrazy mogą mieć swój własny “przepis”. Możemy w łatwy sposób pozbyć się szumu, zmienić próg względem, którego przebiega proces binaryzacji czy też dostosować jasność obrazu.

Wyniki zapisywane są starannie do plików, każdy przetwarzany obraz ma swój własny plik z wynikami. Dzięki temu możemy w prosty sposób odczytać informacje na temat tego, z jakich pikseli składa się dana komórka. Każdy piksel opisany został współrzędnymi x oraz y . Przez co łatwo możemy zlokalizować go na obrazie wynikowym. Każdy plik z wynikami ma na końcu informacje o sumie pól powierzchni wszystkich komórek na obrazie, a także ile takich komórek zostało znalezionych.

Wybór językia C++ okazał się być mieczem obosiecznym. Dzięki niemu mamy bardzo sprawnie działające algorytmy, a przez niego kod źródłowy rozrósł się tak bardzo, że został dołączony jako załącznik do tej pracy.

Opracowana biblioteka testowana była w warunkach rzeczywistych tzn. takich, w których przyjdzie jej pracować. Pozwoliło to na poprawienie jej działania i usprawnienia niektórych funkcji i klas. Dzięki temu wiemy, że wyzwania, które na nią czekają nie będą większe, niż to z czym do tej pory przyszło jej się zmagać.

Stworzona aplikacja okienkowa pozwala na wykorzystanie części możliwości jakie niesie ze sobą biblioteka “BIA.Core”. Dobrym pomysłem na przyszłość byłoby rozwijanie tej aplikacji, aby mogła wykorzystać wszystkie funkcjonalności jakie wspomniana wcześniej biblioteka posiada. Jednak przenoszenie funkcjonalności z języki C++ do C# za pomocą C++/CLI jest bardzo czasochłonne.

7 Podsumowanie

Celem niniejszej pracy było stworzenie biblioteki zajmującej się analizą i przetwarzaniem obrazów cyfrowych. Biblioteka miała za zadanie ocenę adhezji komórek krwi, co robi w stopniu zadawalającym wykorzystując szybkie algorytmy oraz proces personalizacji przetwarzania obrazów. Dzięki zautomatyzowaniu procesów, które do tej pory były wykonywane ręcznie spełnia również drugi cel - usprawnienie pracy ludzi. Utrzymywany kontakt z osobami, które będą z niej korzystały pozwoliło na stworzenie jej w wygodny do użycia sposób.

Tworząc bibliotekę “BIA.Core” nie brano pod uwagę jej zastosowania na szerszą skalę. Została napisana w jednym konkretnym celu i do jednego konkretnego zadania. Rozwijanie jej jednak mogłoby okazać się dobrym pomysłem, gdyż część biblioteki napisana jest w sposób pozwalający na wykorzystanie jej w ogólnych przypadkach.

Czasochłonność tworzenia oprogramowania, które za pomocą C++/CLI łączy funkcjonalności C++ oraz C# jest trudna do wyestymowania. Nauka jaka z tego płynie na pewno zostanie ze mną na zawsze. Architektura oprogramowania jaką zastosowano w tym rozwiązaniu nie jest zbyt optymalna pod względem rozwoju i utrzymania kodu.

8 Bibliografia

Literatura

- [1] Microsoft Docs *MSBuild* <https://docs.microsoft.com/en-us/visualstudio/msbuild/msbuild?view=vs-2019>
Stan na dzień 05/12/2020
- [2] Alberts B, Johnson A, Lewis J *Molecular Biology of the Cell. 4th edition* New York, Garland Science, 2002
- [3] *Wikipedia* <https://en.wikipedia.org/wiki/Adhesion#:~:text=Adhesion%20is%20the%20tendency%20of,be%20divided%20into%20several%20types>
Stan na dzień 30/12/2020
- [4] James D. Murray, William VanRyper *Encyclopedia of Graphics File Formats: The Complete Reference* O'Reilly Media, Second Edition, 1996, strony 880-908
- [5] Rafael C. Gonzalez, Richard E. Woods *Digital Image Processing Second Edition* Prentice Hall, Upper Saddle River, New Jersey 2002, strony 81, 523-532
- [6] *Cyforwe przetwarzanie obrazów binarnych* https://pl.wikipedia.org/wiki/Cyfrowe_przetwarzanie_obraz%C3%B3w_binarnych
Stan na dzień 30/12/2020
- [7] *LibTIFF - Tutorial* https://research.cs.wisc.edu/graphics/Courses/638-f1999/libtiff_tutorial.html
Stan na dzień 09/01/2021
- [8] *Program GIMP* <https://www.gimp.org/>
Stan na dzień 10/01/2021
- [9] *Program CMake* <https://www.cmake.org/>
Stan na dzień 13/01/2021
- [10] *Biblioteka LibTIFF* <http://www.libtiff.org/>
Stan na dzień 30/12/2020
- [11] *Program Visual Studio 2019* <https://visualstudio.microsoft.com/vs/>
Stan na dzień 10/01/2021
- [12] *L^AT_EX* <https://www.latex-project.org/>
Stan na dzień 10/01/2021
- [13] *Program Visual Studio Code* <https://code.visualstudio.com/>
Stan na dzień 10/01/2021

