НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Автоматизованих систем обробки інформації і управління

«На правах рукопису»
УДК _____

«До захисту допущено»
В.о. Завідувача кафедри
_____ О.А.Павлов
«___»_____20__ р.

Магістерська дисертація

на здобуття ступеня магістра

зі спеціальності 121 Інженерія програмного забезпечення

на тему: «Комплекс програм підтримки діагностики онкологічних захворювань за допомогою гібридних методів обробки зображень»

Виконав (-ла):
студент (-ка) VI курсу, групи ІП-71мн
Сарнацький Владислав Віталійович _____

Керівник:
доц., к.т.н. Баклан І.В. _____

Консультант:
доц., к.т.н. Ліщук К.І. _____

Рецензент:
_____

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент (-ка) _____

Київ – 2019 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Автоматизованих систем обробки інформації і управління

Рівень вищої освіти – другий (магістерський) за освітньо-професійною програмою

Спеціальність – 121 «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ О.А.Павлов

«___»_____20__ р.

ЗАВДАННЯ
на магістерську дисертацію студенту

Сарнацький Владислав Віталійович

1. Тема дисертації «**Комплекс програм підтримки діагностики онкологічних захворювань за допомогою гібридних методів обробки зображень**», науковий керівник дисертації Баклан Ігор Всеволодович, затверджені наказом по університету від «___»_____ 2019 р. №_____

2. Термін подання студентом дисертації_____

3. Об’єкт дослідження: Процес обробки гістологічних зображень з ціллю підвищення їх якості та наочності.

4. Предмет дослідження: Методи та алгоритми, які підвищення якості та наочності гістологічних зображень.

5. Перелік завдань, які потрібно розробити: аналіз існуючих методів обробки гістологічних зображень; виокремлення множини алгоритмів, що активно використовуються у знайдених методах розроблення середи, для компонування алгоритмів у єдиний метод; дослідження ефективності розробленого методу.

6. Орієнтовний перелік графічного (ілюстративного) матеріалу: 3 частини діаграми класів.

7. Орієнтовний перелік публікацій 2 публікації

8. Консультанти розділів дисертації

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Графічний	Баклан І.В., доц.		

9. Дата видачі завдання _____

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1	Аналіз існуючих методів обробки гістологічних зображень. Аналіз літератури.		
2	Виокремлення множини алгоритмів, що активно використовуються у знайдених методах.		
3	Розроблення середи, для компонування алгоритмів у єдиний метод.		
4	Дослідження ефективності розробленого методу.		
5	Розробка технічної документації.		

Студент Сарнацький В.В.

Науковий керівник дисертації Баклан І.В.

РЕФЕРАТ

Актуальність теми.

Онкологічні захворювання є справжньою чумою XXI сторіччя. За даними CDC (Centers for Disease Control and Prevention) кількість нових онкологічних хворих сягне 1.9 млн./рік в 2020 році[1]. В зв'язку с цим, питання розробки та модифікації сучасних методів діагностики та прогнозування раку стає дуже гостро. Одним з таких методів, справжнім «золотим стандартом» є метод гістологічної експертизи, який є дуже складним та багато ступеневим процесом, який виконуються цілком людиною і, нажаль, не позбавлений людського фактору. Так, зразок досліджуваної тканини може бути неякісно взятий у пацієнта, погано нарізаний, надмірно, або недостатньо пофарбований. Все це призводить до того, що результати аналізу експертами одного і того ж зразка можуть сильно різнитися між собою. І для того щоб отримати дійсно точний результат, слід задіяти декількох експертів або один математично обґрунтований алгоритм, який позбавив би цей процес людської суб'єктивності.

Існування таких алгоритмів було би неможливо без розділу комп'ютерних наук, що зветься «комп'ютерний зір». Цей розділ, а на сьогодні можна сказати окрема наука, почав своє існування в 1965 році з кандидатської дисертації, присвяченій темі трьохвимірної реконструкції сцени, що складалася с множини примітивів, таких як, піраміди, конуси, і т.п., по одній їх фотографії [2]. По стану на сьогоднішній день, людство розробило алгоритми, здатні виконувати попіксельну сегментацію зображення, отриманого на встановленій на лобовому склі автомобіля камері, з ціллю виявлення регіонів, що належать дорозі, знакам дорожнього руху, пішоходам, іншим машинам, і т.п [3]. Окрім значної точності, ці алгоритми

являються також досить швидкими, здатними оброблювати зображення в реальному часі.

Зразу після появи комп’ютерного зору, йому зразу знайшли місце і в обробці цифрових медичних зображень. Такі алгоритми як вейвлет- та ширлет-перетворення послугували екстракторами ознак для подальшої класифікації методом опорних векторів гістологічних зображень з ціллю виявлення ступені злоякісності пухлини. Алгоритм водорозділу був успішно адаптований для задачі сегментації окремих клітин. Морфологічний класифікатор був використаний для класифікації знайдених клітин. За допомогою глибоких нейронних мереж вчені змогли провести сегментацію об’єктів на гістологічних зображеннях, які навіть не були пофарбовані.

Зв'язок роботи з науковими програмами, планами, темами.

Дисертаційна робота виконувалась в рамках науково-дослідних робіт кафедри автоматизованих систем обробки інформації та управління факультету обчислювальної техніки та інформатики КПІ імені Ігоря Сікорського № 0117U000914 «Математичні моделі та технології в системах підтримки прийняття рішень».

Мета і задачі дослідження.

Основна *мета* роботи полягає в розробці математичного та алгоритмічного забезпечення для підвищення якості та наочності цифрових гістологічних зображень.

Для досягнення поставленої мети необхідно розв‘язати комплекс наступних взаємопов‘язаних *задач*:

- аналіз існуючих методів обробки гістологічних зображень;
- виокремлення множини алгоритмів, що активно використовуються у знайдених методах;

- розроблення середи, для компонування алгоритмів у єдиний метод;
- дослідження ефективності розробленого методу.

Об’єкт дослідження.

Процес обробки гістологічних зображень з ціллю збільшення їх якості та наочності для людини.

Предмет дослідження.

Методи та алгоритми, які використовуються для обробки та аналізу гістологічних зображень.

Методи дослідження.

При проведенні досліджень і розробок у дисертаційній роботі використовувались методи комп’ютерного зору, обробки зображень, матричні фільтри, алгоритми нормалізації яркості та освітленості, алгоритми сегментації возорозділу, бінаризації с порогом по Оцу, переведення між кольоровими просторами, кластеризація методом К-середніх, для експериментального дослідження – методи об‘єктно-орієнтованого програмування: фреймворк PyQt, бібліотека алгоритмів обробки зображень OpenCV, бібліотека методів машинного навчання sklearn.

Наукова новизна одержаних результатів полягає в наступному:

а) розроблено метод комбінування алгоритмів комп’ютерного зору та їх налаштовування;

б) розроблено метод обробки цифрових гістологічних зображень з ціллю збільшення їх якості та наочності.

Практичне значення одержаних результатів.

Всі запропоновані математичні моделі і алгоритми доведені до практичної реалізації у рамках програмного забезпечення, котре використовується для обробки цифрових гістологічних зображень.

Особистий внесок здобувача.

Усі результати, наведені у дисертації, отримані самостійно.

Апробація результатів дисертації.

Результати дисертації були апробовані на наступних наукових конференціях:

- Науково-практична конференція молодих вчених «Фундаментальна медицина: інтегральні підходи до терапії хворих з онкопатологією» (Київ, 2019 р.).

Публікації.

За результатами дисертації опубліковано 2 наукові праці, в тому числі 1 статті у наукових фахових виданнях, які індексуються у наукометричних базах Copernicus, arXiv:1712.03228v1 [cs.SD] 8 Dec 2017.

Ключові слова.

ГІСТОЛОГІЯ, ОНКОЛОГІЯ, ОБРОБКА ЗОБРАЖЕНЬ, КОМП'ЮТЕРНИЙ ЗІР, АНАЛІЗ ЗОБРАЖЕНЬ, КЛАСТЕРИЗАЦІЯ, СЕГМЕНТАЦІЯ, БІНАРИЗАЦІЯ, ОБРАХУНКОВИЙ ГРАФ.

ABSTRACT

Relevance.

Cancer is a true plague of the 21st Century. According to the CDC (Centers for Disease Control and Prevention), the number of new cancer patients will reach 1.9 million per year in 2020 [1]. In this regard, the issue of developing and modifying modern methods of diagnosis and prediction of cancer becomes very acute. One such method, the true "golden standard" is the method of histological examination, which is a very complex and many-step process, which is performed entirely by man and, unfortunately, not devoid of human factor. Thus, the sample of the investigated tissue may be poorly taken from the patient, badly sliced, excessively, or insufficiently colored. All this leads to the fact that the results of analysis by experts of the same sample can vary greatly among themselves. And in order to get a really accurate result, it is necessary to involve several experts or one mathematically grounded algorithm that would deprive this process of human subjectivity.

The existence of such algorithms would be impossible without the division of computer sciences, called "computer vision". This section, and today one can say a separate science, began its existence in 1965 with a PhD thesis devoted to the topic of three-dimensional reconstruction of a scene consisting of a plurality of primitives such as pyramids, cones, etc., in one of their photographs [2]. As of today, mankind has developed algorithms that can perform the pixelwise segmentation of the image obtained on a camera mounted on the windscreen of the car, with the aim of identifying regions belonging to the road, traffic signs, pedestrians, other cars, etc. [3]. In addition to high accuracy, these algorithms are also fast enough to process images in real time.

Immediately after the appearance of computer vision, it immediately found a place in the processing of digital medical images. Such algorithms as wavelet and

shearlet-transformation were user as feature extractors for further classification by the support vector machines of histological images with the aim of detecting the degree of tumor malignancy. The watershed segmentation algorithm has been successfully adapted for the segmentation of individual cells. The morphological classifier was used to classify the found cells. With the help of deep neural networks, scientists were able to segment objects on histological images that were not even painted.

Connection with other scientific programs, plans and topics.

Dissertation work was developed as part of scientific research works of Computer-Aided Management And Data Processing Systems Department of Faculty of Informatics and Computer Science, Igor Sikorsky Kyiv Polytechnic Institute № 0117U000914 “Mathematical models and technologies in decision making systems”.

Goal and tasks.

The main goal of the work is to develop mathematical and algorithmic support for improving the quality and visibility of digital histological images. To achieve this goal, it is necessary to solve a set of the following interconnected tasks:

- analysis of existing methods of processing histological images; separation of the set of algorithms actively used in the found methods;
- development of the environment for the layout of the algorithms in a single method;
- study of the effectiveness of the developed method.

Research object.

Face recognition from photo for face position detection and 68 facial landmarks position detection.

Research subject.

The process of processing histological images with the aim of increasing their quality and visibility for a person.

Research methods.

During research and development in the dissertation work methods of computer vision, image processing, matrix filters, algorithms for the normalization of brightness and illumination, watershed segmentation algorithms, binaryzation with the threshold of the Otsu, translation between color spaces, clustering by the K-means method were used, for experimental research - methods of object-oriented programming: framework PyQt, library of image processing algorithms OpenCV, library of methods of machine learning sklearn were used.

Scientific novelty is as follows:

- method of combining computer vision algorithms and their configuration is developed;
- method for processing digital histological images has been developed with the aim of increasing their quality and visibility.

Practical use.

All proposed mathematical models and algorithms are brought to practical realization within the framework of the software that is used for the processing of digital histological images.

Publications.

According to this research 2 scientific papers were published. One of them is indexing in the online database Copernicus, arXiv:1712.03228v1 [cs.SD] 8 Dec 2017.

Keywords.

HISTOLOGY, ONCOLOGY, IMAGE PROCESSING, COMPUTER VISION, IMAGE ANALYSIS, CLUSTERING, SEGMENTATION, BINARYIZATION, COMPUTATIONAL GRAPH.

ЗМІСТ

ВСТУП

Актуальність теми.

Онкологічні захворювання є справжньою чумою XXI сторіччя. За даними CDC (Centers for Disease Control and Prevention) кількість нових онкологічних хворих сягне 1.9 млн./рік в 2020 році[1]. В зв'язку с цим, питання розробки та модифікації сучасних методів діагностики та прогнозування раку стає дуже гостро. Одним з таких методів, справжнім «золотим стандартом» є метод гістологічної експертизи, який є дуже складним та багато ступеневим процесом, який виконуються цілком людиною і, нажаль, не позбавлений людського фактору. Так, зразок досліджуваної тканини може бути неякісно взятий у пацієнта, погано нарізаний, надмірно, або недостатньо пофарбований. Все це призводить до того, що результати аналізу експертами одного і того ж зразка можуть сильно різнитися між собою. І для того щоб отримати дійсно точний результат, слід задіяти декількох експертів або один математично обґрунтований алгоритм, який позбавив би цей процес людської суб'єктивності.

Існування таких алгоритмів було би неможливо без розділу комп'ютерних наук, що зветься «комп'ютерний зір». Цей розділ, а на сьогодні можна сказати окрема наука, почав своє існування в 1965 році з кандидатської дисертації, присвяченій темі трьохвимірної реконструкції сцени, що складалася с множини примітивів, таких як, піраміди, конуси, і т.п., по одній їх фотографії [2]. По стану на сьогоднішній день, людство розробило алгоритми, здатні виконувати попіксельну сегментацію зображення, отриманого на встановленій на лобовому склі автомобіля камері, з ціллю виявлення регіонів, що належать дорозі, знакам дорожнього руху, пішоходам, іншим машинам, і т.п [3]. Окрім значної точності, ці алгоритми

являються також досить швидкими, здатними оброблювати зображення в реальному часі.

Зразу після появи комп'ютерного зору, йому зразу знайшли місце і в обробці цифрових медичних зображень. Такі алгоритми як вейвлет- та ширлет-перетворення послугували екстракторами ознак для подальшої класифікації методом опорних векторів гістологічних зображень з ціллю виявлення ступені злоякісності пухлини. Алгоритм водорозділу був успішно адаптований для задачі сегментації окремих клітин. Морфологічний класифікатор був використаний для класифікації знайдених клітин. За допомогою глибоких нейронних мереж вчені змогли провести сегментацію об'єктів на гістологічних зображеннях, які навіть не були пофарбовані.

Зв'язок роботи з науковими програмами, планами, темами.

Дисертаційна робота виконувалась в рамках науково-дослідних робіт кафедри автоматизованих систем обробки інформації та управління факультету обчислювальної техніки та інформатики КПІ імені Ігоря Сікорського № 0117U000914 «Математичні моделі та технології в системах підтримки прийняття рішень».

Мета і задачі дослідження.

Основна *мета* роботи полягає в розробці математичного та алгоритмічного забезпечення для підвищення якості та наочності цифрових гістологічних зображень.

Для досягнення поставленої мети необхідно розв'язати комплекс наступних взаємопов'язаних *задач*:

- аналіз існуючих методів обробки гістологічних зображень;
- виокремлення множини алгоритмів, що активно використовуються у знайдених методах;

- розроблення середи, для компонування алгоритмів у єдиний метод;
- дослідження ефективності розробленого методу.

Об’єкт дослідження.

Процес обробки гістологічних зображень з ціллю збільшення їх якості та наочності для людини.

Предмет дослідження.

Методи та алгоритми, які використовуються для обробки та аналізу гістологічних зображень.

Методи дослідження.

При проведенні досліджень і розробок у дисертаційній роботі використовувались методи комп’ютерного зору, обробки зображень, матричні фільтри, алгоритми нормалізації яркості та освітленості, алгоритми сегментації возорозділу, бінаризації с порогом по Оцу, переведення між кольоровими просторами, кластеризація методом К-середніх, для експериментального дослідження – методи об‘єктно-орієнтованого програмування: фреймворк PyQt, бібліотека алгоритмів обробки зображень OpenCV, бібліотека методів машинного навчання sklearn.

Наукова новизна одержаних результатів полягає в наступному:

а) розроблено метод комбінування алгоритмів комп’ютерного зору та їх налаштовування;

б) розроблено метод обробки цифрових гістологічних зображень з ціллю збільшення їх якості та наочності.

Практичне значення одержаних результатів.

Всі запропоновані математичні моделі і алгоритми доведені до практичної реалізації у рамках програмного забезпечення, котре використовується для обробки цифрових гістологічних зображень.

Особистий внесок здобувача.

Усі результати, наведені у дисертації, отримані самостійно.

Апробація результатів дисертації.

Результати дисертації були апробовані на наступних наукових конференціях:

- Науково-практична конференція молодих вчених «Фундаментальна медицина: інтегральні підходи до терапії хворих з онкопатологією» (Київ, 2019 р.).

Публікації.

За результатами дисертації опубліковано 2 наукові праці, в тому числі 1 статті у наукових фахових виданнях, які індексуються у наукометричних базах Copernicus, arXiv:1712.03228v1 [cs.SD] 8 Dec 2017.

1 ОГЛЯД ЛІТЕРАТУРИ

1.1 Загальні відомості

Основним методом діагностики та прогнозу онкологічних захворювань є метод гістопатологічної експертизи. Згідно з ним, у пацієнта за допомогою біопсії забирається зразок тканини, який після нарізання досліджується експертом за допомогою мікроскопа. Ця процедура є досить складною, що часто приводить к тому, що результат аналізу одного і того ж зразка змінюється від експерта к експерту. В зв’язку з цим, активною областю досліджень є розробка автоматичних методів аналізу гістологічних зображень та його підтримки.

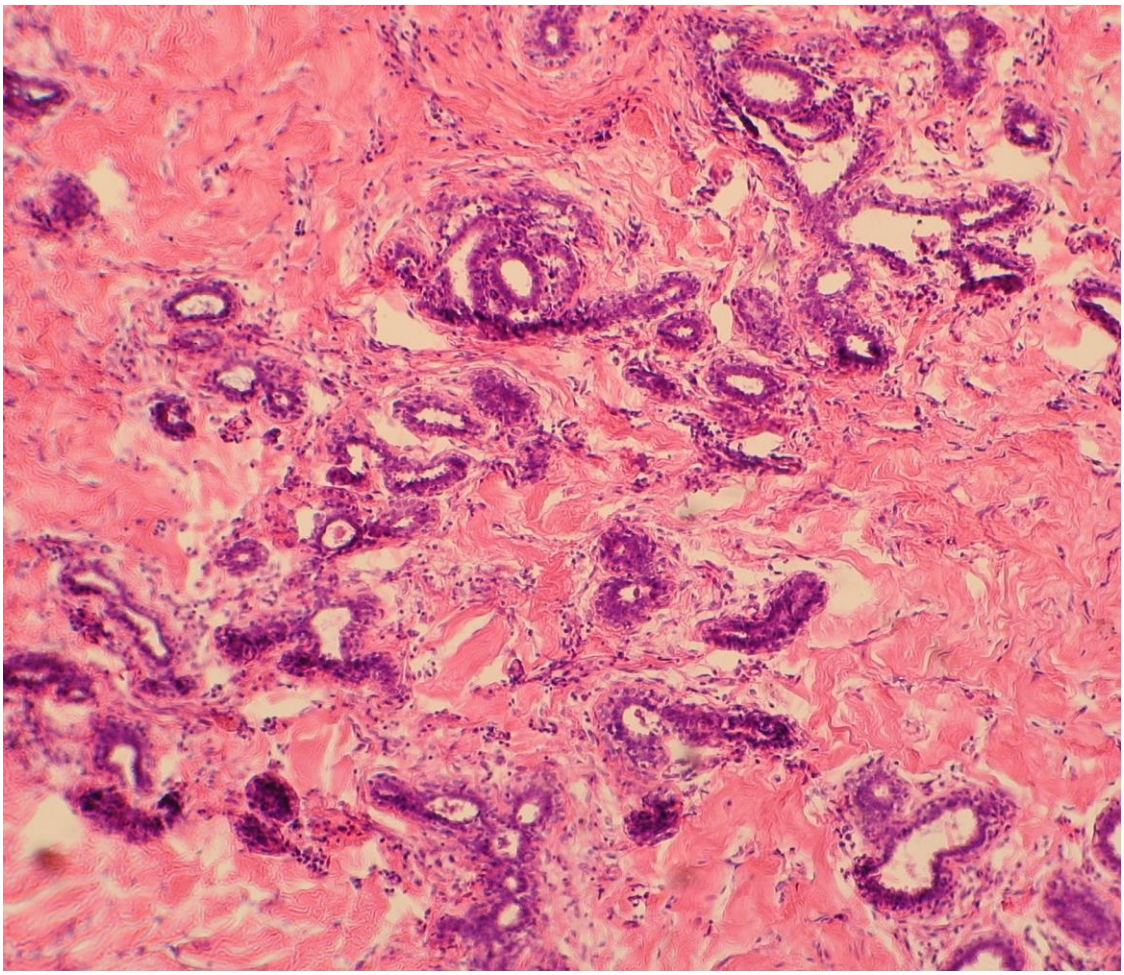


Рисунок 1.1 - Гістологічне зображення тканини

У результаті огляду літератури було виявлено багато різних методів та підходів то поставленої задачі. Так, задача аналізу гістологічних зображень була досліджена зі сторони детектування кровоносних сосудів, наявність яких тісно зв’язана з характером досліджуваного зразка [4].

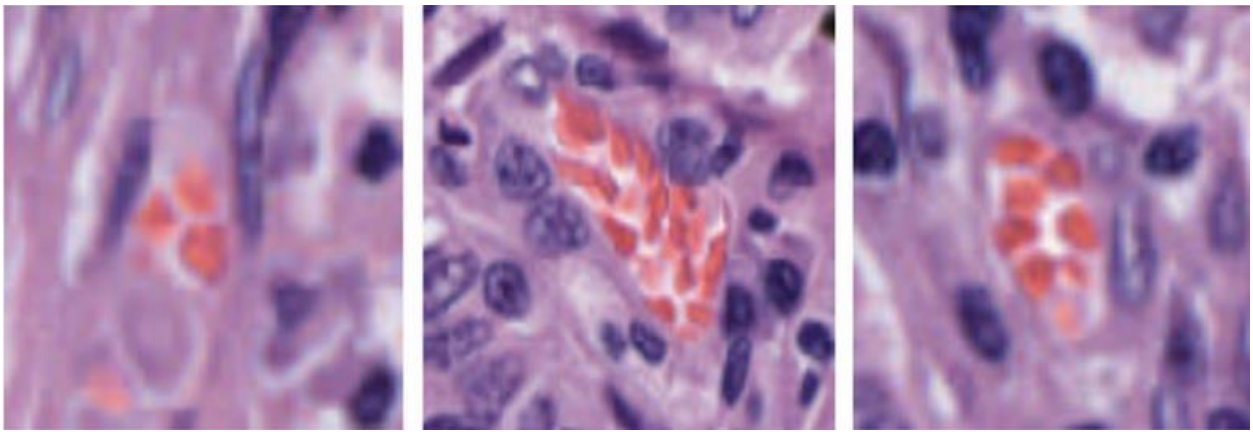


Рисунок 1.2 - Кровоносні сосуди у зразку з маркером H&E [4]

В якості методу детектування був обраний метод попіксельної сегментації за допомогою згорткових нейронних мереж, а саме модифікації архітектури U-Net, що була запропонована для задачі сегментації зображень клітин [5].

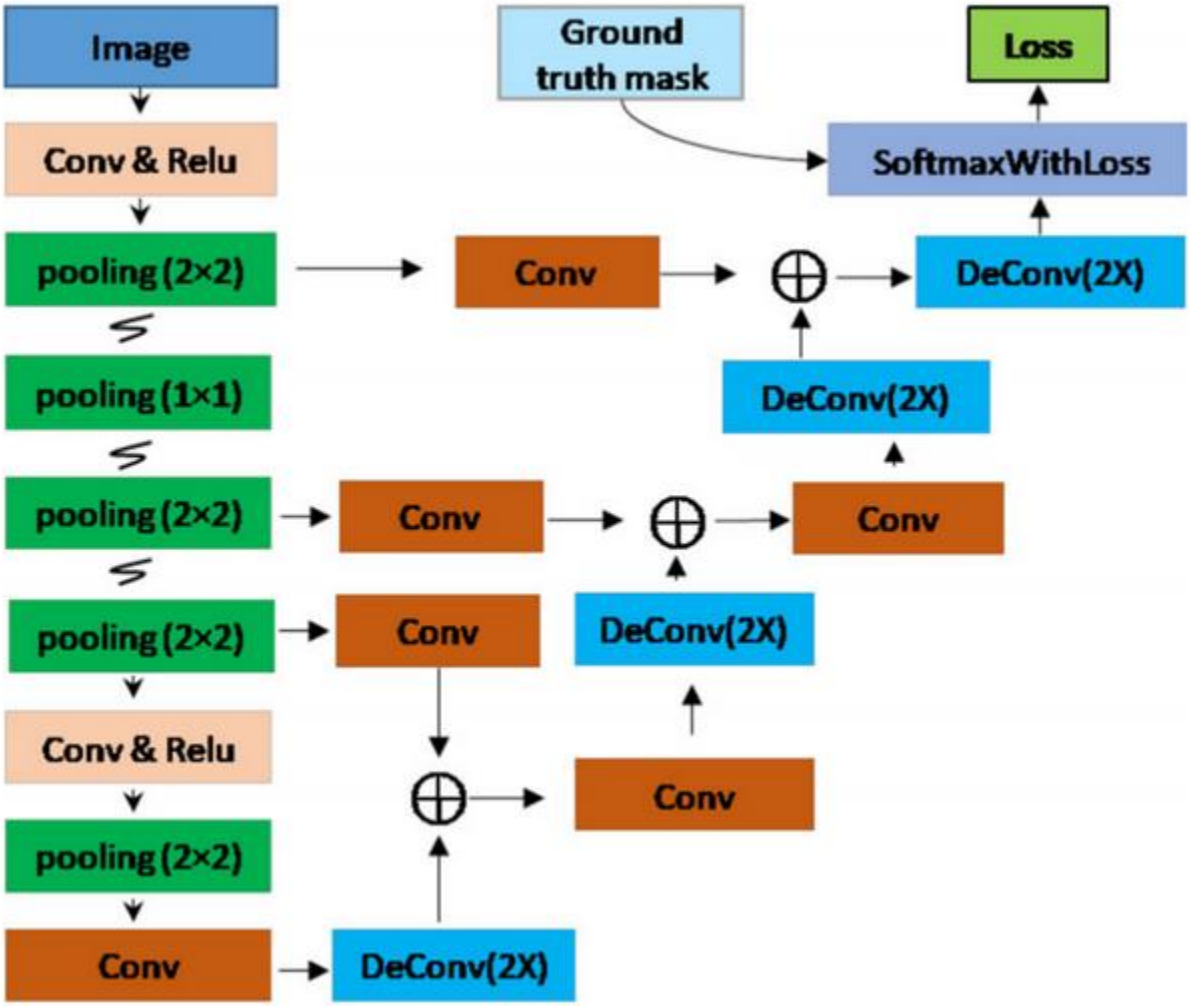


Рисунок 1.3 - Запропонована архітектура нейронной мережі [4]

Для навчання нейронної мережі, авторами була створена база данних зображень та їх сегментаційних карт (Рис 1.4, 1.5). Зображення мали розмір у пікселях 384x384 з роздільною здатністю 500нм/піксель.

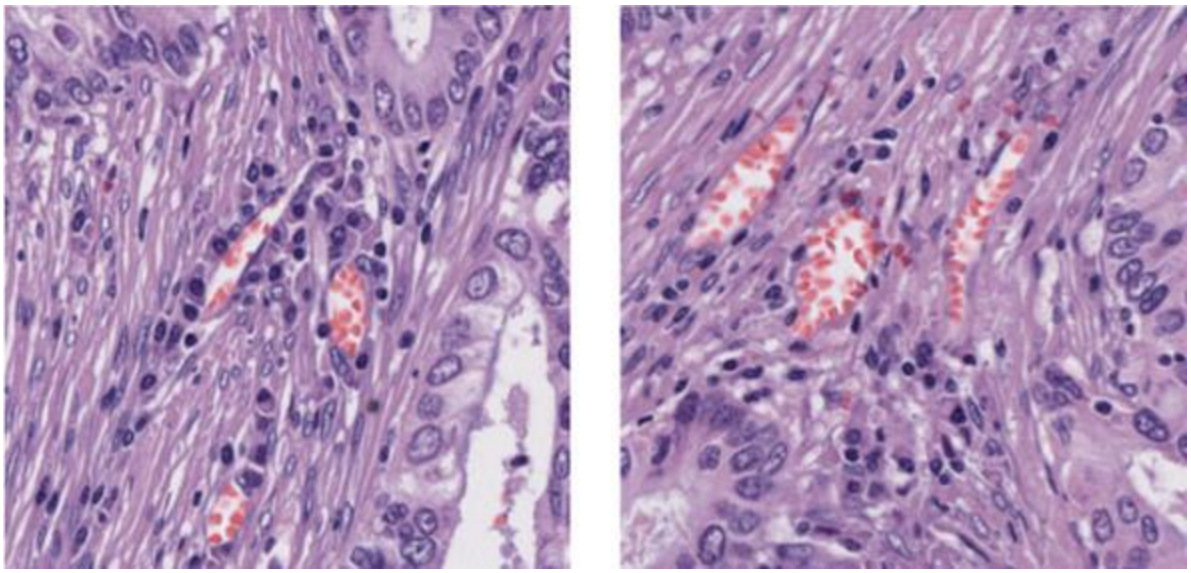


Рисунок 1.4 - Два зображення мікрососудів з навчальної бази зображень [4]

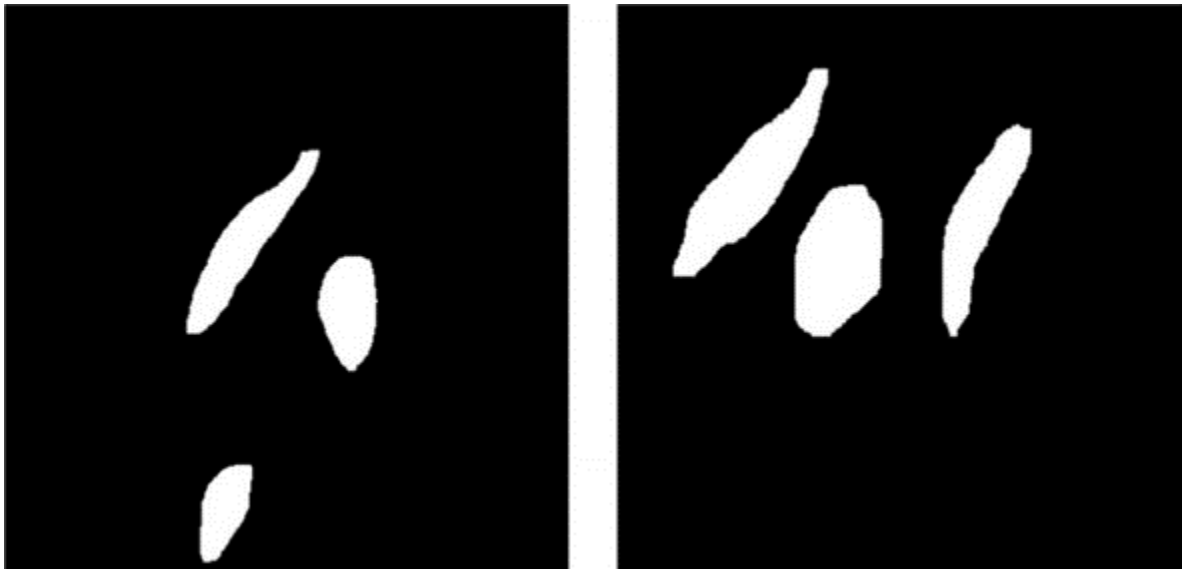


Рисунок 1.5 - Відповідні зображенням з рис 1.4 бінарні сегментаційні карти [4]

Навчання нейронної мережі проводилось за допомогою алгоритму стохастичного градієнтного спуск з моментом. На рисунку 1.6 зображений прогрес цільової функції помилки протягом навчання.

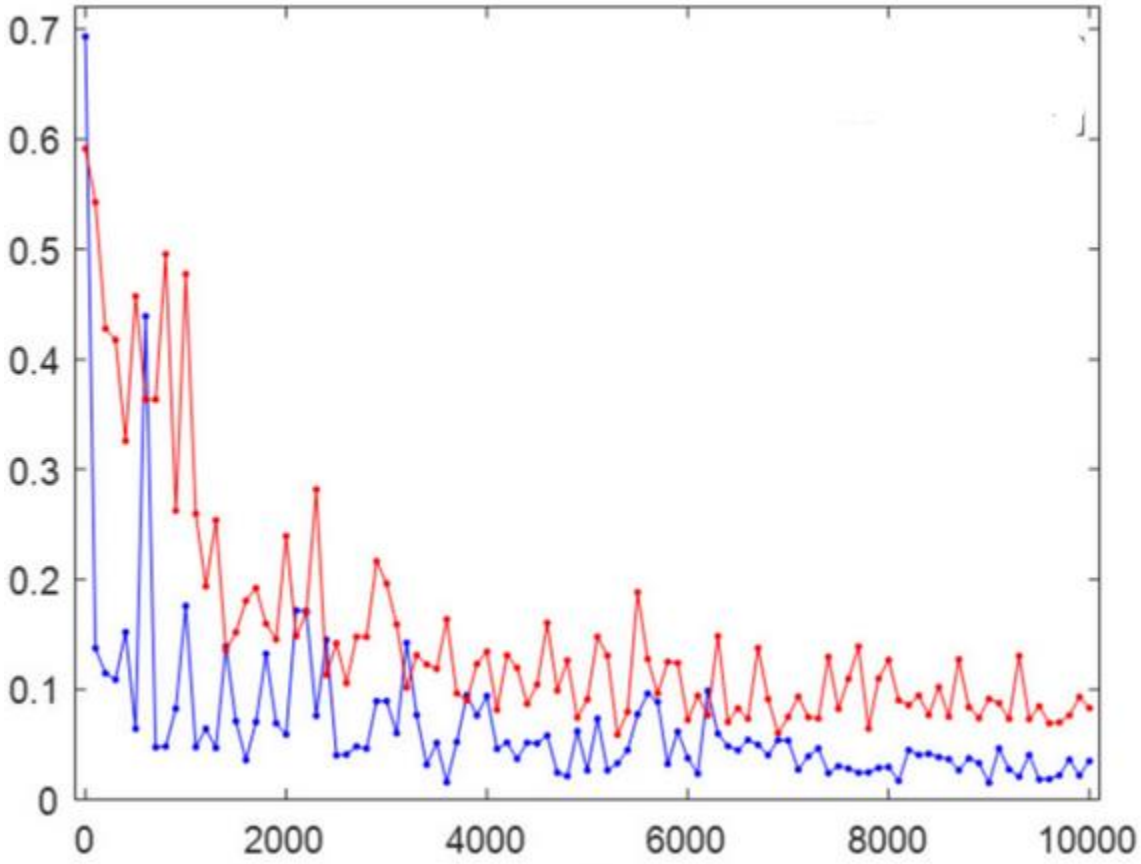


Рисунок 1.6 - Функція помилки на навчальній (синім) та тестовій (червоним) базі зображень [4]. Вісь абсцис – ітерації навчання

В таблиці 1.1 наведені статистичні метрики отриманої моделі, а на рисунках 1.7, 1.8 – візуалізація результату роботи на тестових зображеннях.

Таблиця 1.1 - Результати роботи моделі на тестових данних [4]. (pA – попіксельна точність, mA – середня точність, mIU – середній коефіцієнт Жаккара, FP – кількість помилок першого роду, FN – кількість помилок другого роду)

pA	0.952
mA	0.833
mIU	0.755
FP	119
FN	7

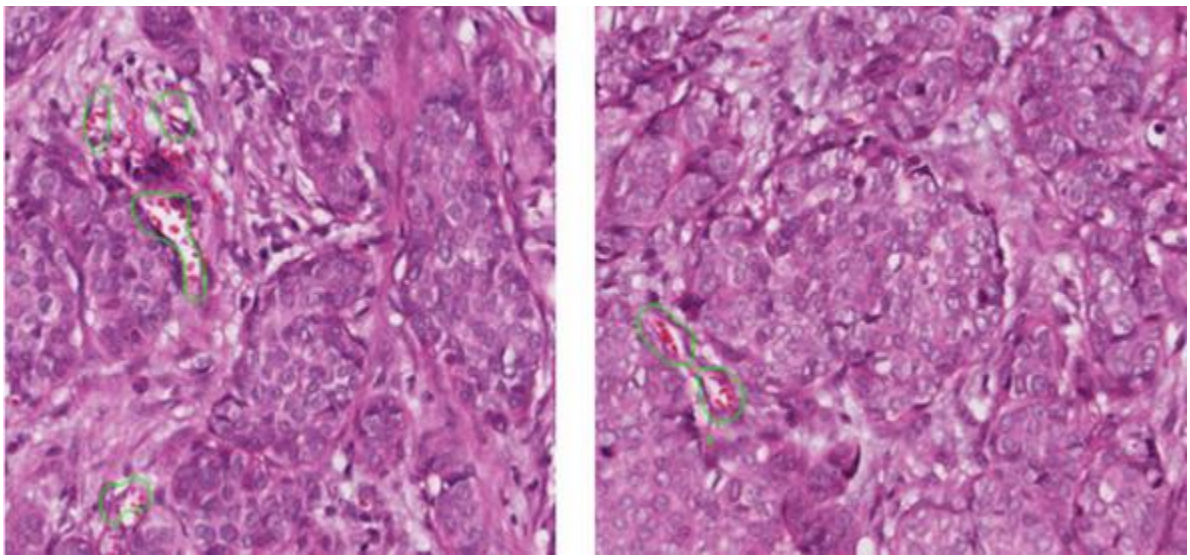


Рисунок 1.7 - Результат передбачення для на пофарбованих H&E зображеннях тканин раку молочної залози [4]

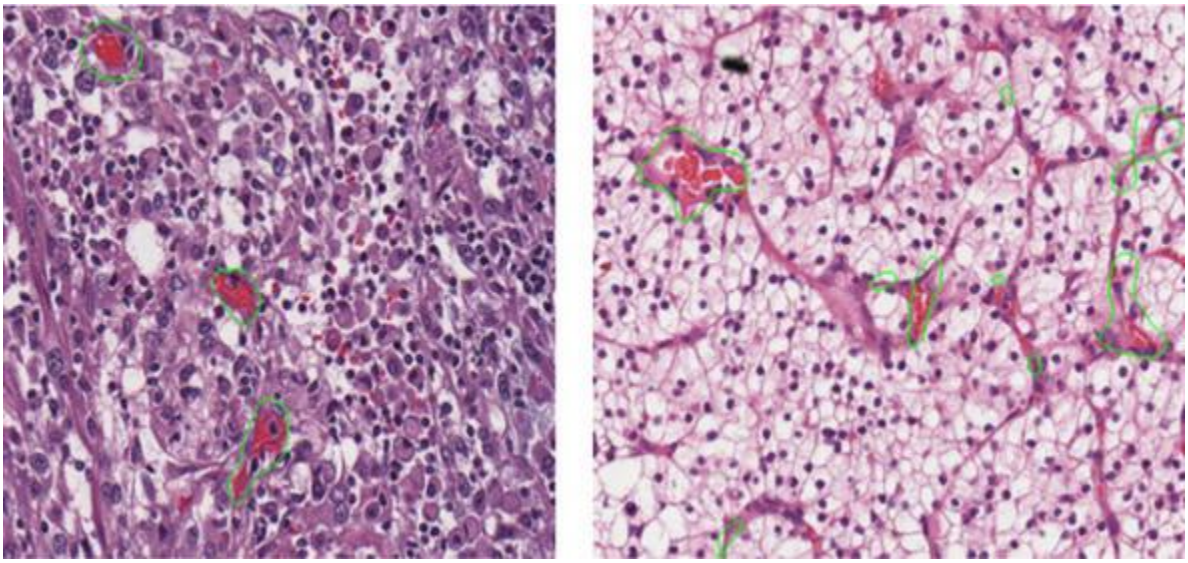


Рисунок 1.8 - Результат передбачення для на пофарбованих H&E зображеннях тканин раку нирок [4]

Окрім ширикого классу алгоритмів, що базуються на нейронних мережах, багато праць присвячені використанню методів так званої «класичної» обробки зображень. Так, для аналізу мамограм використовують методи [6].

- Векторне квантування. Цей механізм використовує сегментацію зображень, яке супроводжується створенням кодової книги та навчального вектору, ділячи зображення на сегменти одного розміру. Алгоритм LindeBuzo and Grey (LBG) для категоризації векторного квантування перетворює повне зображення на навчальні вектори. За нього, визначається центроїд і обчислюється евклідова відстань між ним та навчальними векторами. Техніка робить можливим реконструювати образ для подальшого дослідження.
- Метод К-середніх та нечітка логіка у режимі навчання з та без вчителя. Ці методи використовуються для маркування та кластеризації обокремленних регіонів зображення.

- Бінаризація та вейвлет-перетворення. Оскільки всі несинтетичні зображення містять нечіткості та шуми, що обумовлені процесом їх отримання, подальший їх аналіз з ціллю виявлення невеликих аномалій є складнм. Ці методи використовуються для детектування шуму та для надання чіткості цифровим зображенням.
- Сегментація алгоритмом водорозділу. Алгоритм базується на аналізі зв’язних компонентів та Фур’є-перетворенні. У початковій фазі сегментація здійснюється через морфологічні математичні операції. Далі, фаза вилучення ознак кольору та форми здійснюються за допомогою дескрипторів Фур'є-перетворення. Результатом алгоритму є сегментаційна карта.
- Метод опорних векторів. Цей метод використовується у режимі навчання з вчителем и здійснює классифікацію ознак за наданими класами.

Недавня поява широко-слайдових цифрових сканерів дозволила розробити кількісні підходи до аналізу гістоморфометрії, які тепер можуть дозволити детальну просторову експертизу (наприклад, захоплення орієнтації ядер, текстури, форми, архітектури) всього пухлинного морфологічного ландшафту та його найбільш інвазивні елементи зі стандартного, забарвленого гематоксиліном та еозином (H&E) слайду. Проте передумовою для визначення цих характеристик є необхідність виявлення і сегментації гістологічних примітивів (наприклад, ядер, залоз). Иршад та ін. [7] склав велику колекцію документів в області ядерної сегментації для гістологічного аналізу за останні 20 років. Фатакдавала та ін. [8] і Глотсос та ін. [9] використовували активну контурну модель для сегментації окремих ядер на мікроскопічних зображеннях, пофарбованих H&E. Ксу та ін. [10] застосували ідею активних контурів для сегментації окремих залоз на зображеннях гістопатології раку простати. Сиринукунваттана та ін. [11] ввів

нову стохастичний модель залозистих структур в гістологічних зображеннях. Підхід розглядає кожну залозну структуру на зображенні як багатокутник з випадковим числом вершин, де вершини являють собою приблизні розташування ядер епітелію. Підхід був успішним, и його використовують для виявлення та вилучення залозистих структур по гістологічним зображенням нормальної тканини товстої кишки (Сиринукунваттана та ін. [12]). Результати, що стосуються підходу, показані на рисунку 1.9.

У цьому розділі були знайдені наукові праці в області цифрових рішень щодо аналізу гістологічних зображень. Знайдена література була опрацiована на предмет виявлення спільних методів та алгоритмів обробки зображень.