- [14] *What is .NET Framework?* <https://dotnet.microsoft.com/learn/dotnet/what-is-dotnet-framework>
Stan na dzień 09/01/2021
- [15] *nlohmann.json 3.9.1* <https://github.com/nlohmann/json>
Stan na dzień 30/01/2021
- [16] *OpenMP 5.1* <https://www.openmp.org/>
Stan na dzień 14/02/2021
- [17] *LibTIFF - Building the Software Distribution* <https://libtiff.gitlab.io/libtiff/build.html>
Stan na dzień 09/01/2021

Spis rysunków

1	(<i>Opracowanie własne</i>) Element Strukturalny. (a) Przykład elementu strukturalnego o rozmiarze 3×3 z punktem centralnym w środku. Wszystkie piksele elementu strukturalnego posiadają wartość równą 1. (b) Przykład elementu strukturalnego o rozmiarze 3×3 z punktem centralnym w środku. Lewy dolny, lewy górny, prawy dolny oraz prawy górny piksel posiada wartość 0. (c) Przykład elementu strukturalnego o rozmiarze 3×3 z punktem centralnym w lewym dolnym rogu. Lewy górny, prawy górny oraz prawy dolny piksel posiada wartość 0. (d) Przykład elementu strukturalnego o rozmiarze 3×3 z punktem centralnym w środku. Lewy środkowy, prawy środkowy, dolny środkowy oraz górny środkowy piksel posiada wartość 0. (e) Przykład elementu strukturalnego o rozmiarze 3×3 z punktem centralnym w dolnym środkowym pikselu. Lewy dolny, lewy środkowy, środkowy oraz prawy dolny piksel mają wartość równą 0.	9
2	(<i>Opracowanie własne</i>) (a) Sąsiedztwo Moore'a. (b) Sąsiedztwo Von Neumann'a.	9
3	(<i>Opracowanie własne</i>) Erozja. (a) Element strukturalny użyty w operacji. (b) Stan przed erozją z wykorzystaniem elementu strukturalnego z podpunktu (a). (c) Stan po erozji z wykorzystaniem elementu strukturalnego z podpunktu (a). Granatowym kolorem zaznaczono punkt centralny elementu strukturalnego, natomiast czerwonym piksele, które zmieniły wartości z 1 na 0.	10
4	(<i>Opracowanie własne</i>) Dylatacja. (a) Element strukturalny użyty w operacji. (b) Stan przed dylatacją z wykorzystaniem elementu strukturalnego z podpunktu (a). (c) Stan po dylatacji z wykorzystaniem elementu strukturalnego z podpunktu (a). Granatowym kolorem zaznaczono punkt centralny elementu strukturalnego, natomiast zielonym piksele, które zmieniły wartości z 0 na 1.	10
5	(<i>Opracowanie własne</i>) Otwarcie Morfologiczne. (a) Stan początkowy. (b) Stan po erozji z wykorzystaniem elementu strukturalnego z rysunku 1(a). (c) Stan po dylatacji z wykorzystaniem elementu strukturalnego z rysunku 1(a). Szarym kolorem zaznaczono piksele, które posiadają wartość 1, czerwonym te, które zmieniły wartość z 1 na 0. Zielonym natomiast zaznaczone te piksele, które zmieniły wartość z 0 na 1.	12
6	(<i>Opracowanie własne</i>) Zamknięcie Morfologiczne. (a) Stan początkowy. (b) Stan po dylatacji z wykorzystaniem elementu strukturalnego z rysunku 1(a). (c) Stan po erozji z wykorzystaniem elementu strukturalnego z rysunku 1(a). Szarym kolorem zaznaczono piksele, które posiadają wartość 1, czerwonym te, które zmieniły wartość z 1 na 0. Zielonym natomiast zaznaczone te piksele, które zmieniły wartość z 0 na 1.	13
7	(<i>Opracowanie własne</i>) Korekcja Gamma. (a) Stan początkowy. (b) Stan po korekcji gamma z parametrem <i>0.85</i> . (c) Stan po korekcji gamma z parametrem <i>1.10</i>	14
8	(<i>Opracowanie własne</i>) Diagramy klas biblioteki BIA.Core. (a) Główny diagram klas. Przedstawione na nim klasy odpowiadają za najważniejsze funkcje biblioteki. (b) Diagram klas przedstawiający operacje, których wykonanie na obrazach umożliwia biblioteka.	17

9	(<i>Opracowanie własne</i>) Przykłady plików. (a) Zbinaryzowany obraz wynikowy. (b) Przykład pliku “recipe.json”. Tablica “OPERATIONS” zawiera listę operacji, które mają zostać wykonane na danym obrazie. W tym przypadku zostaną wykonane dwie operacje: Otwarcie morfologiczne (rozdział 3.8) elementem strukturalnym o rozmiarach 3 na 3. Drugą operacją jest etykietowanie wykorzystując sąsiedztwo Vonneumann’a (rozdział 3.5). Zapisana w polu “THRESHOLD” wartość stanowi argument operacji binaryzacji. (c) Przykład pliku “results.json”. Na rysunku przedstawiony jest obiekt, któremu nadano etykietę “265”. Piksele, z których dany obiekt się składa mają odpowiednio indeksy “698367”, “699390”, “699391”, “700414”, “700415”. Każdy z nich posiada współrzędne zapisane w polach “x” oraz “y”. Pole “size” zawiera informacje o tym jaki rozmiar ma dany obiekt. (d) Przykład zakończenia pliku “results.json”. Pole “area” zawiera sumę powierzchni obiektów znalezionych na obrazie, Natomiast pole “cells” to liczba znalezionych obiektów (obiekty o etykietach “96”, “97”, “98”, “99” mogą sugerować, że obiektów jest tylko 99 pomimo wartości “cells” 772. Należy jednak pamiętać o tym, że obiekty są zapisywane do pliku w kolejności losowej).	21
10	(<i>Opracowanie własne</i>) Widok aplikacji okienkowej tuż po jej uruchomieniu.	22
11	(<i>Opracowanie własne</i>) Widok aplikacji okienkowej po kliknięciu przycisku “Initialize” oraz wybraniu eksperymentu “B372_06 5 minut” oraz kliknięciu prawym przyciskiem myszy na jeden z eksperymentów częściowych.	22
12	(<i>Opracowanie własne</i>) Widok aplikacji okienkowej po kliknięciu przycisku “Prepare Process”. Poniżej napisu “Prepare Process” można obserwować postęp tego procesu.	23
13	(<i>Opracowanie własne</i>) Widok aplikacji okienkowej po kliknięciu przycisku “Start Operations”. Poniżej napisu “Perform Operations” można obserwować postęp tego procesu.	23
14	(<i>Opracowanie własne</i>) Przykłady przygotowania katalogu głównego. (a) Folder “vertical” po przeprowadzeniu procesu przygotowania katalogu głównego. (b) Folder “horizontal” po przeprowadzeniu procesu przygotowania katalogu głównego. (c) Folder główny przed uruchomieniem procesu przygotowania katalogu głównego.	25
15	(<i>Opracowanie własne</i>) Przykład operacji binaryzacji. (a) Obraz wynikowy (powstały po podzieleniu obrazu głównego) (b) Obraz powstały w wyniku procesu binaryzacji obrazu wynikowego. Parametr binaryzacji odczytywany jest z pliku “recipe.json”, który zapisany jest w sposób jaki pokazano na rysunku 9(b).	26
16	(<i>Opracowanie własne</i>) Przykład operacji otwarcia. (a) Przykładowy obraz zawierający jeden duży obiekt oraz szum. (b) Obraz powstały w wyniku operacji otwarcia na obrazie przykładowym elementem strukturalnym o rozmiarach 3 na 3. Szum został usunięty.	26
17	(<i>Opracowanie własne</i>) Przykład operacji zamknięcia. (a) Przykładowy obraz. (b) Obraz powstały w wyniku operacji zamknięcia na obrazie przykładowym elementem strukturalnym o rozmiarach 9 na 9. Niektóre obiekty zostały połączone i zwiększyły swoją powierzchnię.	27

18	(<i>Opracowanie własne</i>) Przykład operacji dylatacji. (a) Przykładowy obraz. (b) Obraz powstały w wyniku operacji dylatacji na obrazie przykładowym elementem strukturalnym o rozmiarach 9 na 9. Wszystkie obiekty wyraźnie zwiększyły swoją powierzchnię. .	27
19	(<i>Opracowanie własne</i>) Przykład operacji erozji. (a) Przykładowy obraz. (b) Obraz powstały w wyniku operacji erozji na obrazie przykładowym elementem strukturalnym o rozmiarach 3 na 3. Wszystkie obiekty wyraźnie zmniejszyły swoją powierzchnię, niektóre całkiem zniknęły.	28
20	(<i>Opracowanie własne</i>) Przykładowy obraz.	28
21	(<i>Opracowanie własne</i>) Zapis z pliku “results.json”.	29