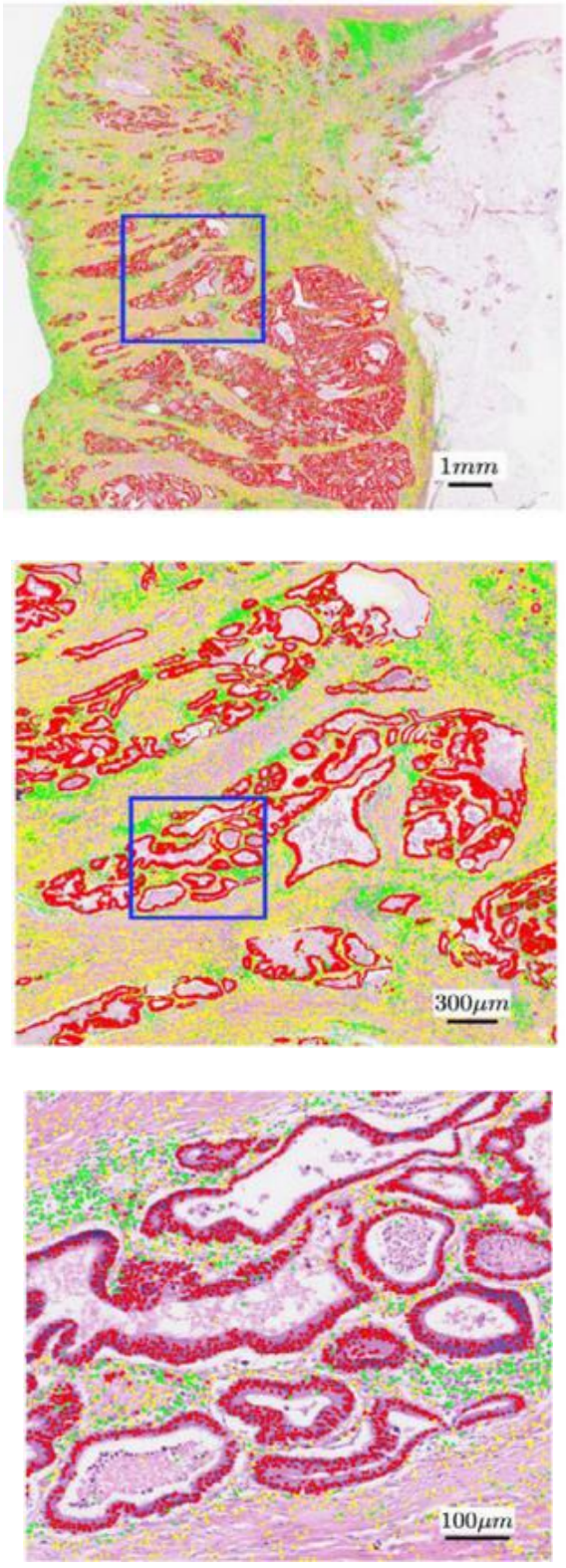


Рисунок 1.9 - Результати детектування ядер за допомогою локально обмеженних методів глибокого навчання [13]

2 ОПИС ЗАПРОПОНОВАНОГО РІШЕННЯ

2.1 Загальний опис

У результаті аналізу літератури була виявлена множина алгоритмів, пре-, постпроцесингу а також аналізу цифрових зображень. Було вирішено спроектувати програмний комплекс, при роботі з яким, користувач мав би можливість самостійно обирати, налаштовувати етапи обробки зображення та об’єднувати їх в обрахунковий граф. Кожен алгоритм може бути розглянутий як множина входів, множина внутрішніх параметрів та множина виходів. На рисунку 2.1 зображений приклад вузла, що реалізує один з таких алгоритмів. Можна побачити, що множина входів складається з одного елемента – вхідного зображення; множина виходів також має один елемент – вихідне зображення, а множина внутрішніх параметрів включає в себе одне число – розмір фільтру Гауса [14].

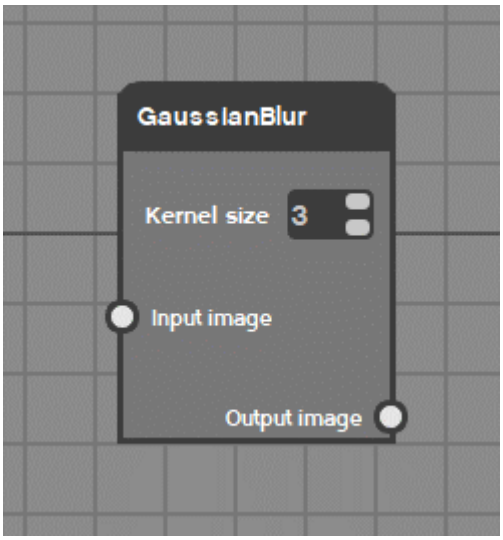


Рисунок 2.1 - Приклад обрахункового вузла, що виконує розмиття по Гаусу

Кожен вхід та вихід обрахункового вузла має асоційований з ним тип даних, а самі типи даних організовані у деревовидну структуру (рис. 2.2).

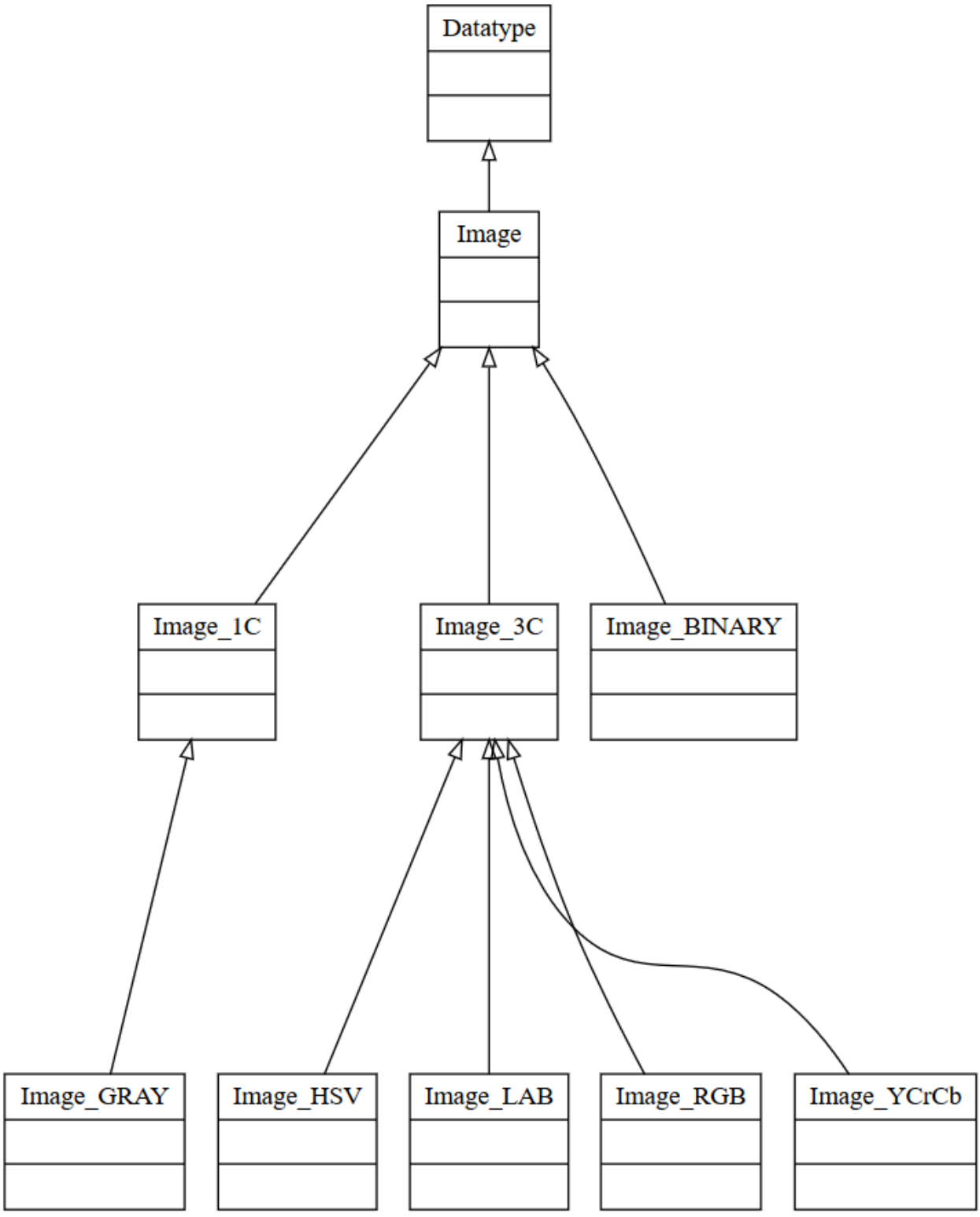


Рисунок 2.2 - Структура типів даних входів та виходів обрахункових вузлів

Наявність цієї ієрархії дозволяє контролювати зв’язки між вузлами. Так, вихідний вузол може бути під’єднаний до вхідного тоді і тільки тоди, коли він має такий сами тип даних, або його тип даних є нащадком (прямим або непрямим) типу даних вхідного вузла (рис 2.3).

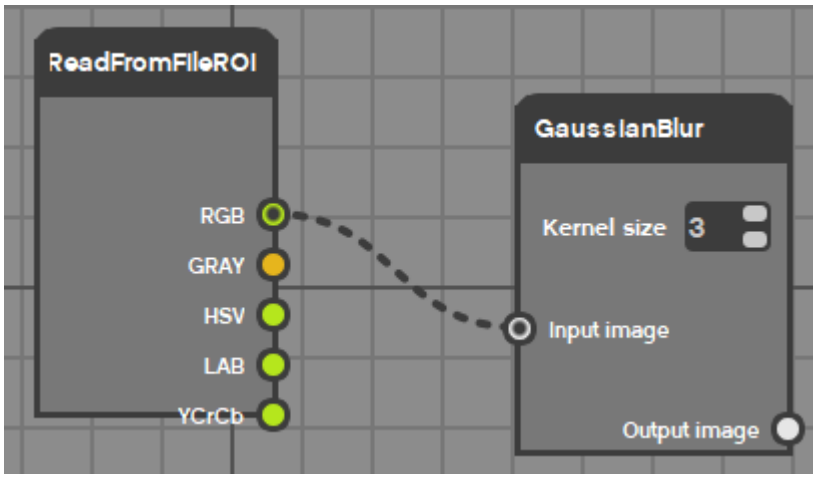


Рисунок 2.3 - Зв’язок між вузлами є можливим, оскільки вихід “RGB” має тип Image_RGB, що є нащадком типу входу “Input image” (Image)

Кожен обрахунковий вузол має обрахункову функцію, аргументами якої є множини величин на входах та внутрішніх параметрів, а виходом – множина значень на виходах. Обрахункова функція може бути виконана тільки тоді, коли усі входи вузла під’єднані до інших вузлів. На рисунку 2.4 зображений приклад готового обрахункового графу.

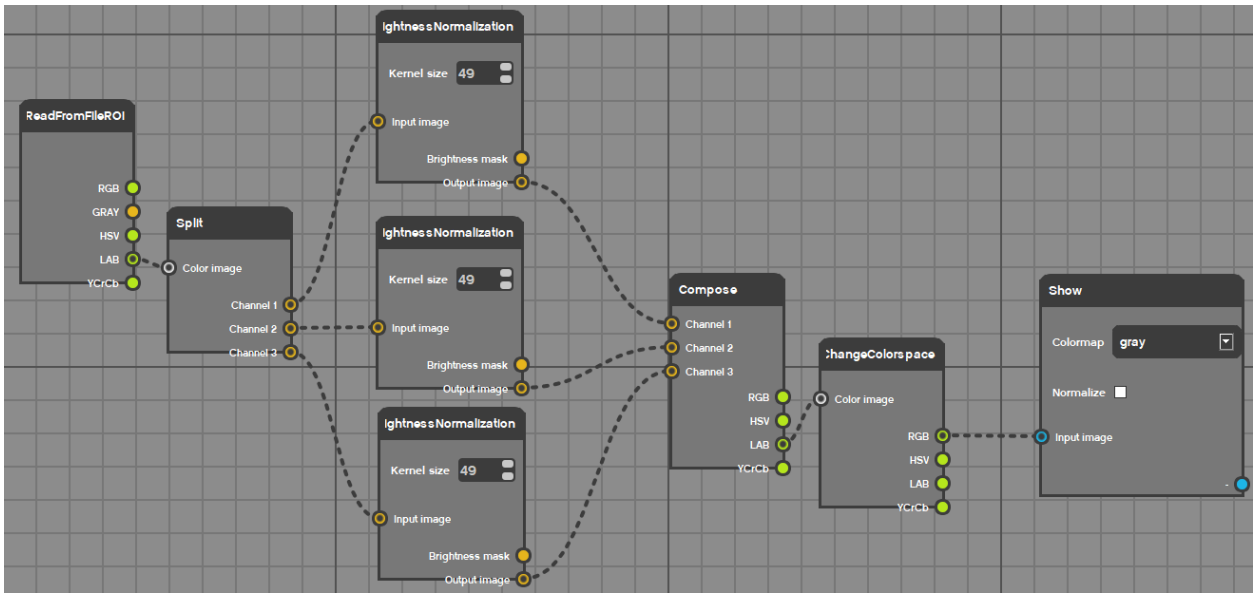


Рисунок 2.4 - Приклад обрахункового графу

Для спрощення аналізу та пришвидшення роботи обрахункового графу, зображення завантажується з файлу, після чого обирається зона зображення, що подається в граф. На рисунку 2.5 зображений цей процес.

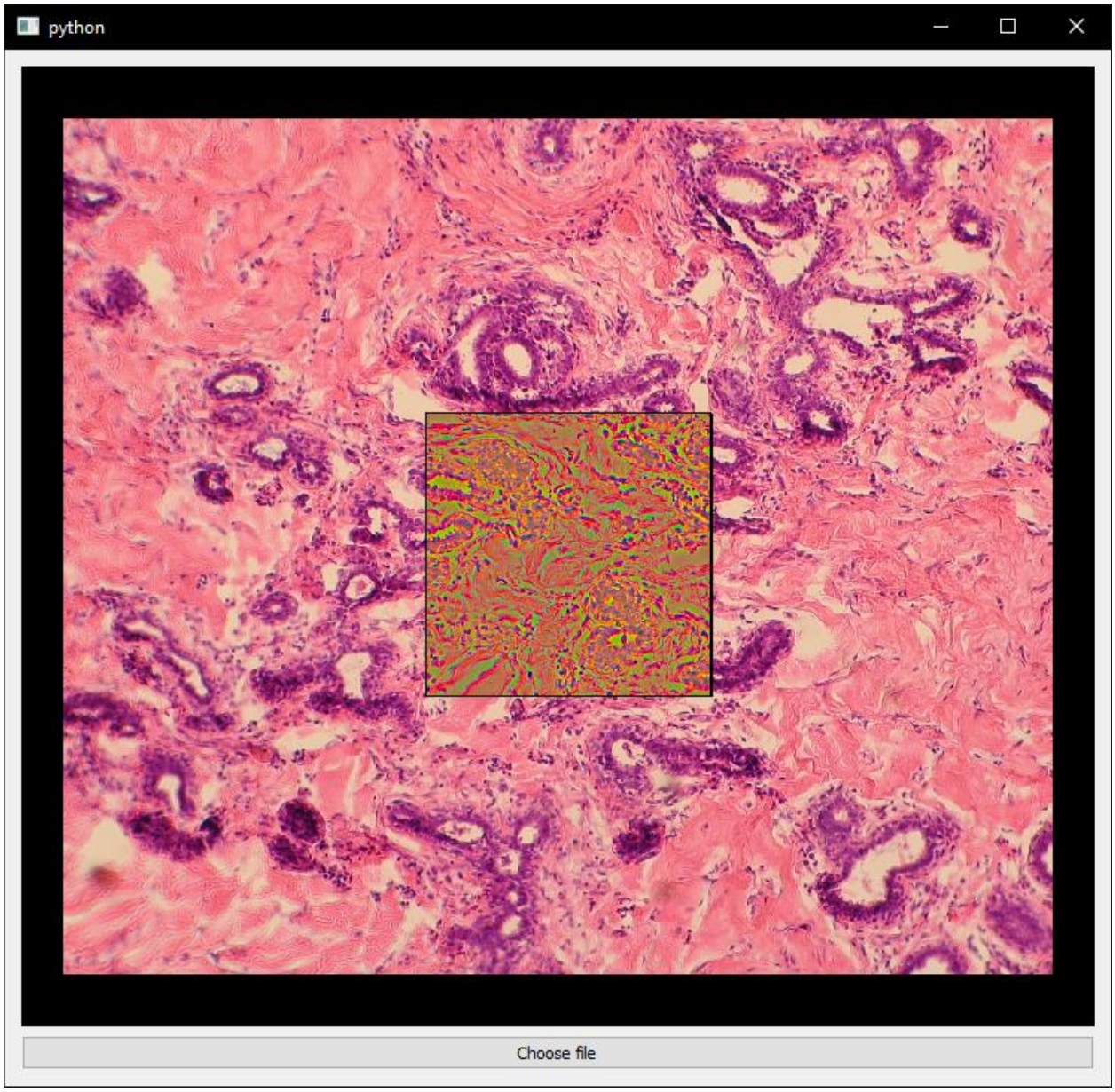


Рисунок 2.5 - Процес вибору аналізованої області. Оригінальне зображення займає усе вікно, аналізована область – простір усередині рамки в центрі вікна. Для простоти подальшого аналізу, результат роботи графу накладається на відповідну область оригінального зображення

2.2 Опис алгоритмів

Тут і надалі буде використана наступна нотація:

- A, B, C, … - тензори 3го рангу з розміром h×w×c, де h – висота зображення, w – ширина зображення, c – кількість каналів зображення (1 або 3);
- : - повний зріз тензору вздовж осі;
- a:b – частковий зріз тензору уздовж осі;
- A[h], A[:, w] – зрізи тензору з відкиданням розмірності;
- A[h1:h2, w1:w2, c1:c2] – комбінований зріз тензору. Результатом операції є новий тензор B, для якого вірно:

- X, X_i – вхідні зображення (тензори);
- Y, Y_i – вихідні зображення (тензори).

Функція, яку містить кожен вузол, є реалізацією певного алгоритму. Серед цих алгоритмів є:

2.2.1 Отримання зображення з обраної зони іншого зображення (ReadFileFromROI)

Якщо (h1, w1, h2, w2) – координати обраного прямокутного регіону, то Y = X[h1:h2, w1:w2]

2.2.2 Альфа-змішування

Якщо коефіцієнт змішування рівен *a*, то

$Y=X_1 * a + X_2 * (1 – a)$.

На рис 2.6-2.8 зображений приклад роботи алгоритму.

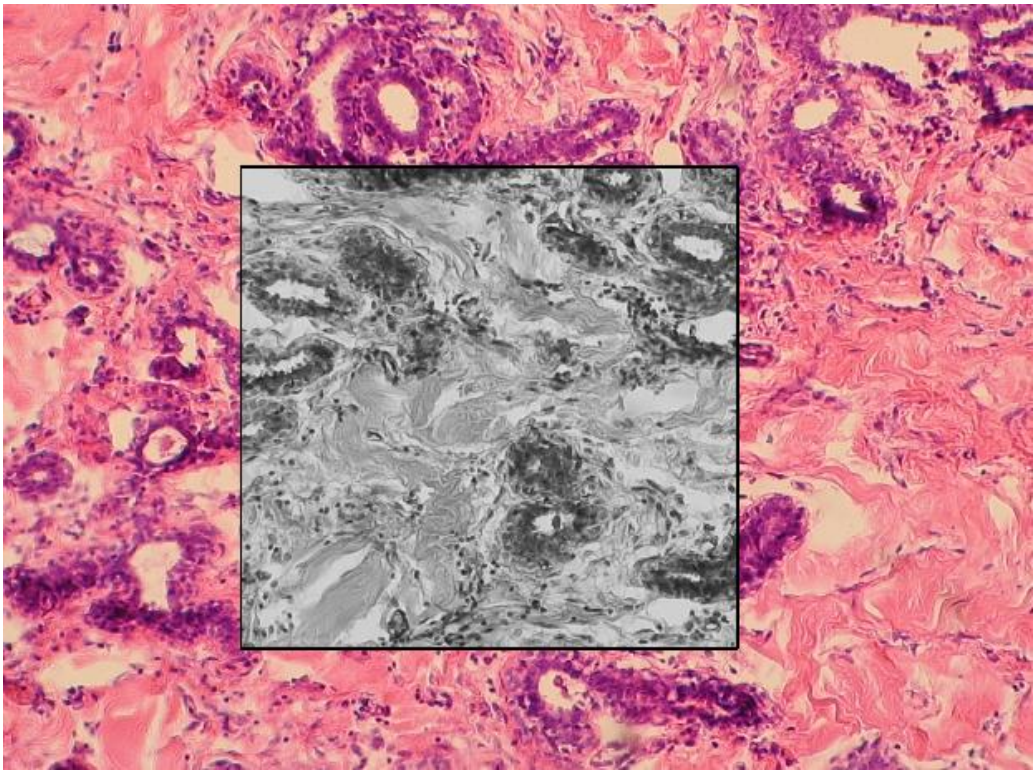


Рисунок 2.6 - Результат при a = 0

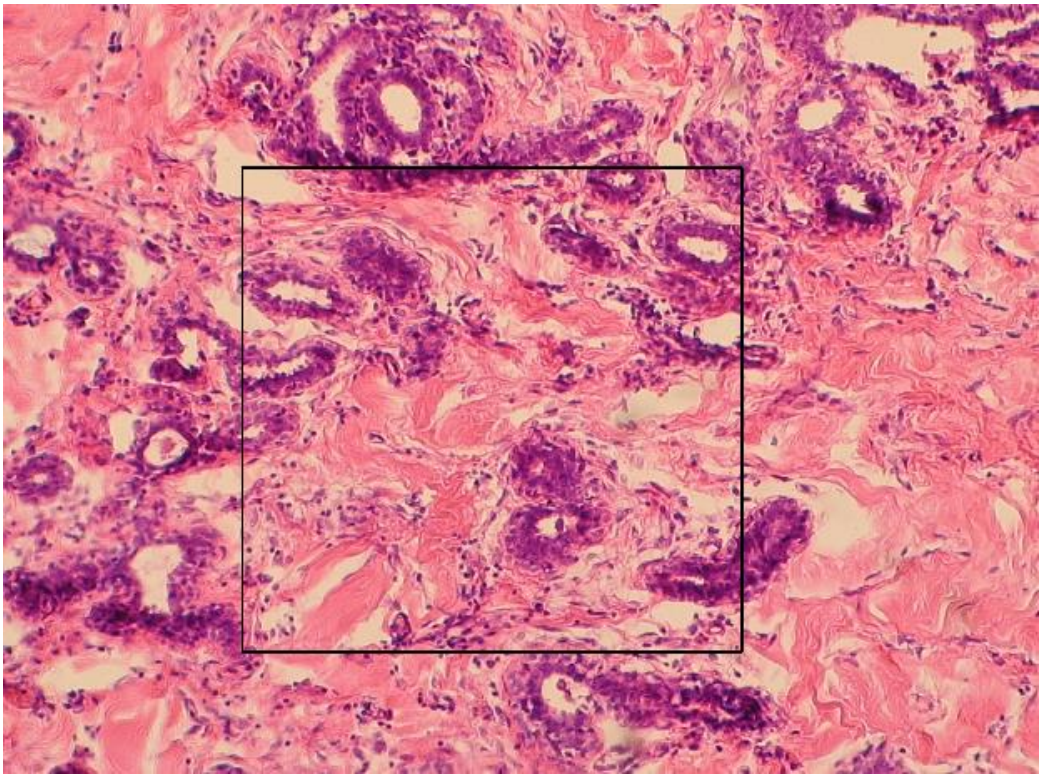


Рисунок 2.7 - Результат при a = 1

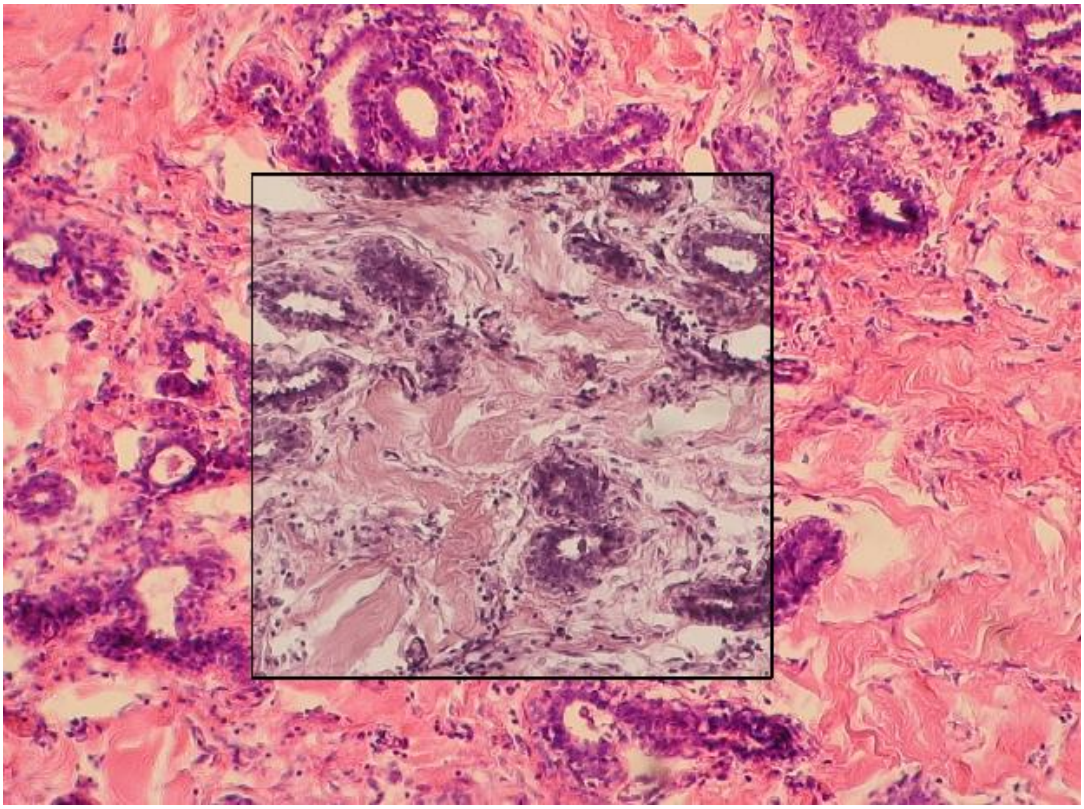


Рисунок 2.8 - Результат при a = 0.5

2.2.3 Зміна кольорового простору

Маючи 5 доступних кольорових просторів RGB, HSV, LAB, YCbCr та градації сірого (Gray), можно вивести наступні формули переходу між ними.

RGB → HSV:

$$R' = R/255$$

$$G' = G/255$$

$$B' = B/255$$

$$C_{max} = max(R', G', B')$$

$$C_{min} = min(R', G', B')$$

$$\Delta = C_{max} - C_{min}$$

$$H = \begin{cases} 0^\circ & \Delta = 0 \\ 60^\circ \times \left(\frac{G'-B'}{\Delta} mod 6\right) & , C_{max} = R' \\ 60^\circ \times \left(\frac{B'-R'}{\Delta} + 2\right) & , C_{max} = G' \\ 60^\circ \times \left(\frac{R'-G'}{\Delta} + 4\right) & , C_{max} = B' \end{cases}$$

$$S = \begin{cases} 0 & , C_{max} = 0 \\ \frac{\Delta}{C_{max}} & , C_{max} \neq 0 \end{cases}$$

$$V = C_{max}$$

RGB → LAB:

$$L = 116 f_y - 16$$
$$a = 500(f_x - f_y)$$
$$b = 200(f_y - f_z)$$
$$f_x = \begin{cases} \sqrt[3]{x_r}, x_r > \epsilon \\ \frac{k x_r + 16}{116}, x_r \leq \epsilon \end{cases}$$
$$f_y = \begin{cases} \sqrt[3]{y_r}, y_r > \epsilon \\ \frac{k y_r + 16}{116}, y_r \leq \epsilon \end{cases}$$
$$f_z = \begin{cases} \sqrt[3]{z_r}, z_r > \epsilon \\ \frac{k z_r + 16}{116}, z_r \leq \epsilon \end{cases}$$
$$x_r = \frac{X}{X_r}$$
$$y_r = \frac{Y}{Y_r}$$
$$z_r = \frac{Z}{Z_r}$$
$$\epsilon = 0.008856$$
$$k = 903.3$$

RGB → YCbCr:

$$Y' = 16 + \frac{65.738 \times R'_D}{256} + \frac{129.057 \times G'_D}{256} + \frac{25.064 \times B'_D}{256}$$

$$C_B = 128 + \frac{-37.945 \times R'_D}{256} - \frac{74.494 \times G'_D}{256} + \frac{112.439 \times B'_D}{256}$$

$$C_R = 128 + \frac{112.439 \times R'_D}{256} - \frac{94.154 \times G'_D}{256} + \frac{18.285 \times B'_D}{256}$$

RGB → Gray:

$$G = 0.2126\ R + 0.7152\ G + 0.0722\ B$$

Інші перетворення між кольоровими просторами здійснюється через простір RGB.

2.2.4 Гамма контраст

Якщо коефіцієнт контрасту γ, то:

$$Y = \gamma^{X/255} \times 255$$

На рис. 2.9-2.10 зображені результати роботи алгоритму.

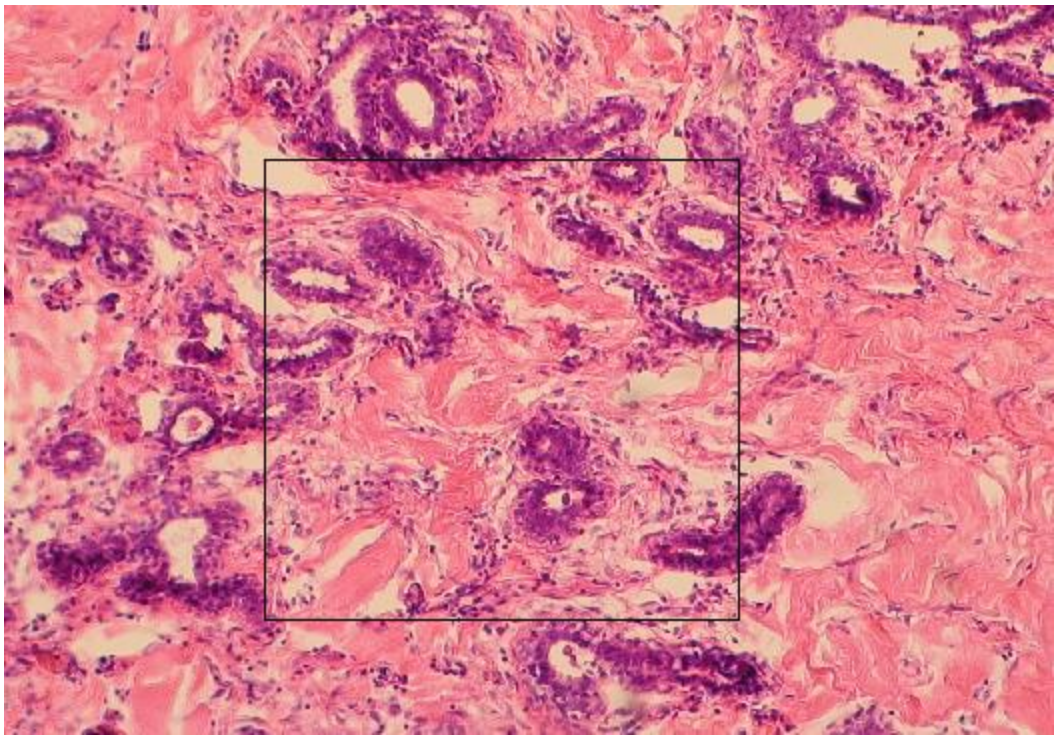


Рисунок 2.9 - Результат при γ=1

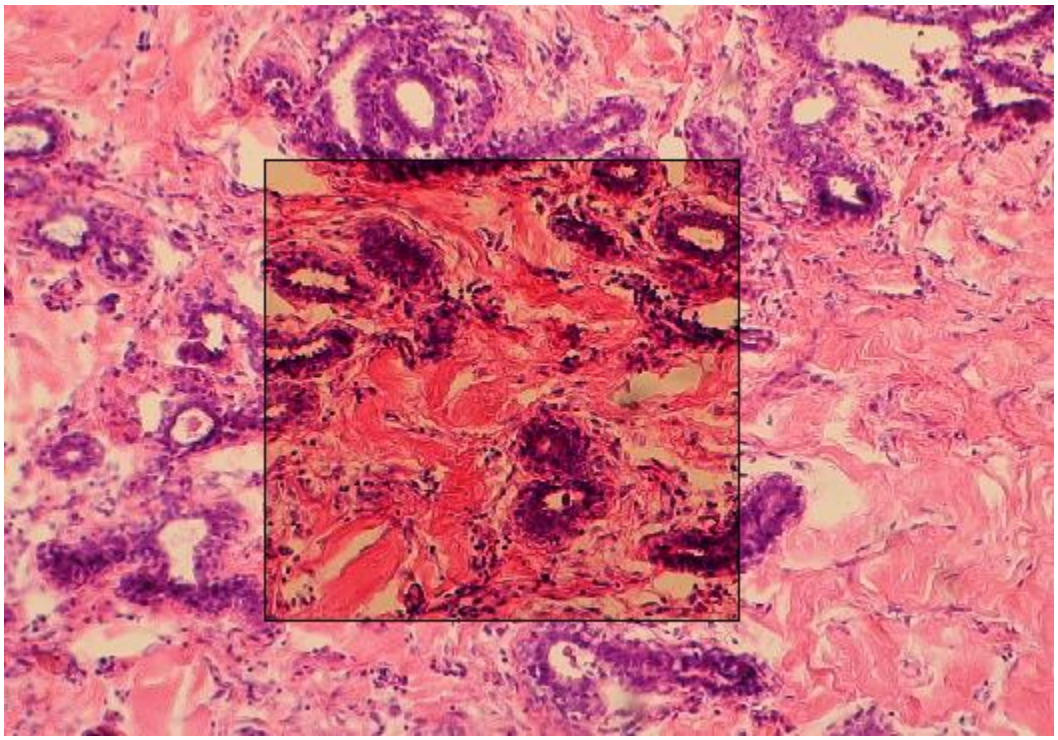


Рисунок 2.10 - Результат при γ=2

2.2.5 Лінійний контраст

Якщо коефіцієнт контрасту γ, то:

$$Y = 128 + (X - 128) \times \gamma$$

На рис. 2.11 зображений результат роботи алгоритму.

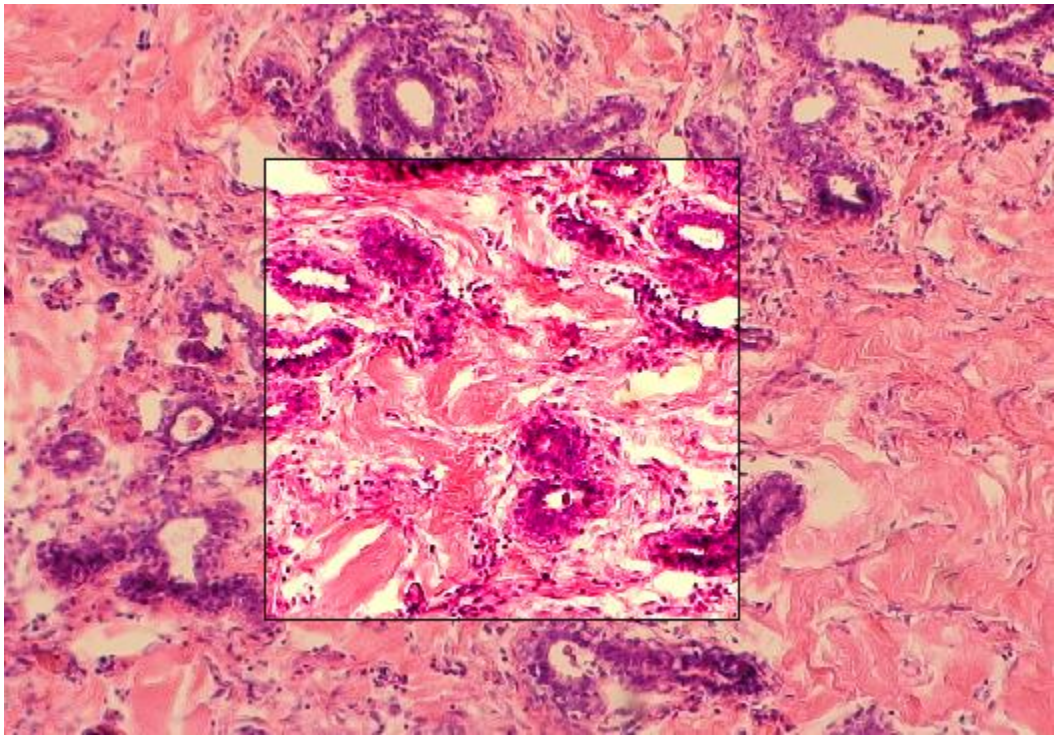


Рисунок 2.11 - Результат при γ=2

2.2.6 Кластерізація методом К-середніх

Якщо кількість кластерів k, то:

$$\forall i, j : x_{ij} \in X, y_{ij} \in Y, y_{ij} = \mu_l$$
$$l = \arg\min \left\| x_{ij} - \mu_o \right\|, o \in 1..k \text{ ,}$$

де S_l отримані мінімізуванням функції:

$$V = \sum_{i=1}^{k} \sum_{x \in S_i} (x - \mu_i)^2$$

На рисунках 2.12-2.15 зображені результати роботи алгоритму.

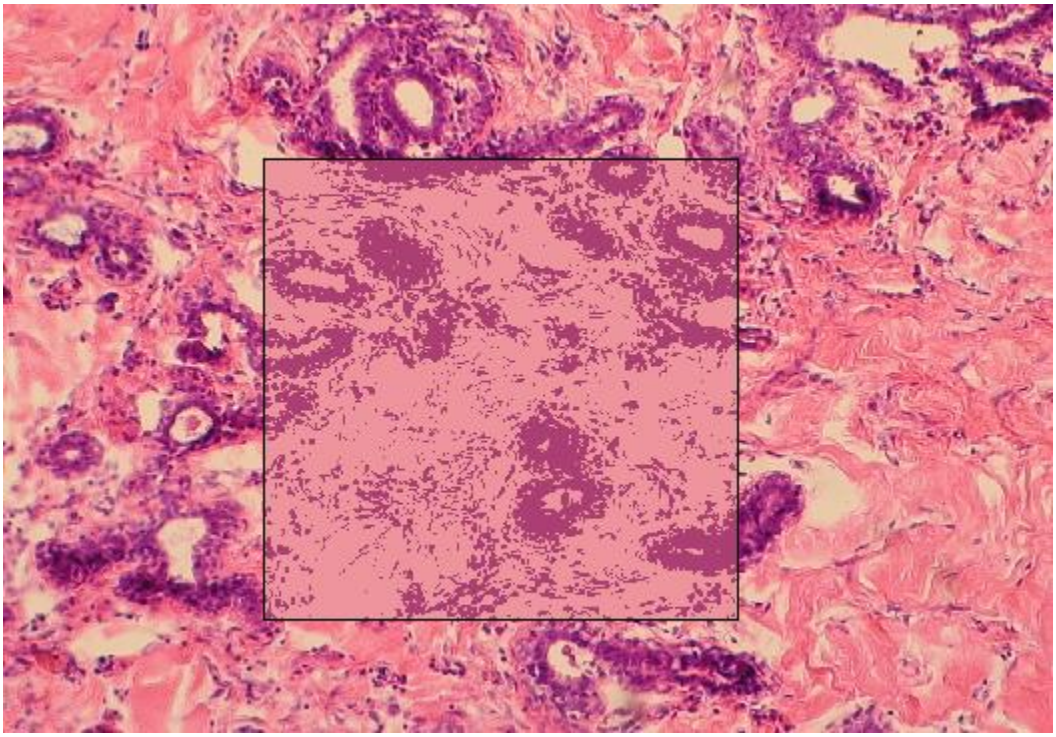


Рисунок 2.12 - Результат кластеризації зображення на 2 кластери

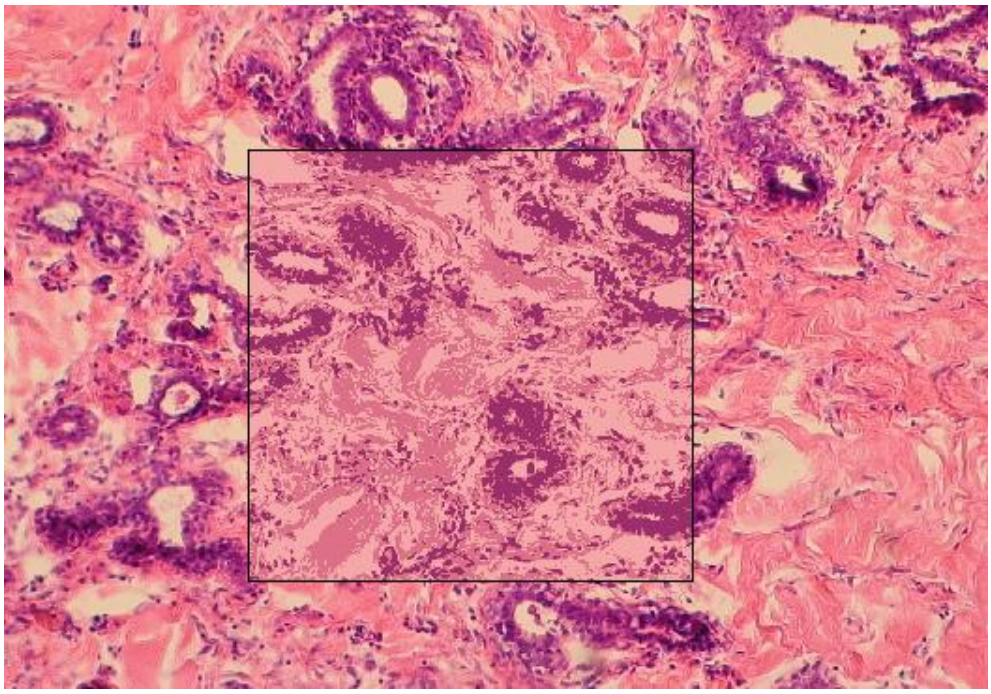


Рисунок 2.13 - Результат кластеризації зображення на 2 кластери

Метод кластеризації К-середніх може бути використаний для бінаризації зображень (рис 2.14).

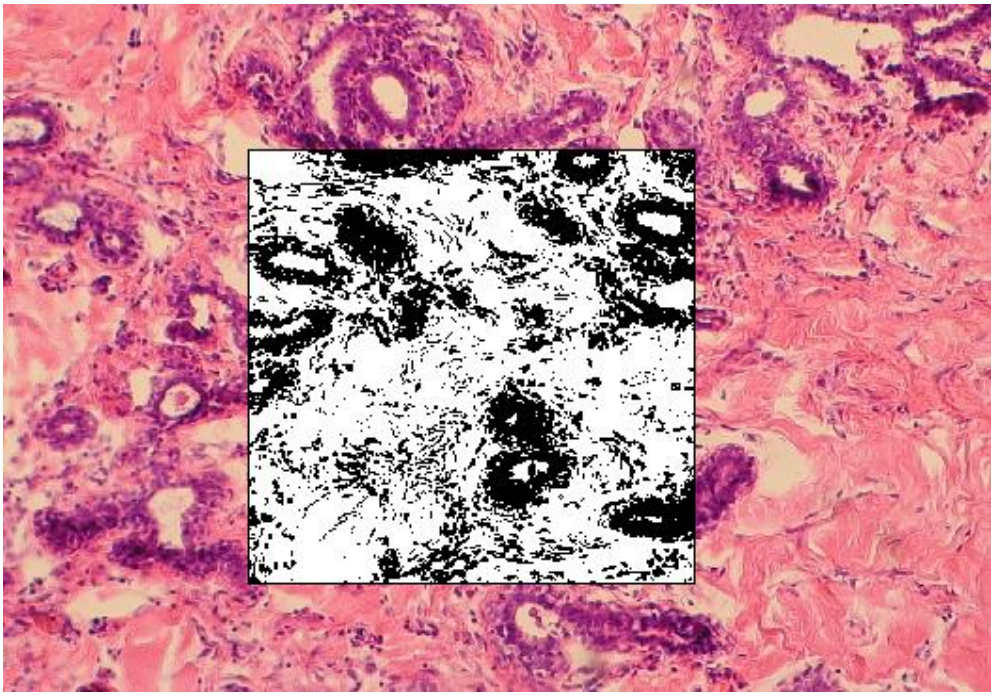


Рисунок 2.14 - Бінаризація методом К-середніх

2.2.7 Фільтр Гауса

Якщо розмір вікна фільтру k, то:

$$X = Y * g$$

$$g(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}, x, y \in \left(-\frac{k}{2}, \frac{k}{2}\right)$$

Приклад роботи алгоритму наведений на рисунку 2.15.

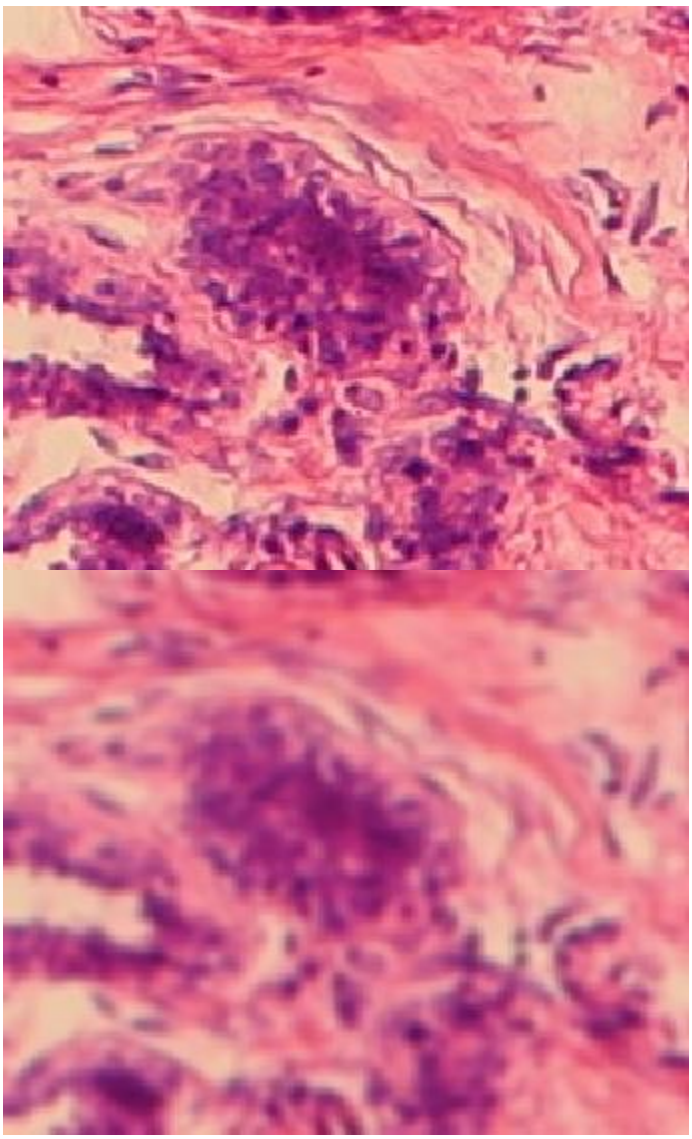


Рисунок 2.15 - Оригінальне зображення (зверху) та оброблене фільтром Гауса (знизу)

2.2.8 Морфологічні операції

Морфологічні операції представлені ерозією, розширенням, відкриттям, закриттям та морфологічним градієнтом. Якщо b – структурний елемент у матричній формі, то (у порядку ерозія, розширення):

$$(f \odot b)(X) = \inf_{Y \in B} \left[f(X+Y) - b(Y) \right]$$

$$(f \oplus b)(X) = \sup_{Y \in B} \left[f(Y) + b(X-Y) \right]$$

Відкриття задане як послідовне застосування розширення після ерозії, а закриття – навпаки. Приклади роботи алгоритмів наведені на рисунках 2.16-2.19.

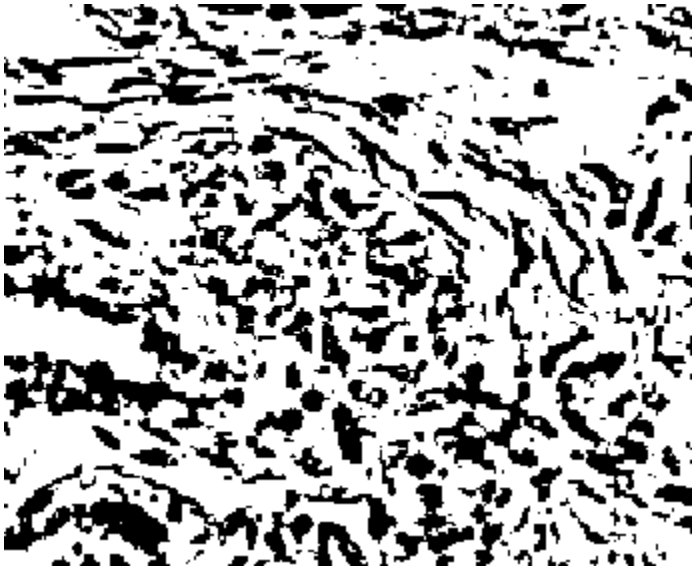


Рисунок 2.16 - Оригінальне бінарне зображення

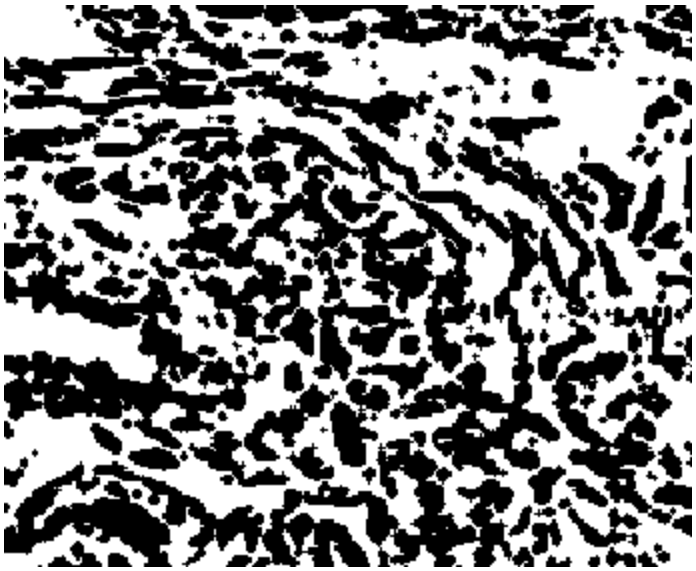


Рисунок 2.17 - Ерозія. Структурний елемент – елліпс, розмір 3х3

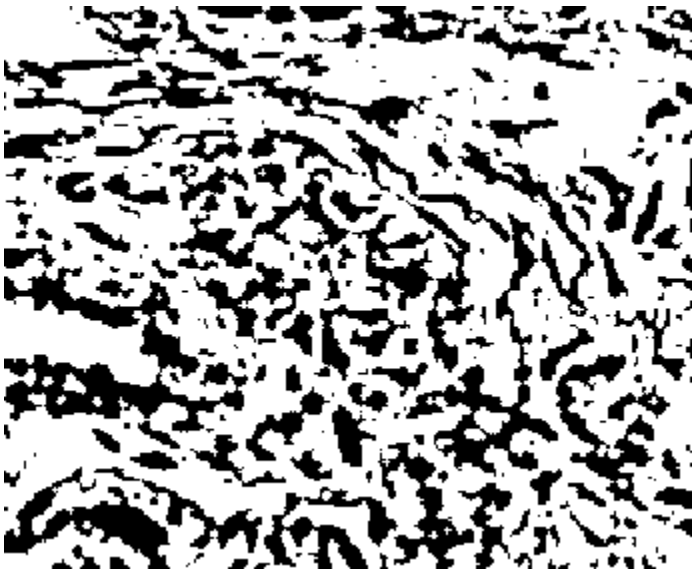


Рисунок 2.18 - Відкриття. Структурний елемент – елліпс, розмір 3х3

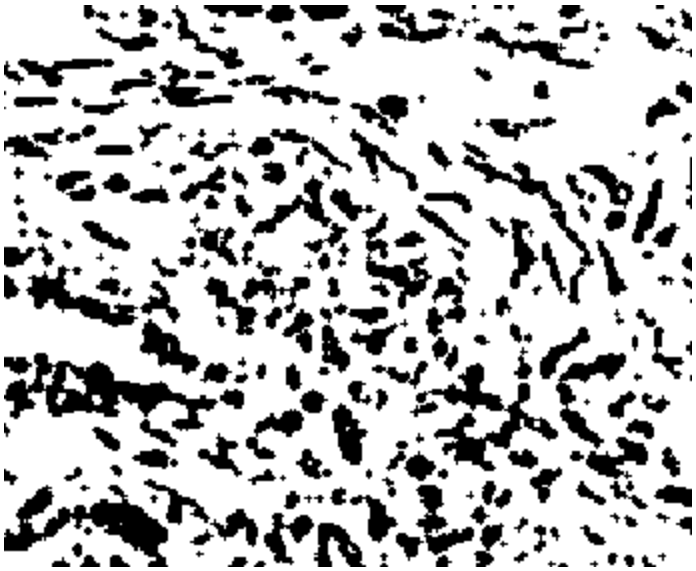


Рисунок 2.19 - Закриття. Структурний елемент – елліпс, розмір 3x3

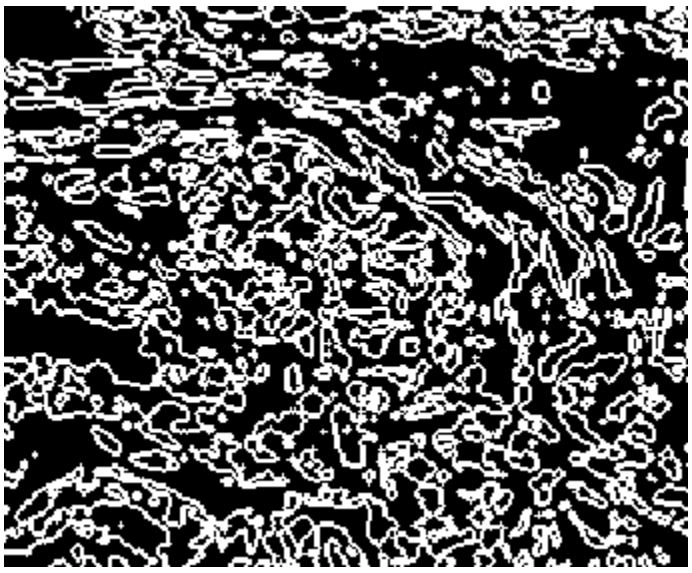


Рисунок 2.20 - Морфологічний градієнт. Структурний елемент – елліпс, розмір 3x3

2.2.9 Псевдовипадкове зображення

Зображення генерується за допомогою рівномірно-розподіленої випадкової величини:

$$\forall y \in Y, y \sim U(0,255)$$

2.2.10 Поканальний зріз

Якщо *c* – індекс каналу, то:

$$Y = X[:,:,c]$$

Приклад поканального зрізу наведений на рисунку 2.21.

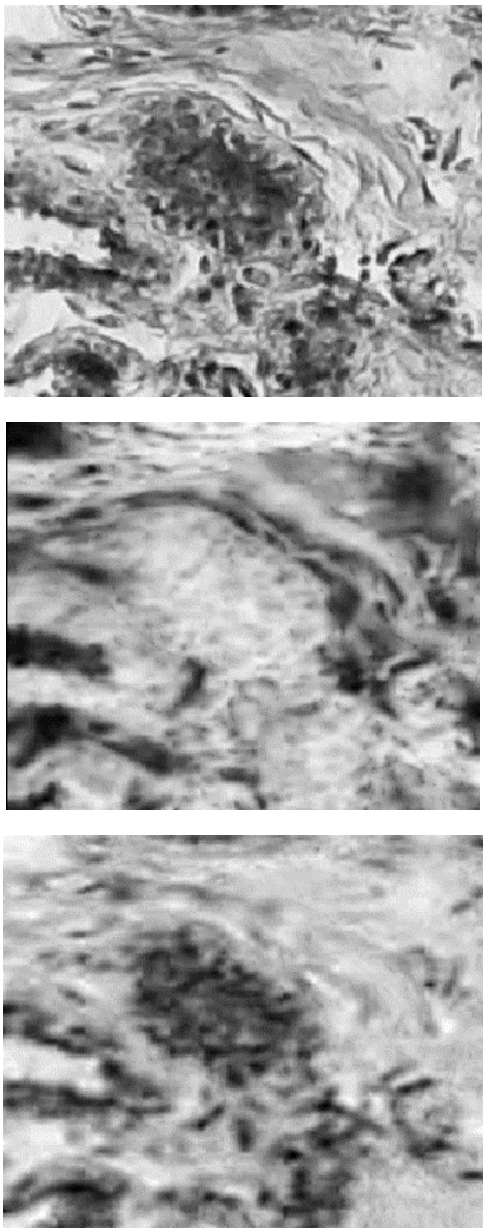


Рисунок 2.21 - Компоненти L, a, b (зверху вниз) для зображення з рисунку 2.15

2.2.11 Розбиття на канали

$$\forall i = 1..c, Y_i = X[:,:,i]$$

2.2.12 Злиття каналів

$$\forall i = 1..c, Y[:,:,i] = X_i$$

2.2.13 Бінаризація

Якщо b - поріг бінаризації, то:

$$\forall i, j : Y[i, j] = \begin{cases} 0, X[i, j] < b \\ 255, X[i, j] \geq b \end{cases}$$

На рис 2.22-2.23 представлені результати бінаризації з різними пороговими значеннями.

Рисунок 2.22 - Бінаризація з порогом 0.31

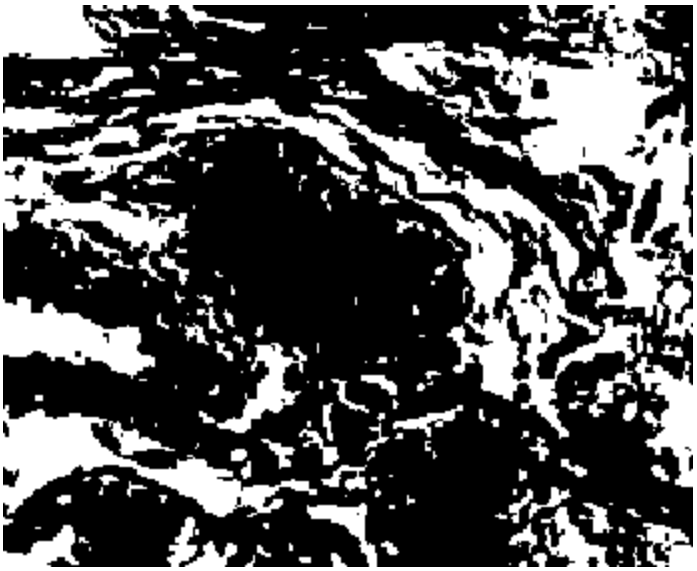


Рисунок 2.23 - Бінаризація з порогом 0.70

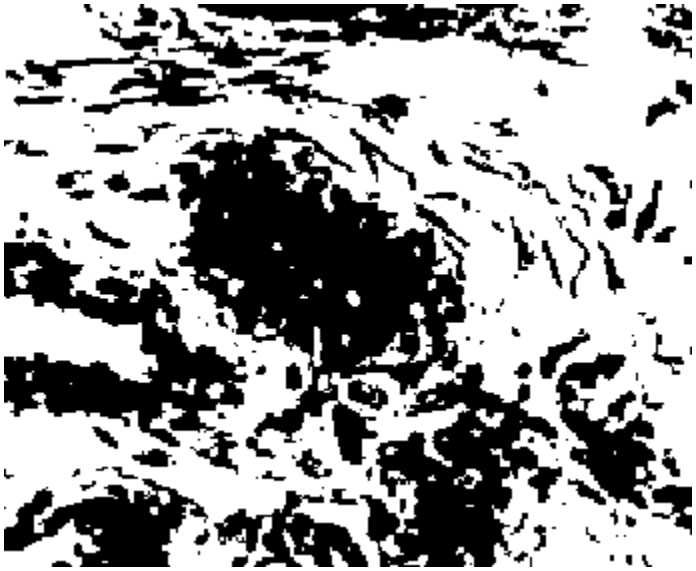


Рисунок 2.24 - Бінаризація з автоматичним визначення порогового значення за алгоритмом Оцу

2.2.14 Нормалізація інтенсивності

$$Y = norm\left(X - g\left(X, k\right)\right),$$ де

g – фільтр Гауса, k – розмір вікна, norm – гістограмне вирівнювання. Призначенням цієї операції є виключення фактору нерівномірного освітлення у роботі послідуючих етапі обробки. В якості прикладу можна привести

обробку алгоритмом бінаризації Оцу оригінального зображення, та зображення з нормалізованою освітленністю.

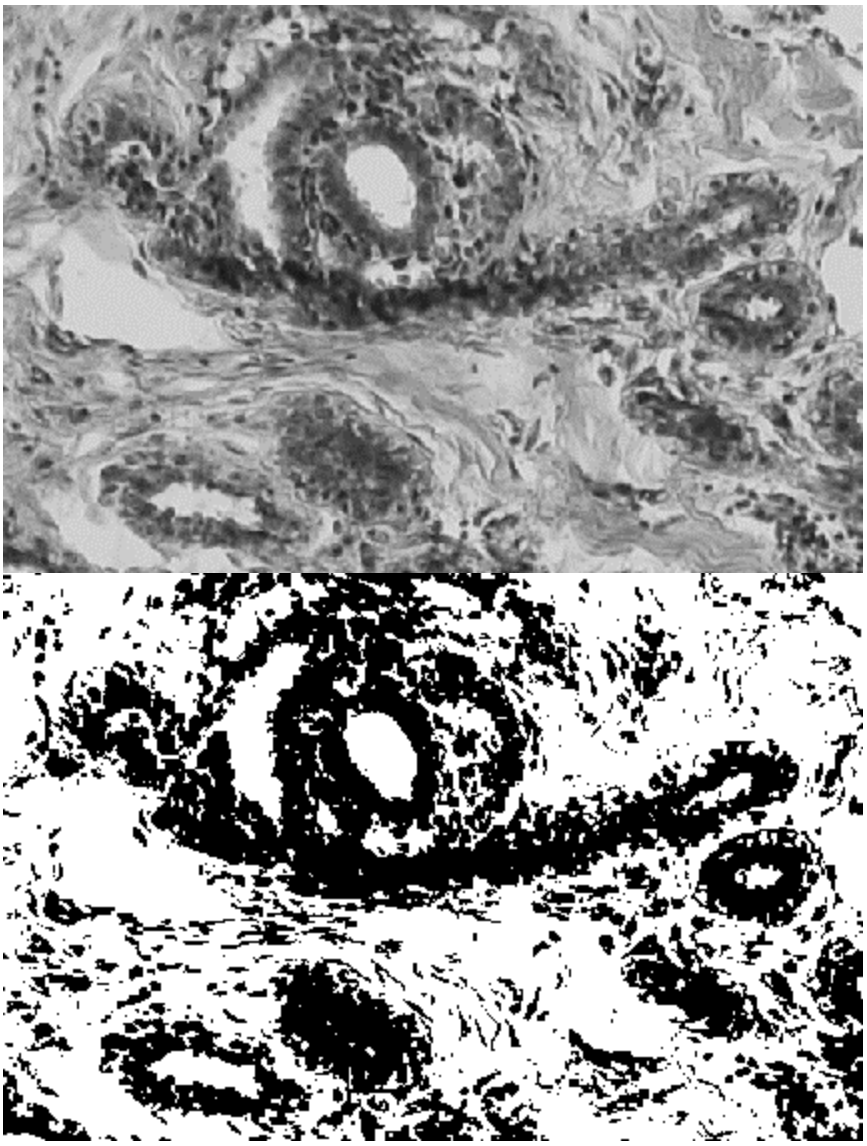


Рисунок 2.25 - Бінаризація ненормованного зображення

Як видно з рисунку 2.25, тканинні структури сприяють низькій якості бінаризації.

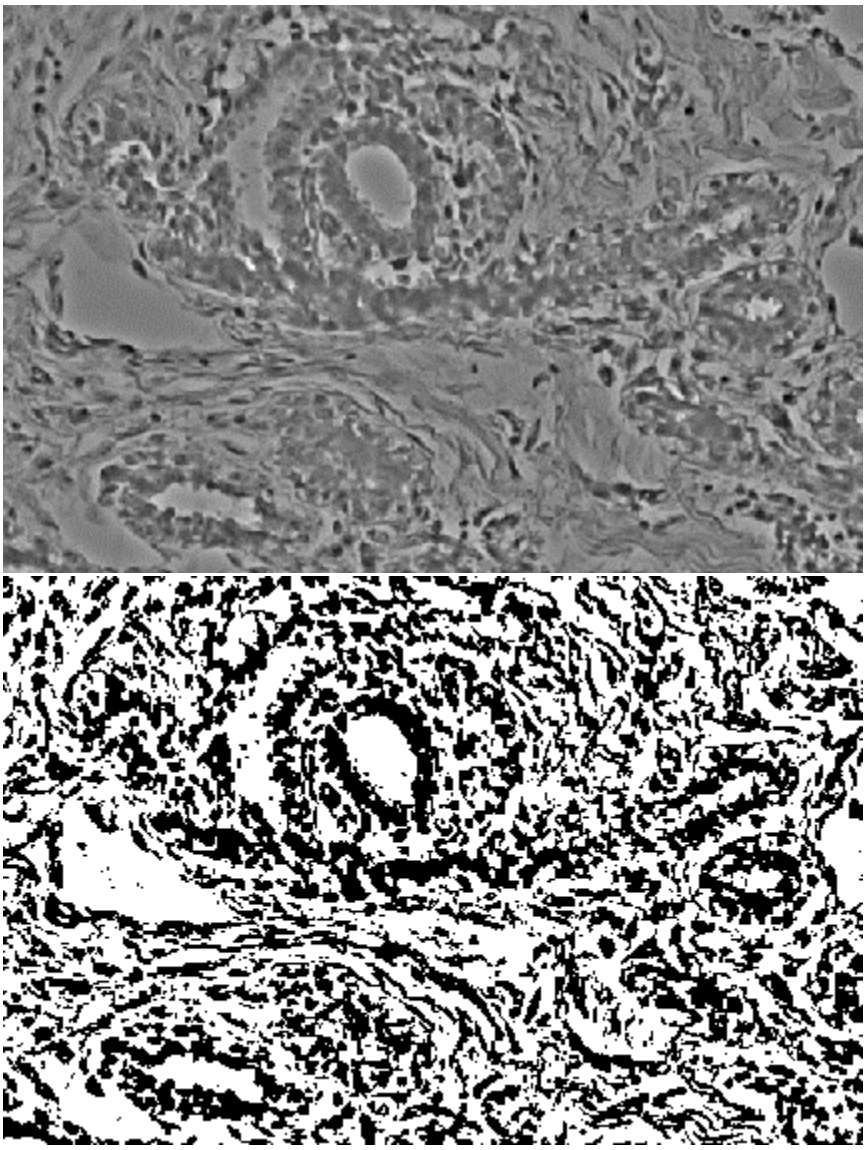


Рисунок 2.26 - Бінаризація нормованного зображення

На рисунку 2.26 зображене нормованне зображення (зверху), та результат його нормалізації. Як можно побачити, неоднорідності тканинних структур в набагато меншій степені впливають на якість бінаризації.

У цьому розділі було виконане дослідження знайдених алгоритмів, виокремлення множини вхідних, вихідних значень а також внутрішніх параметрів кожного з них, опрацювання принципів їх роботи. Також, було виконана розробка загального концепту програмного комплексу.

3 ПРАКТИЧНА ЧАСТИНА

3.1 Мова, бібліотеки

Для реалізації програмного комплексу була обрана мова python.

Бібліотека PyQt для розробки інтерфейсу користувача та зв’язку між клієнтською та серверною частинами за протоколом WebSocket.

Бібліотеки numpy, scikit-learn та OpenCV були використані для реалізації методів обробки зображень.

3.2 Бази даних

Для тестування роботи методів обробки зображень була використана база зображень, надана Інститутом експериментальної патології, онкології і радіобіології ім. Р.Є. Кавецького НАН України.

3.3 Специфікація класів

Програмний код наведений у Додатку 1. Архітектура програмного комплексу наведена в графічних матеріалах.

Програмний комплекс спроектований у вигляді клієнт-серверного додатку, зв’язок між якими здійснюється за допомогою протоколу WebSockets. Робота алгоритмів обробки зображень здійснюється на серверній частині. Результат обробки зображень відправляється на клієнтську частину та відображається користувачу. Так, серверна частина представлена наступними абстракціями:

Клас Graph.

Відповідає за роботу с графом, підтримує операції додавання/видалення вершини/ребра, підтримує цілісність та ациклічність графу, виконує операції серіалізації/десеріалізації графу.

Клас GraphServer.

Відповідає за роботу WebSocket серверу.

Клас Node.

Є абстракцією вершини графу та елементарною обрахунковою одиницею програмного комплексу. Специфікує типи вхідних/вихідних данних, внутрішні параметри та обрахунковий алгоритм; підтримує процес перевірки готовності роботи обрахункового алгоритму за значеннями вхідних даних та виконання самого алгоритму.

Модуль nodes.

Містить нащадків класу Node з указанням конкретних типів данних та алгоритмів.

Клас Edge.

Є абстракцією ребра обрахункового графу. Підтримує своєчасне створення/видалення програмних хуків.

Клас Datatype та нащадки.

Задає доступні для використання типи вхідних/вихідних даних.

Клас Datavalue та нащадки.

Є абстракцією значення входу/виходу/внутрішнього параметра. Відповідає за серіалізацію/десеріалізацію.

Клієнтська частина представлена наступними абстракціями:

Клас GraphWidget.

Відповідає за виклики методів відображення графу та підтримку WebSocket-клієнту.

Клас NodeWidget.

Відповідає за відображення вершин обрахункового графу.

Клас EdgeWidget.

Відповідає за відображення ребер обрахункового графу.

Клас Terminal.

Відповідає за відображення входів/виходів вершин обрахункового графу.

4 АНАЛІЗ РОБОТИ ПРОГРАМНОГО КОМПЛЕКСУ

4.1 Результати поліпшення наочності зображення.

В якості тестового зображення було обрано гістологічне зображення на рис. 4.1.

Рисунок 4.1 - Тестове гістологічне зображення у вікні програмного комплексу

Як можна побачити с рис. 4.1, представлений гістологічний матеріал має значну гетерогенну структуру, що супроводжується наявністю областей з різною кольоровою інтенсивністю. Для нівелювання ефекту

неоднорідностей, був розроблений обрахунковий граф, представлений на рис. 4.2.

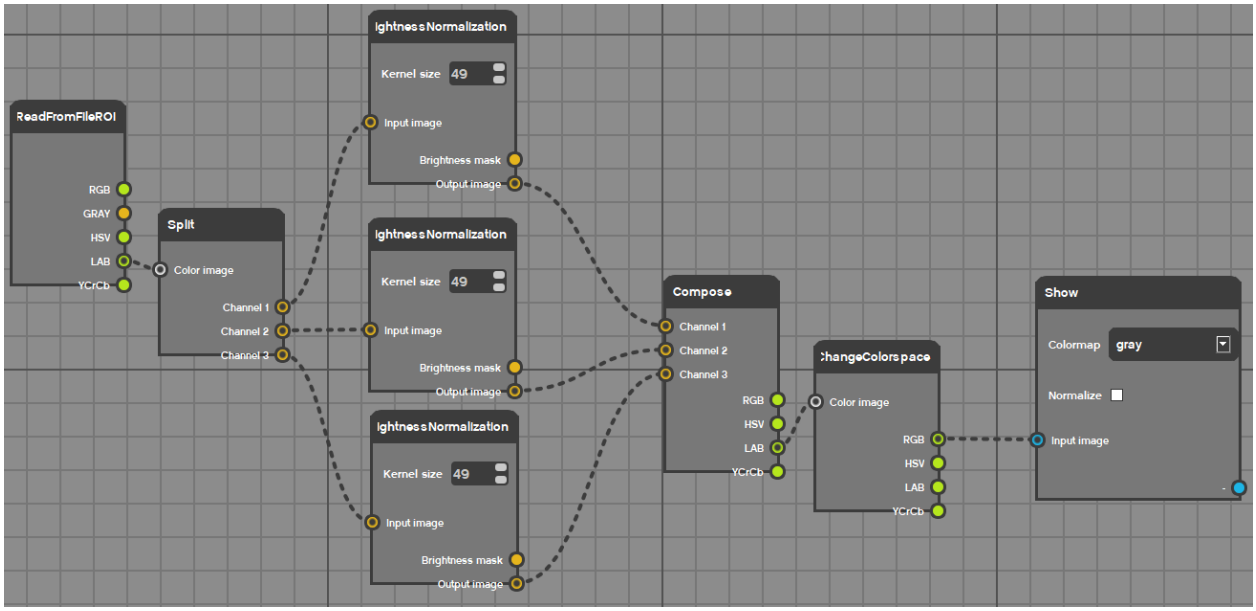


Рисунок 4.2 - Розроблений обрахунковий граф

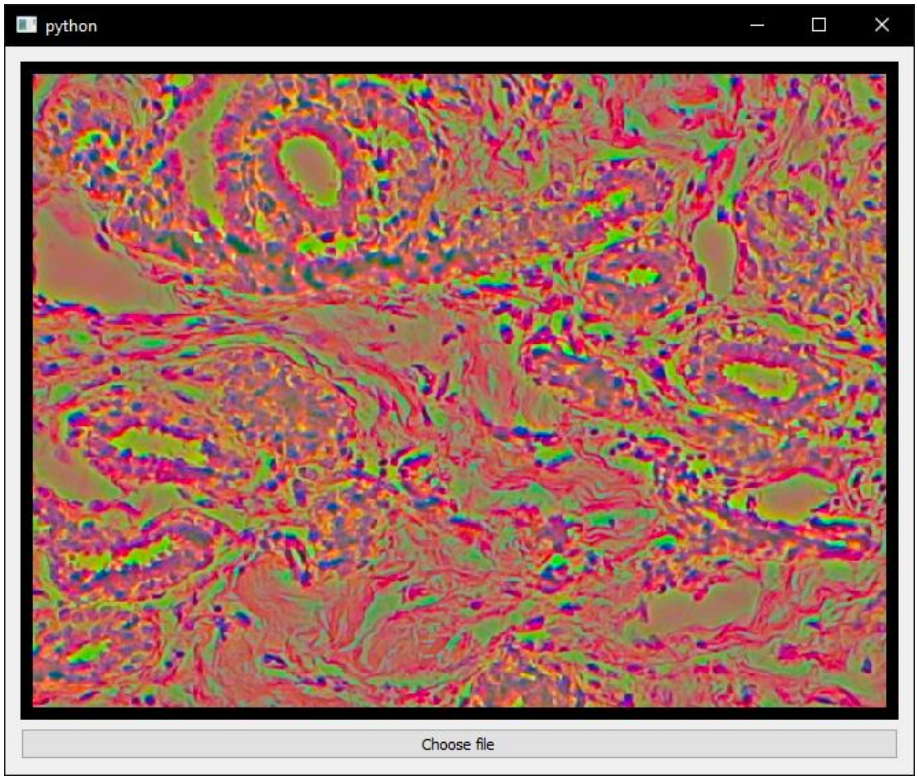


Рисунок 4.3 - Результат роботи графу

Результат роботи графу для заданого зображенння представлений на рис. 4.3.

Як можна побачити, вплив гетерогенності середи на якість зображення був значно зменшений. Так, мікроструктури, виокремлені синім кольором стали більш чіткими, що зробить їх аналіз більш простим як для людини, так і для подальшої комп’ютерної обробки. Наприклад, за допомогою бінаризації з автоматичним порогам, можна досягти автоматичної сегментації мікроструктур. Результат наведений на рис. 4.4.

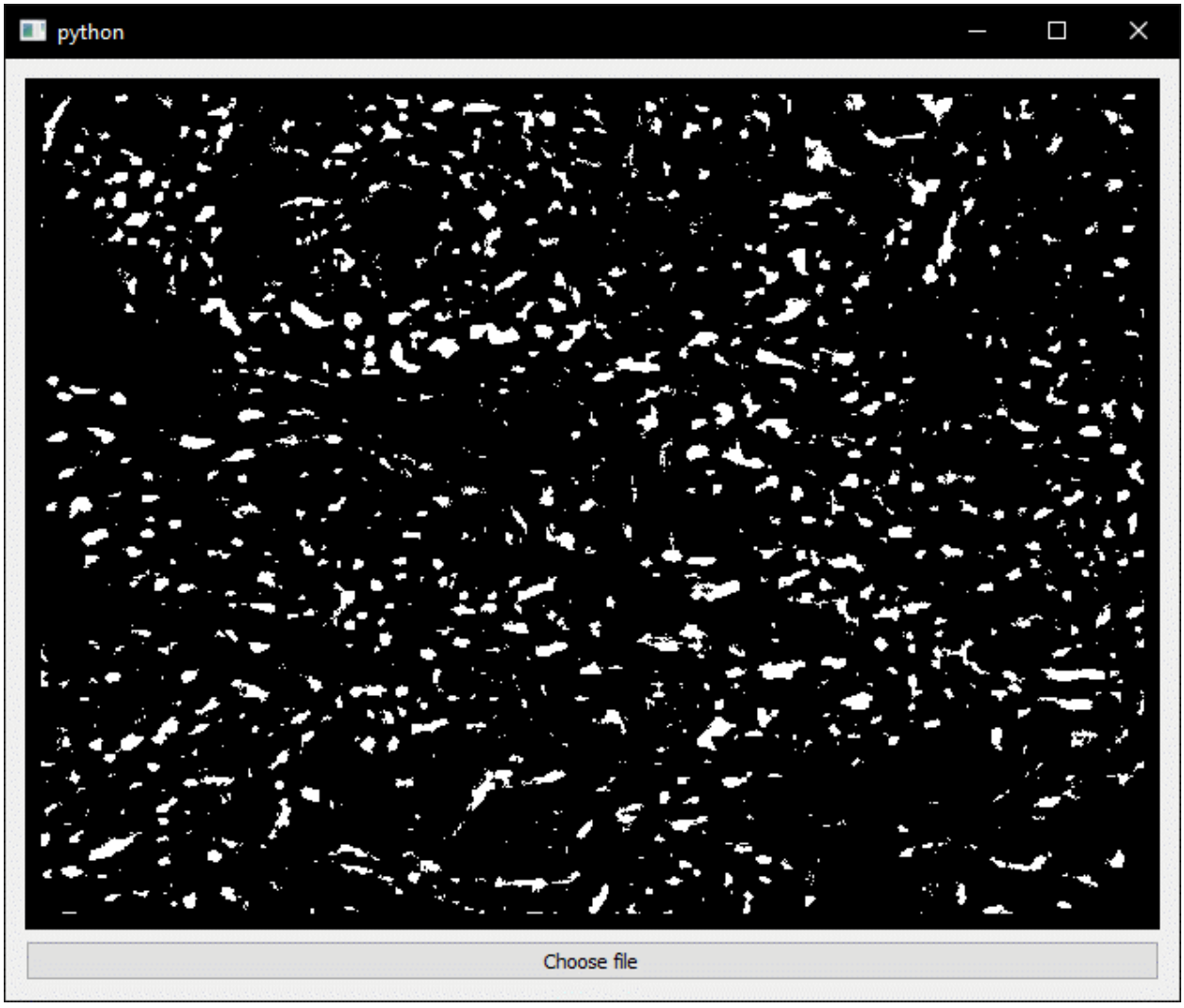


Рисунок 4.4 - Результат автоматичної сегментації мікроструктур

Також було виконане тестування програмного комплексу для задачі сегментації на даних з EM segmentation challenge [15]. Результати наведені у таблиці 4.2.

Таблиця 4.2 – Результати сегментації на даних з EM segmentation challenge [5]

Rank	Group name	Warping error	Rand error	Pixel error
-	** human values **	0.000005	0.0021	0.0010
1	u-net	0.000353	0.0382	0.06112
2	DIVE-SCI	0.000355	0.0305	0.0584
3	тестований метод	0.000483	0.0646	0.0758
4	IDSIA-SCI	0.000653	0.0189	0.1027

Як можна побачити, спроектований алгоритм здатний конкурувати з існуючими методами.

Таким чином, розроблена методологія компонування алгоритмів обробки зображень у єдиний алгоритм є достатньо гнучкою та потужною для моделювання складних функцій.

ВИСНОВКИ

Дослідження методів полуавтоматичного або автоматичного прогнозування та діагностики онкологічних захворювань є надзивчайно приорітетною задачею, що стоїть перед людством. Сучасні методи базуються на аналізі людиною мікрозображень зразків тканини, взятої з пухлини за допомогою біопсії. І хоча цьому процесу вже багато років, існує значна ймовірність того, що зразок буде містити артефакти, ускладнюючі подальший аналіз. В зв’язку з цим, приймаючи до уваги останні досягнення в області комп’ютерного зору та найпотужніші в історії людства обчислювальні ресурси, доступні до використання, питання заміни, хоча би часткової, людського експерта на комп’ютерну програми має сенс.

Для використання комп’ютерної програми для зазначених цілей, вона має виконувати хоча б одну з двох функцій: поліпшення якості зображення для людини; автоматичний аналіз компонентів структури, наявних на зображенні. В рамках цієї роботи була проаналізована відповідна література та створений програмний комплекс, що дозволяє користувачу самостійно створювати алгоритм аналізу зображення, маючи в якості елементарних одиниць окремі алгоритми, що використовуються в комп’ютерному зорі. Ці одиниці можуть бути сгруповані в обрахунковий граф, при якому входами одних алгоритмів слугують виходи з попередніх. Також, кожен алгоритм може бути налаштований під потреби вирішуваної задачі. Окрім цього, було проведене тестування роботи програмного комплексу для задачі бінарної сегментації мікроструктур на неоднорідному зображенні.

Перевагами розробленої програми є гнучкість її налаштування користувачем, а також мінімальна складність додавання нових та модифікації існуючих елементарних алгоритмів.

До недоліків програмного комплексу можна віднести відносну складність його користування та неможливість автоматизувати процес створення обчислювального графу та налаштування його окремих частин.

СПИСОК ЛІТЕРАТУРИ

1) Centers for Desease Control and Prevention [Електронний ресурс] – Режим доступу до ресурсу: https://www.cdc.gov/.

2) Computer vision [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Computer_vision.

3) Label Refinement Network for Coarse-to-Fine Semantic Segmentation [Електронний ресурс] – Режим доступу до ресурсу: https://vitalab.github.io/deep-learning/2018/11/22/Label-Refinement-Network.html.

4) Faliu Y. Microvessel prediction in H&E Stained Pathology Images using fully convolutional neural networks / Y. Faliu, Y. Lin, W. Shidan. // BMC Bioinformatics. – 2018.

5) Olaf R. U-Net: Convolutional Networks for Biomedical Image Segmentation [Електронний ресурс] / R. Olaf, F. Philipp, B. Thomas. – 2015. – Режим доступу до ресурсу: https://arxiv.org/pdf/1505.04597.pdf..

6) Venkat Narayana Rao T. CLASSICAL AND NOVEL DIAGNOSIS TECHNIQUES FOR EARLY BREAST CANCER DETECTION – A COMPARATIVE APPROACH / T. Venkat Narayana Rao, A. Govardhan. // International Journal of Engineering Mathematics and Computer Sciences. – 2014. – С. 12–16.

7) Irshad, H., Veillard, A., Roux, L., Racoceanu, D., 2014. Methods for nuclei detection, segmentation, and classification in digital histopathology: a review–current status and future potential. IEEE Rev. Biomed. Eng. 7, 97–114. doi:10.1109/RBME.2013.2295804.

8) Fatakdawala, H., Xu, J., Basavanhally, A., Bhanot, G., Ganesan, S., Feldman, M., Tomaszewski, J.E., Madabhushi, A., 2010. Expectation-maximization-driven geodesic active contour with overlap resolution

(EMaGACOR): application to lymphocyte segmentation on breast cancer histopathology. IEEE Trans. Biomed. Eng. 57, 1676–1689. doi:10.1109/TBME.2010.2041232.

9) Glotsos, D., Spyridonos, P., Cavouras, D., Ravazoula, P., Dadioti, P.-A., Nikiforidis, G., 2004. Automated segmentation of routinely hematoxylin-eosin-stained microscopic images by combining support vector machine clustering and active contour models. Anal. Quant. Cytol. Histol. 26, 331–340.

10) Xu, J., Janowczyk, A., Chandran, S., Madabhushi, A., 2011. A high-throughput active contour scheme for segmentation of histopathological imagery. Med. Image Anal. 15, 851–862. doi:10.1016/j.media.2011.04.002.

11) Sirinukunwattana, K., Raza, S., Tsang, Y.-W., Snead, D., Cree, I., Rajpoot, N., 2016. Locality sensitive deep learning for detection and classification of nuclei in routine colon cancer histology images. IEEE Trans. Med. Imaging. doi:10.1109/TMI.2016.2525803..

12) Sirinukunwattana, K., Snead, D.R.J., Rajpoot, N.M., 2015. A stochastic polygons model for glandular structures in colon histology images. IEEE Trans. Med. Imaging 34, 2366–2378. doi:10.1109/TMI.2015.2433900.

13) Madabhushi A. Image analysis and machine learning in digital pathology: Challenges and opportunities / A. Madabhushi, G. Lee. // Medical Image Analysis. – 2016. – №33. – С. 170–175.

14) Gaussian filter [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Gaussian_filter.

15) ISBI Challenge: Segmentation of neuronal structures in EM stacks [Електронний ресурс] – Режим доступу до ресурсу: http://brainiac2.mit.edu/isbi_challenge/.

ДОДАТОК 1 ПРОГРАМНИЙ КОД

graph.py

```
from PyQt5 import QtWidgets
from PyQt5 import QtCore, QtGui
from PyQt5 import Qt
from PyQt5 import QtWebSockets

import sys

from roi_image_widget import ImageROIWidget

from ui import GraphWidget
from graph_server import GraphServer
from local_ws_client import LocalWebSocket

app = QtWidgets.QApplication(sys.argv)

client_ws = LocalWebSocket()
server_ws = LocalWebSocket()
client_ws.connect(server_ws)

serverObject = QtWebSockets.QWebSocketServer(
    'My Socket', QtWebSockets.QWebSocketServer.NonSecureMode)
server = GraphServer(serverObject, local_ws=server_ws)
serverObject.closed.connect(app.quit)

image_view = ImageROIWidget()
client = GraphWidget(image_view, local_ws=client_ws)

client.show()
image_view.show()
client.resize(800, 600)
app.exec_()
```

graph_client.py

```
from PyQt5 import QtWidgets

import sys

from ui import GraphWidget
from roi_image_widget import ImageROIWidget


app = QtWidgets.QApplication(sys.argv)
image_view = ImageROIWidget()
widget = GraphWidget(image_view)
widget.resize(800, 600)

widget.show()
image_view.show()
app.exec_()
```

graph_server.py

```
from core import Graph, Node, Edge
from core.nodes import str_to_classname
from utils import bidict

from PyQt5.QtWidgets import QApplication
from PyQt5 import QtCore, QtWebSockets, QtNetwork
import json
import uuid

import logging
logging.basicConfig(level=logging.WARN)


class GraphServer(QtCore.QObject):
    def __init__(self, parent, host='127.0.0.1', port=1302, local_ws=None):
        super(QtCore.QObject, self).__init__(parent)
        self.client_graphs = dict()
        self.uuid_object_map = dict()
        self.is_remote_mode = (local_ws is None)
        if local_ws is None:
            print("server name: {}".format(parent.serverName()))
            self.server = QtWebSockets.QWebSocketServer(
                parent.serverName(), parent.secureMode(), parent)
            if self.server.listen(QtNetwork.QHostAddress(host), port):
                print(
                    'Listening: {}:{}:{}'.format(
                        self.server.serverName(),
self.server.serverAddress().toString(), str(
                            self.server.serverPort())))
            else:
                print('error')
            self.server.acceptError.connect(self.onAcceptError)
            self.server.newConnection.connect(self.onNewConnection)
            self.clientConnection = None
            print('Server listening' if self.server.isListening()
                  else 'Server not listening')
        else:
            self.clientConnection = local_ws
            self.clientConnection.textMessageReceived.connect(
                self.process_action)
            self.client_graphs[self.clientConnection] = Graph()
            self.uuid_object_map[self.clientConnection] = bidict()

    def onAcceptError(self, accept_error):
        print("Accept Error: {}".format(accept_error))

    def onNewConnection(self):
        logging.debug('connected')
        self.clientConnection = self.server.nextPendingConnection()
        self.clientConnection.textMessageReceived.connect(self.process_action)

        self.clientConnection.disconnected.connect(self.socketDisconnected)

        self.client_graphs[self.clientConnection] = Graph()
        self.uuid_object_map[self.clientConnection] = bidict()
```

```
    def process_action(self, message):
        self.clientConnection = self.sender()
        logging.debug('process_action: received %s' % message)
        if self.sender() and self.clientConnection in self.client_graphs:
            if isinstance(message, str):
                message = json.loads(message)
            action = message['action']

            if action == 'add_node':
                self.add_node(message)
            elif action == 'remove_node':
                self.remove_node(message)
            elif action == 'add_edge':
                self.add_edge(message)
            elif action == 'remove_edge':
                self.remove_edge(message)
            elif action == 'change_param':
                self.change_param(message)
            elif action == 'serialize':
                self.serialize(message)
            elif action == 'restore':
                self.restore(message)

        else:
            logging.warn('Something wrong with clientConnection')

        logging.info(list(self.uuid_object_map[self.clientConnection].keys()))

    def add_node(self, message):
        graph = self.client_graphs[self.clientConnection]

        if 'node_id' in message:
            node_id = message['node_id']
        else:
            node_id = uuid.uuid4().hex
        node = str_to_classname[message['node_classname']]()
        graph.add_node(node)
        self.uuid_object_map[self.clientConnection][node_id] = node
        node.valueChanged.connect(self.onNodeValueChanged)

        response_action = message
        response_action['node_id'] = node_id
        if self.is_remote_mode:
            response_action = json.dumps(response_action)
        logging.debug('add_node response: %s' % response_action)
        self.clientConnection.sendTextMessage(response_action)

        # Try to update new node
        node.update()

    def add_edge(self, message):
        graph = self.client_graphs[self.clientConnection]

        start_node =
self.uuid_object_map[self.clientConnection][message['start_node_id']]
```

```
        end_node =
self.uuid_object_map[self.clientConnection][message['end_node_id']]
        start_out_name = message['start_out_name']
        end_in_name = message['end_in_name']

        # Check if end_node is already occupied
        for start in graph.adj_matrix.values():
            if end_node in start:
                for edge in start[end_node]:
                    if edge.end_in_name == end_in_name:
                        return

        # Check datatypes
        if not (
            issubclass(
                start_node.out_types[start_out_name],
                end_node.in_types[end_in_name])):
            return

        edge_id = uuid.uuid4().hex
        edge = Edge(start_node, end_node, start_out_name, end_in_name)
        self.uuid_object_map[self.clientConnection][edge_id] = edge
        graph.add_edge(edge)

        response_action = message
        response_action['edge_id'] = edge_id
        if self.is_remote_mode:
            response_action = json.dumps(response_action)
        logging.debug('edge_node response: %s' % response_action)
        self.clientConnection.sendTextMessage(response_action)

    def remove_node(self, message):
        graph = self.client_graphs[self.clientConnection]
        node_id = message['node_id']
        node = self.uuid_object_map[self.clientConnection][node_id]

        deleted_edges = graph.remove_node(node)

        response_action = dict(
            action='remove',
            node_ids=[node_id],
            edge_ids=[self.uuid_object_map[self.clientConnection][e]
                      for e in deleted_edges]
        )
        if self.is_remote_mode:
            response_action = json.dumps(response_action)
        logging.debug('remove_node response: %s' % response_action)
        self.clientConnection.sendTextMessage(response_action)

        del self.uuid_object_map[self.clientConnection][node]
        for e in deleted_edges:
            del self.uuid_object_map[self.clientConnection][e]

    def remove_edge(self, message):
        graph = self.client_graphs[self.clientConnection]
        edge_id = message['edge_id']
```

```
        edge = self.uuid_object_map[self.clientConnection][edge_id]
        graph.remove_edge(edge)
        del self.uuid_object_map[self.clientConnection][edge_id]
        response_action = dict(
            action='remove',
            node_ids=[],
            edge_ids=[edge_id]
        )
        if self.is_remote_mode:
            response_action = json.dumps(response_action)
        logging.debug('remove_edge response: %s' % response_action)
        self.clientConnection.sendTextMessage(response_action)

    def change_param(self, message):
        node_id = message['node_id']
        node = self.uuid_object_map[self.clientConnection][node_id]
        param_value = node.parameters[message['param_name']].datavalue
        value = param_value.decode(message['value'])
        param_value.set_data(value)
        node.update()

    def onNodeValueChanged(self):
        node = self.sender()
        node_id = self.uuid_object_map[self.clientConnection][node]
        encoded = {k: v.encode(compress=self.is_remote_mode)
                   for k, v in node.out_values.items()
                   if k in node.out_values_to_return
                   }

        request_action = dict(
            action='update_values',
            node_id=node_id,
            values=encoded
        )
        if self.is_remote_mode:
            request_action = json.dumps(request_action)
        self.clientConnection.sendTextMessage(request_action)

    def serialize(self, message):
        serialized = dict(
            nodes=list(),
            edges=list()
        )
        objs = self.uuid_object_map[self.clientConnection]
        for _id, obj in objs.items():
            if isinstance(obj, Node):
                parameters = obj.serialize()
                serialized['nodes'].append(dict(
                    node_id=_id,
                    node_classname=type(obj).__name__,
                    parameters=parameters
                ))
            elif isinstance(obj, Edge):
                serialized['edges'].append(dict(
                    node_id=_id,
                    start_node_id=objs[obj.start_node],
```

```
                    end_node_id=objs[obj.end_node],
                    start_out_name=obj.start_out_name,
                    end_in_name=obj.end_in_name
                ))

        request_action = dict(
            action='serialize',
            save_filename=message['save_filename'],
            serialized=serialized
        )
        if self.is_remote_mode:
            request_action = json.dumps(request_action)
        self.clientConnection.sendTextMessage(request_action)

    def socketDisconnected(self):
        logging.debug('disconnected')
        if self.clientConnection:
            if self.clientConnection in self.client_graphs:
                del self.client_graphs[self.sender()]
            if self.clientConnection in self.uuid_object_map:
                del self.uuid_object_map[self.sender()]
            self.clientConnection.deleteLater()


if __name__ == '__main__':
    import sys
    app = QApplication(sys.argv)
    serverObject = QtWebSockets.QWebSocketServer(
        'My Socket', QtWebSockets.QWebSocketServer.NonSecureMode)
    server = GraphServer(serverObject)
    serverObject.closed.connect(app.quit)
    app.exec_()
```

local_ws_client.py

```
from PyQt5 import QtCore


class LocalWebSocket(QtCore.QObject):

    textMessageReceived = QtCore.pyqtSignal(dict)
    binaryMessageReceived = QtCore.pyqtSignal(bytes)

    def __init__(self):
        super(LocalWebSocket, self).__init__()
        self.target = None

    def connect(self, local_ws):
        if self.target is None:
            self.target = local_ws
            local_ws.connect(self)

    def sendTextMessage(self, message):
        self.target.textMessageReceived.emit(message)

    def sendBinaryMessage(self, message):
        self.target.binaryMessageReceived.emit(message)
```

```
roi_image_widget.py
from PyQt5 import QtWidgets
from PyQt5 import QtCore, QtGui

import numpy as np
import pyqtgraph as pg
import cv2

import os


class ImageROI(pg.GraphicsLayoutWidget):

    roiChanged = QtCore.pyqtSignal()

    def __init__(self):
        super(ImageROI, self).__init__()
        self.view_box = self.addViewBox(lockAspect=True, invertY=True)

        self.image_item = pg.ImageItem()
        self.view_box.addItem(self.image_item)

        roi_pen = QtGui.QPen()
        roi_pen.setWidth(3)
        self.roi_plot = pg.PlotCurveItem(pen=roi_pen)
        self.view_box.addItem(self.roi_plot)

        self.view_box.setMenuEnabled(False)

        self.click = 0
        self.selected_image = None

    def setROI(self, l, t, r, b):
        l, r = min(l, r), max(l, r)
        t, b = min(t, b), max(t, b)
        self.roi = l, t, r, b
        self.roi_plot.setData(x=np.array(
            [l, r, r, l, l]), y=np.array([t, t, b, b, t]))
        self.selected_image = self.image[t:b, l:r]
        self.roiChanged.emit()

    def setImage(self, image):
        self.image_item.setImage(image.transpose(1, 0, 2))
        self.image = image
        self.setROI(
            image.shape[0] // 4,
            image.shape[1] // 4,
            3 * image.shape[0] // 4,
            3 * image.shape[1] // 4)

    def mousePressEvent(self, event):
        if event.buttons() == QtCore.Qt.LeftButton:
            self.click = 1
        elif event.buttons() == QtCore.Qt.RightButton:
            self.click = 2
```

```
            super(ImageROI, self).mousePressEvent(event)

        def mouseMoveEvent(self, event):
            self.click = 0
            super(ImageROI, self).mouseMoveEvent(event)

        def mouseReleaseEvent(self, event):
            if self.click > 0 and hasattr(self, 'roi'):
                pos = event.pos()
                pos = pos.x() / self.width(), pos.y() / self.height()
                rect = self.view_box.viewRect()
                coord = rect.x() + pos[0] * \
                    rect.width(), rect.y() + pos[1] * rect.height()
                coord = np.round(coord).astype(np.int32)

                if self.click == 1:
                    self.setROI(*coord, *self.roi[2:])
                elif self.click == 2:
                    self.setROI(*self.roi[:2], *coord)
            self.click = 0
            super(ImageROI, self).mouseReleaseEvent(event)


    class ImageROIWidget(QtWidgets.QWidget):

        roiChanged = QtCore.pyqtSignal()

        def __init__(self):
            super(ImageROIWidget, self).__init__()

            layout = QtWidgets.QVBoxLayout()
            self.setLayout(layout)

            self.image_roi = ImageROI()
            self.choose_file_button = QtWidgets.QPushButton('Choose file')

            self.image_roi.roiChanged.connect(self.roiChanged)

            def choose_file(v):
                file, _ = QtWidgets.QFileDialog().getOpenFileName()
                if os.path.exists(file):
                    image = cv2.cvtColor(cv2.imread(file), cv2.COLOR_BGR2RGB)
                    self.image_roi.setImage(image)
            self.choose_file_button.clicked.connect(choose_file)

            layout.addWidget(self.image_roi)
            layout.addWidget(self.choose_file_button)

            self.setLayout(layout)

        def get_image(self):
            return self.image_roi.selected_image

        def insertImageInROI(self, image):
            if image is None:
                return
```

```
        full_image = self.image_roi.image.copy()
        l, t, r, b = self.image_roi.roi
        if image.ndim == 2:
            image = np.expand_dims(image, axis=-1)
        full_image[t:b, l:r] = image
        self.image_roi.image_item.setImage(full_image.transpose(1, 0, 2))
```

Ui_node.py

```
# -*- coding: utf-8 -*-

# Form implementation generated from reading ui file
'c:\Users\Marselliy\Desktop\analyzer\node.ui'
#
# Created by: PyQt5 UI code generator 5.11.3
#
# WARNING! All changes made in this file will be lost!

from PyQt5 import QtCore, QtGui, QtWidgets


class Ui_Frame(object):
    def setupUi(self, Frame):
        Frame.setObjectName("Frame")
        Frame.resize(413, 557)
        Frame.setStyleSheet("")
        Frame.setFrameShape(QtWidgets.QFrame.StyledPanel)
        Frame.setFrameShadow(QtWidgets.QFrame.Raised)
        self.verticalLayout_2 = QtWidgets.QVBoxLayout(Frame)
        self.verticalLayout_2.setContentsMargins(0, 0, 0, 0)
        self.verticalLayout_2.setSpacing(0)
        self.verticalLayout_2.setObjectName("verticalLayout_2")
        self.verticalLayout = QtWidgets.QVBoxLayout()
        self.verticalLayout.setSpacing(10)
        self.verticalLayout.setObjectName("verticalLayout")
        self.horizontalLayout = QtWidgets.QHBoxLayout()
        self.horizontalLayout.setContentsMargins(6, -1, 6, -1)
        self.horizontalLayout.setObjectName("horizontalLayout")
        self.label = QtWidgets.QLineEdit(Frame)
        self.label.setStyleSheet("background-color: #3c3c3c;\n"
                                 "border: 8px solid #3c3c3c;\n"
                                 "border-top-left-radius: 12px;\n"
                                 "border-top-right-radius: 12px;\n"
                                 "color: \"#FFFFFF\";\n"
                                 "padding: 0px;\n"
                                 "font: 12pt \"Impact\";")
        self.label.setObjectName("label")
        self.horizontalLayout.addWidget(self.label)
        self.verticalLayout.addLayout(self.horizontalLayout)
        self.config_layout = QtWidgets.QVBoxLayout()
        self.config_layout.setObjectName("config_layout")
        self.verticalLayout.addLayout(self.config_layout)
        self.horizontalLayout_3 = QtWidgets.QHBoxLayout()
        self.horizontalLayout_3.setObjectName("horizontalLayout_3")
        self.in_terminals_layout = QtWidgets.QVBoxLayout()
        self.in_terminals_layout.setObjectName("in_terminals_layout")
        self.widget = QtWidgets.QWidget(Frame)
```

```
self.widget.setObjectName("widget")
self.horizontalLayout_5 = QtWidgets.QHBoxLayout(self.widget)
self.horizontalLayout_5.setContentsMargins(0, 0, 0, 0)
self.horizontalLayout_5.setSpacing(8)
self.horizontalLayout_5.setObjectName("horizontalLayout_5")
self.pushButton_3 = QtWidgets.QPushButton(self.widget)
self.pushButton_3.setStyleSheet("* {\n"
                                "    width: 9px;\n"
                                "    height: 9px;\n"
                                "    border-radius: 7px;\n"
                                "    border: 3px solid #3c3c3c;\n"
                                "}\n"
                                "*[occupied=false] {\n"
                                "    background-color: #b5e61d\n"
                                "}\n"
                                "*[occupied=true] {\n"
                                "    background-color: red\n"
                                "}")
self.pushButton_3.setText("")
self.pushButton_3.setProperty("occupied", False)
self.pushButton_3.setObjectName("pushButton_3")
self.horizontalLayout_5.addWidget(self.pushButton_3)
self.label_2 = QtWidgets.QLabel(self.widget)
self.label_2.setObjectName("label_2")
self.horizontalLayout_5.addWidget(self.label_2)
self.in_terminals_layout.addWidget(self.widget)
self.pushButton_2 = QtWidgets.QPushButton(Frame)
sizePolicy = QtWidgets.QSizePolicy(
    QtWidgets.QSizePolicy.Fixed,
    QtWidgets.QSizePolicy.Fixed)
sizePolicy.setHorizontalStretch(0)
sizePolicy.setVerticalStretch(0)
sizePolicy.setHeightForWidth(
    self.pushButton_2.sizePolicy().hasHeightForWidth())
self.pushButton_2.setSizePolicy(sizePolicy)
self.pushButton_2.setStyleSheet("width: 9px;\n"
                                "height: 9px;\n"
                                "border-radius: 7px;\n"
                                "border: 3px solid #3c3c3c;\n"
                                "background-color: #b5e61d")
self.pushButton_2.setText("")
self.pushButton_2.setObjectName("pushButton_2")
self.in_terminals_layout.addWidget(self.pushButton_2)
self.horizontalLayout_3.addLayout(self.in_terminals_layout)
spacerItem = QtWidgets.QSpacerItem(
    40, 20, QtWidgets.QSizePolicy.Expanding, QtWidgets.QSizePolicy.Minimum)
self.horizontalLayout_3.addItem(spacerItem)
spacerItem1 = QtWidgets.QSpacerItem(
    20, 40, QtWidgets.QSizePolicy.Minimum, QtWidgets.QSizePolicy.Expanding)
self.horizontalLayout_3.addItem(spacerItem1)
self.verticalLayout.addLayout(self.horizontalLayout_3)
self.horizontalLayout_2 = QtWidgets.QHBoxLayout()
self.horizontalLayout_2.setSpacing(0)
self.horizontalLayout_2.setObjectName("horizontalLayout_2")
spacerItem2 = QtWidgets.QSpacerItem(
    20, 40, QtWidgets.QSizePolicy.Minimum, QtWidgets.QSizePolicy.Expanding)
```

```
        self.horizontalLayout_2.addItem(spacerItem2)
        spacerItem3 = QtWidgets.QSpacerItem(
            40, 20, QtWidgets.QSizePolicy.Expanding, QtWidgets.QSizePolicy.Minimum)
        self.horizontalLayout_2.addItem(spacerItem3)
        self.out_terminals_layout = QtWidgets.QVBoxLayout()
        self.out_terminals_layout.setObjectName("out_terminals_layout")
        self.pushButton = QtWidgets.QPushButton(Frame)
        self.pushButton.setStyleSheet("width: 9px;\n"
                                      "height: 9px;\n"
                                      "border-radius: 7px;\n"
                                      "border: 3px solid #3c3c3c;\n"
                                      "background-color: #b5e61d")
        self.pushButton.setText("")
        self.pushButton.setObjectName("pushButton")
        self.out_terminals_layout.addWidget(self.pushButton)
        self.horizontalLayout_2.addLayout(self.out_terminals_layout)
        self.verticalLayout.addLayout(self.horizontalLayout_2)
        self.verticalLayout_2.addLayout(self.verticalLayout)

        self.retranslateUi(Frame)
        QtCore.QMetaObject.connectSlotsByName(Frame)

    def retranslateUi(self, Frame):
        _translate = QtCore.QCoreApplication.translate
        Frame.setWindowTitle(_translate("Frame", "Frame"))
        self.label.setText(_translate("Frame", "Test"))
        self.label_2.setText(_translate("Frame", "TextLabel"))

utils.py
from PyQt5 import Qt, QtGui
import numpy as np


def draw_connection(start, end, qp):
    if not isinstance(start, Qt.QPointF):
        start = Qt.QPointF(start)
    if not isinstance(end, Qt.QPointF):
        end = Qt.QPointF(end)

    path = QtGui.QPainterPath(start)
    path.cubicTo(
        Qt.QPointF(start.x() + (end - start).x() / 2, start.y()),
        Qt.QPointF(start.x() + (end - start).x() / 2, end.y()),
        end
    )
    qp.drawPath(path)


def distance_to_connection(p, start, end):
    M = np.array([
        [-1, 3, -3, 1],
        [3, -6, 3, 0],
        [-3, 3, 0, 0],
        [1, 0, 0, 0],
    ])
    P = np.array([
```

```
        [start.x(), start.y()],
        [start.x() + (end - start).x() / 2, start.y()],
        [start.x() + (end - start).x() / 2, end.y()],
        [end.x(), end.y()],
    ])
    t = np.linspace(0, 1, max(int(np.linalg.norm(P[3] - P[0]) / 30), 3))
    pts = np.stack([t ** 3, t ** 2, t, np.ones_like(t)], axis=1) @ M @ P
    return np.linalg.norm(pts - [p.x(), p.y()], axis=1).min()


class bidict(dict):
    def __setitem__(self, key, val):
        dict.__setitem__(self, key, val)
        dict.__setitem__(self, val, key)

    def __delitem__(self, key):
        dict.__delitem__(self, self[key])
        dict.__delitem__(self, key)

core\datatypes.py
class Datatype:
    pass


class Image(Datatype):
    pass


class Image_1C(Image):
    pass


class Image_GRAY(Image_1C):
    pass


class Image_BINARY(Image_GRAY):
    pass


class Image_3C(Image):
    pass


class Image_RGB(Image_3C):
    pass


class Image_HSV(Image_3C):
    pass


class Image_LAB(Image_3C):
    pass
```

```
class Image_YCrCb(Image_3C):
    pass


class Atom(Datatype):
    pass

core\datavalue.py
from PyQt5 import QtCore
from core.datatypes import Image
import numpy as np
from PIL import Image as PImage
from io import BytesIO
import base64


class DataValue(QtCore.QObject):

    valueChanged = QtCore.pyqtSignal(bool)

    @staticmethod
    def from_datatype(dtype):
        if issubclass(dtype, Image):
            return ImageDataValue(dtype)
        return AtomDataValue(dtype)

    def __init__(self, dtype):
        super(DataValue, self).__init__()
        self.data = None
        self.dtype = dtype

    def set_data(self, data):
        self.data = data
        self.valueChanged.emit(self.data is not None)

    def get_data(self):
        return self.data

    def encode(self, compress=True):
        pass

    def decode(self):
        pass


class AtomDataValue(DataValue):
    def encode(self, compress=True):
        return self.data

    def decode(self, buf):
        return buf


class ImageDataValue(DataValue):
    def encode(self, compress=True):
        if self.data is None or self.data['image'] is None:
```

```
            return None
        if not compress:
            return self.data
        pil_image = PImage.fromarray(self.data['image'])
        buffered = BytesIO()
        pil_image.save(buffered, format="PNG")
        img_str = base64.b64encode(buffered.getvalue()).decode('utf-8')
        return dict(
            image=img_str,
            colorspace=self.data['colorspace']
        )

    def decode(self, buf):
        if buf is None:
            return None
        if not isinstance(buf['image'], str):
            return buf
        img_buf = base64.b64decode(buf['image'].encode('utf-8'))
        pil_image = PImage.open(BytesIO(img_buf))
        image = np.array(pil_image)
        return dict(
            image=image,
            colorspace=buf['colorspace']
        )
```

core\edge.py

```
from PyQt5 import QtCore


class Edge(QtCore.QObject):

    removing = QtCore.pyqtSignal()

    def __init__(self, start_node, end_node, start_out_name, end_in_name):
        super(Edge, self).__init__()
        self.start_node = start_node
        self.end_node = end_node
        self.start_out_name = start_out_name
        self.end_in_name = end_in_name

        end_node.connect_to_input(
            end_in_name, start_node.out_values[start_out_name])
        start_node.valueChanged.connect(end_node.update)

    def deleteLater(self):
        self.end_node.disconnect_from_input(self.end_in_name)
        self.start_node.valueChanged.disconnect(self.end_node.update)
        self.removing.emit()
        super(Edge, self).deleteLater()
```

core\graph.py

```
from PyQt5 import QtCore


class Graph(QtCore.QObject):
```

```
    def __init__(self):
        super(Graph, self).__init__()

        self.adj_matrix = dict()

    def add_node(self, node):
        self.adj_matrix[node] = dict()

    def remove_node(self, node):
        deleted_edges = list()

        for node2 in self.adj_matrix[node]:
            deleted_edges.extend(self.adj_matrix[node][node2])
            for edge in self.adj_matrix[node][node2]:
                edge.deleteLater()
        del self.adj_matrix[node]

        for v in self.adj_matrix.values():
            if node in v:
                deleted_edges.extend(v[node])
                for edge in v[node]:
                    edge.deleteLater()
                del v[node]

        return deleted_edges

    def add_edge(self, edge):
        if edge.end_node in self.adj_matrix[edge.start_node]:
            self.adj_matrix[edge.start_node][edge.end_node].append(edge)
        else:
            self.adj_matrix[edge.start_node][edge.end_node] = [edge]

    def remove_edge(self, edge):
        self.adj_matrix[edge.start_node][edge.end_node].remove(edge)
        edge.deleteLater()
```

core\node.py

```
from PyQt5 import QtCore
from core import DataValue


class Node(QtCore.QObject):

    valueChanged = QtCore.pyqtSignal()

    instances = dict()
    """
        in_types - {str name1: DataType type1, ...}
        out_types - {str name1: DataType type1, ...}
        parameters - {str name1: Parameter param1, ...}
    """

    def __init__(
        self,
        in_types=dict(),
        out_types=dict(),
```

```
    out_values_to_return=set(),
    parameters=dict(),
    fn=None,
    all_inputs_required=True
):
    super(Node, self).__init__()

    self.in_types = in_types
    self.out_types = out_types
    self.out_values_to_return = out_values_to_return
    self.in_values = {name: None for name in in_types}
    self.out_values = {
        name: DataValue.from_datatype(_type) for name,
        _type in out_types.items()}
    self.parameters = parameters
    self.fn = fn
    self.all_inputs_required = all_inputs_required

def get_params_for_widget_creation(self):
    return dict(
        in_types=self.in_types,
        out_types=self.out_types,
        parameters=self.parameters
    )

def connect_to_input(self, in_name, out_data):
    if in_name not in self.in_values.keys():
        return

    self.in_values[in_name] = out_data

    self.update()

def disconnect_from_input(self, in_name):
    if in_name not in self.in_values.keys():
        return

    self.in_values[in_name] = None

    self.update()

def ready(self):
    for p in self.parameters.values():
        if p.get_value() is None:
            return False
    if self.all_inputs_required:
        for v in self.in_values.values():
            if v is None:
                return False
            if v.get_data() is None:
                return False
        return True
    else:
        for v in self.in_values.values():
            if (v is not None) and (v.get_data() is not None):
                return True
```

```
            return False

    def update(self):
        if not self.ready():
            for v in self.out_values.values():
                v.set_data(None)
            self.valueChanged.emit()
            return

        if not callable(self.fn):
            return

        data_in = {
            **{k: None if v is None else v.get_data() for k, v in
self.in_values.items()},
            **{k: v.get_value() for k, v in self.parameters.items()}
        }

        data_out = self.fn(data_in)

        for k, v in data_out.items():
            self.out_values[k].set_data(v)

        self.valueChanged.emit()

    def _params_to_serialize(self):
        return self.parameters

    def serialize(self):
        if self.parameters == self._params_to_serialize():
            return {
                k: v.datavalue.encode()
                for k, v in self.parameters.items()
            }
        else:
            return {
                k: (v.datavalue.encode() if k in self._params_to_serialize() else
None)
                for k, v in self.parameters.items()
            }

core\parameter.py
from PyQt5 import QtCore, QtWidgets, QtGui
from core.datatypes import Atom
from core.datavalue import DataValue


class Parameter(QtCore.QObject):

    valueChanged = QtCore.pyqtSignal()

    def __init__(self, name, dtype=Atom):
        super(Parameter, self).__init__()
        self.name = name
        self.datavalue = DataValue.from_datatype(dtype)
        self.datavalue.set_data(None)
```

```
    def _new_control(self):
        pass

    def new_control(self):
        control = self._new_control()
        if control is None:
            return
        frame = QtWidgets.QFrame()
        layout = QtWidgets.QHBoxLayout()
        frame.setLayout(layout)

        label = QtWidgets.QLabel()
        label.setStyleSheet("color: #ffffff;\n"
                            "font: 9pt \"cm sans serif 2012\";"
                            "padding-left: 8px"
                            )
        label.setText(self.name)
        layout.addWidget(label)
        label.setSizePolicy(
            QtGui.QSizePolicy.Fixed,
            QtGui.QSizePolicy.Preferred)

        layout.addWidget(control)

        return frame

    def get_value(self):
        return self.datavalue.get_data()

    def _set_value(self, value):
        self.datavalue.set_data(value)

    def set_value(self, value):
        self._set_value(value)
        self.valueChanged.emit()
```

```
core\__init__.py
from .edge import Edge

from .datavalue import DataValue

from .parameter import Parameter
from .parameters import *

from .node import Node
from .nodes import *

from .datatypes import *


from .graph import Graph
```

```
ui\edge_widget.py
from PyQt5 import QtCore
from utils import draw_connection, distance_to_connection
```

```
from ui import InTerminal, OutTerminal


class EdgeWidget(QtCore.QObject):

    remove_signal = QtCore.pyqtSignal()

    @staticmethod
    def check_terminals(in_terminal, out_terminal):
        if not isinstance(in_terminal, InTerminal) or not isinstance(
                out_terminal, OutTerminal):
            return False
        if in_terminal.connected_edge is not None:
            return False

        return True

    def __init__(self, in_terminal, out_terminal):
        self.start = in_terminal
        self.end = out_terminal
        self.start.add_edges_count(1)
        self.end.add_edges_count(1)
        self.highlight = False

        super(EdgeWidget, self).__init__()

    def draw(self, graph_widget, qp):
        draw_connection(
            graph_widget.mapFromGlobal(self.start.global_pos()),
            graph_widget.mapFromGlobal(self.end.global_pos()),
            qp
        )

    def distance_to_point(self, graph_widget, p):
        return distance_to_connection(
            p,
            graph_widget.mapFromGlobal(self.start.global_pos()),
            graph_widget.mapFromGlobal(self.end.global_pos()),
        )

    def set_highlight(self, v):
        self.highlight = v

    def get_highlight(self):
        return self.highlight

    def deleteLater(self):
        self.remove_signal.emit()
        self.start.add_edges_count(-1)
        self.end.add_edges_count(-1)
        super(EdgeWidget, self).deleteLater()

ui\graph_widget.py
from core.parameters import ImageROIParameter
from core.nodes import str_to_classname, node_tree
from utils import draw_connection, bidict
```

```
from ui import NodeWidget, EdgeWidget, InTerminal, OutTerminal
from PyQt5 import QtWidgets
from PyQt5 import QtCore, QtGui, QtWebSockets
from PyQt5.QtCore import QUrl
import os
import json
import logging
logging.basicConfig(level=logging.WARN)

SWAP_FILE = '.swap.graph'

class GraphWidgetFrame(QtWidgets.QFrame):

    def __init__(self, image_view, *args, local_ws=None,
                 host='127.0.0.1', port=1302, **kwargs):
        super(GraphWidgetFrame, self).__init__(*args, **kwargs)

        QtGui.QFontDatabase.addApplicationFont('fonts/cm sans serif 2012.ttf')

        self.image_view = image_view
        self.is_remote_mode = (local_ws is None)
        if local_ws is None:
            self.ws_client = QtWebSockets.QWebSocket(
                "", QtWebSockets.QWebSocketProtocol.Version13, None)
            self.ws_client.error.connect(self.error)
            self.ws_client.open(QUrl("ws://%s:%d" % (host, port)))
        else:
            self.ws_client = local_ws
        self.ws_client.textMessageReceived.connect(self.process_action)

        self.uuid_object_map = bidict()

        self.nodes = list()
        self.edges = list()

        self.active_edge_pen = QtGui.QPen()
        self.active_edge_pen.setColor(QtGui.QColor('#3c3c3c'))
        self.active_edge_pen.setCapStyle(QtCore.Qt.RoundCap)
        self.active_edge_pen.setWidth(4)

        self.inactive_edge_pen = QtGui.QPen()
        self.inactive_edge_pen.setColor(QtGui.QColor('#3c3c3c'))
        self.inactive_edge_pen.setStyle(QtCore.Qt.PenStyle.DotLine)
        self.inactive_edge_pen.setCapStyle(QtCore.Qt.RoundCap)
        self.inactive_edge_pen.setWidth(4)

        self.hightlight_edge_pen = QtGui.QPen()
        self.hightlight_edge_pen.setColor(QtGui.QColor('#fff'))
        self.hightlight_edge_pen.setCapStyle(QtCore.Qt.RoundCap)
        self.hightlight_edge_pen.setWidth(8)

        self.bg_brush = QtGui.QBrush(QtGui.QColor('#8c8c8c'))
        self.none_brush = QtGui.QBrush(QtGui.QColor('#00000000'))
        self.small_bg_grid = QtGui.QPen()
        self.small_bg_grid.setColor(QtGui.QColor('#777'))
        self.small_bg_grid.setWidth(2)
```

```
        self.large_bg_grid = QtGui.QPen()
        self.large_bg_grid.setColor(QtGui.QColor('#525252'))
        self.large_bg_grid.setWidth(2)

        self.last_clicked_terminal = None
        self.hovered_edge = None
        self.last_mouse_pos = QtCore.QPointF(0, 0)

        self.pan = QtCore.QPointF(0, 0)

        self.installEventFilter(self)

        # Add custom menu with node selection
        self.popMenu = QtWidgets.QMenu(self)
        menu_tree = {'': self.popMenu}
        for v in node_tree.values():
            key = ''
            for category in v:
                next_key = key + category
                if next_key not in menu_tree:
                    menu_tree[next_key] = menu_tree[key].addMenu(category)
                key = next_key

        for nodename in str_to_classname:
            menu_tree[''.join(node_tree[nodename])].addAction(nodename)
        self.popMenu.triggered.connect(self.on_context_menu_add_clicked)

        self.setContextMenuPolicy(QtCore.Qt.CustomContextMenu)
        self.customContextMenuRequested.connect(self.on_context_menu)

        # self.setStyleSheet('background-color: "#777";')

        self.setMouseTracking(True)

        if os.path.exists(SWAP_FILE):
            self.restore(SWAP_FILE)

    def error(self, error_code):
        print("error code: {}".format(error_code))
        print(self.ws_client.errorString())

    def on_context_menu(self, point):
        # Check if user wanted to remove edge
        if (self.hovered_edge is not None) and self.hovered_edge in
self.uuid_object_map:
            edge_id = self.uuid_object_map[self.hovered_edge]
            message = dict(
                action='remove_edge',
                edge_id=edge_id
            )
            if self.is_remote_mode:
                message = json.dumps(message)
            self.ws_client.sendTextMessage(message)
        else:
            self.popMenu.exec_(self.mapToGlobal(point))
```

```
def on_context_menu_add_clicked(self, sender):
    node_classname = sender.text()

    message = dict(
        action='add_node',
        node_classname=node_classname
    )
    if self.is_remote_mode:
        message = json.dumps(message)
    self.ws_client.sendTextMessage(message)

def process_action(self, message):
    if isinstance(message, str):
        message = json.loads(message)
    action = message['action']
    if action == 'add_node':
        self.add_node_widget(message)
    elif action == 'add_edge':
        self.add_edge(message)
    elif action == 'remove':
        self.remove_widgets(message)
    elif action == 'update_values':
        self.update_values(message)
    elif action == 'serialize':
        self.serialize(message)

    logging.info(list(self.uuid_object_map.keys()))

def add_node_widget(self, message):
    node_params = str_to_classname[message['node_classname']](
    ).get_params_for_widget_creation()
    node_params['name'] = message['node_classname']
    node_widget = NodeWidget(**node_params)
    uuid = message['node_id']

    self.uuid_object_map[uuid] = node_widget

    if 'geometry' in message:
        geometry = message['geometry']
        node_widget.setGeometry(*geometry)
    else:
        node_widget.move(self.mapFromGlobal(self.popMenu.pos()))

    node_widget.setParent(self)
    node_widget.show()

    node_widget.remove_signal.connect(self.on_node_remove)
    node_widget.paramChanged.connect(self.on_param_changed)
    for t in node_widget.in_terminals.values():
        t.clicked.connect(self.on_terminal_clicked)
    for t in node_widget.out_terminals.values():
        t.clicked.connect(self.on_terminal_clicked)

    if 'parameters' in message:
        for name, value in message['parameters'].items():
            param = node_widget.parameters[name]
```

```
                param.set_value(param.datavalue.decode(value))

        for param in node_widget.parameters.values():
            if isinstance(param, ImageROIParameter):
                param.assign_control(self.image_view)

        # Special case for Show node
        def update_image_view():
            value = node_widget.out_values['-'].get_data()
            if value is None:
                return
            self.image_view.insertImageInROI(value['image'])

        if message['node_classname'] == 'Show':
            node_widget.out_values['-'].valueChanged.connect(update_image_view)

    def remove_widgets(self, message):
        objects_to_remove = [self.uuid_object_map[e]
                             for e in message['node_ids'] + message['edge_ids']]
        for obj in objects_to_remove:
            del self.uuid_object_map[obj]
            obj.setParent(None)
            obj.deleteLater()

    def add_edge(self, message):

        start_node = self.uuid_object_map[message['start_node_id']]
        end_node = self.uuid_object_map[message['end_node_id']]

        edge_id = message['edge_id']
        edge = EdgeWidget(
            start_node.out_terminals[message['start_out_name']],
end_node.in_terminals[message['end_in_name']])
        self.uuid_object_map[edge_id] = edge

    def update_values(self, message):
        node = self.uuid_object_map[message['node_id']]
        for k, v in message['values'].items():
            data = node.out_values[k].decode(v)
            node.out_values[k].set_data(data)
        self.update()

    @QtCore.pyqtSlot()
    def on_terminal_clicked(self):
        # If first click
        if self.last_clicked_terminal is None:
            self.last_clicked_terminal = self.sender()
        else:
            if isinstance(self.sender(), InTerminal) and isinstance(
                    self.last_clicked_terminal, OutTerminal):
                start_node, end_node = self.last_clicked_terminal.parent_node,
self.sender().parent_node
                start_out_name, end_in_name = self.last_clicked_terminal.name,
self.sender().name
            elif isinstance(self.sender(), OutTerminal) and
isinstance(self.last_clicked_terminal, InTerminal):
```

```
                end_node, start_node = self.last_clicked_terminal.parent_node,
self.sender().parent_node
                end_in_name, start_out_name = self.last_clicked_terminal.name,
self.sender().name
            else:
                self.last_clicked_terminal = None
                return

            start_node_id = self.uuid_object_map[start_node]
            end_node_id = self.uuid_object_map[end_node]

            message = dict(
                action='add_edge',
                start_node_id=start_node_id,
                end_node_id=end_node_id,
                start_out_name=start_out_name,
                end_in_name=end_in_name
            )
            if self.is_remote_mode:
                message = json.dumps(message)
            self.ws_client.sendTextMessage(message)

            self.last_clicked_terminal = None

    @QtCore.pyqtSlot()
    def on_node_remove(self):
        node = self.sender()
        node_id = self.uuid_object_map[node]

        message = dict(
            action='remove_node',
            node_id=node_id
        )
        if self.is_remote_mode:
            message = json.dumps(message)
        self.ws_client.sendTextMessage(message)

    @QtCore.pyqtSlot()
    def on_edge_remove(self):
        edge = self.sender()
        edge.end.parent_node.value_changed.disconnect(
            edge.start.parent_node.recalculate)

        edge.start.connected_edge = None
        edge.end.connected_edges.remove(edge)
        self.edges.remove(edge)

    def on_param_changed(self, param_name):
        node = self.sender()
        node_id = self.uuid_object_map[node]

        value = node.parameters[param_name].datavalue.encode(
            compress=self.is_remote_mode)
        message = dict(
            action='change_param',
            node_id=node_id,
```

```
            param_name=param_name,
            value=value
        )
        if self.is_remote_mode:
            message = json.dumps(message)
        self.ws_client.sendTextMessage(message)

    def mousePressEvent(self, event):
        self.last_clicked_terminal = None
        self.update()

    def paintEvent(self, event):

        # Setup painter
        qp = QtGui.QPainter()
        qp.begin(self)
        qp.setRenderHint(QtGui.QPainter.Antialiasing)

        # Draw background grid
        grid_step = 35
        qp.setPen(self.small_bg_grid)
        qp.setBrush(self.bg_brush)
        w, h = self.width(), self.height()
        qp.drawRect(0, 0, w, h)
        for x in range(int(self.pan.x()) % grid_step, w, grid_step):
            qp.drawLine(x, 0, x, h)
        for y in range(int(self.pan.y()) % grid_step, h, grid_step):
            qp.drawLine(0, y, w, y)
        qp.setPen(self.large_bg_grid)
        w, h = self.width(), self.height()
        for x in range(int(self.pan.x()) %
                       (grid_step * 10), w, grid_step * 10):
            qp.drawLine(x, 0, x, h)
        for y in range(int(self.pan.y()) %
                       (grid_step * 10), h, grid_step * 10):
            qp.drawLine(0, y, w, y)

        # Draw edges
        qp.setPen(self.active_edge_pen)
        qp.setBrush(self.none_brush)
        for edge in self.uuid_object_map.keys():
            if isinstance(edge, EdgeWidget):
                if edge.get_highlight():
                    qp.setPen(self.hightlight_edge_pen)
                    edge.draw(self, qp)
                    qp.setPen(self.active_edge_pen)
                if edge.start.parent_node.out_values[edge.start.name].get_data(
                ) is None:
                    qp.setPen(self.inactive_edge_pen)

                edge.draw(self, qp)

        # Draw connection edge
        if self.last_clicked_terminal is not None:
            draw_connection(
                self.mapFromGlobal(
```

```
                self.last_clicked_terminal.global_pos()
            ),
            QtCore.QPoint(int(self.last_mouse_pos.x()),
                          int(self.last_mouse_pos.y())),
            qp
        )

    qp.end()

    # Workaround to prevent recursion on updates
    if self.geometry().width() != event.rect().width(
    ) or self.geometry().height() != event.rect().height():
        self.update()

def mouseMoveEvent(self, event):
    update = False

    if event.buttons() == QtCore.Qt.LeftButton:
        cur_pos = event.localPos()
        cur_pos = QtCore.QPoint(int(cur_pos.x()), int(cur_pos.y()))
        shift = self.last_mouse_pos - cur_pos
        nodes = [
            e for e in self.uuid_object_map if isinstance(
                e, NodeWidget)]
        pan = all([not node.geometry().contains(cur_pos)
                   for node in nodes])
        if pan:
            self.pan -= shift
            for node in nodes:
                if not isinstance(node, NodeWidget):
                    continue
                new_pos = node.pos() - shift
                node.move(QtCore.QPoint(
                    int(new_pos.x()), int(new_pos.y())))
            update = True

    self.last_mouse_pos = event.localPos()
    if self.last_clicked_terminal is not None:
        update = True

    # Check if user wanted to delete edge
    edges = [e for e in self.uuid_object_map if isinstance(e, EdgeWidget)]
    self.hovered_edge = None
    if len(edges) > 0:
        distances = [
            edge.distance_to_point(
                self, self.last_mouse_pos) for edge in edges]
        min_idx = distances.index(min(distances))
        if distances[min_idx] < 30:
            self.hovered_edge = edges[min_idx]
        for i, edge in enumerate(edges):
            if i == min_idx:
                if (distances[min_idx] < 30) and (
                        not edge.get_highlight()):
                    update = True
                    edge.set_highlight(True)
```

```
                    elif (distances[min_idx] > 30) and edge.get_highlight():
                        update = True
                        edge.set_highlight(False)
                elif i != min_idx and edge.get_highlight():
                    update = True
                    edge.set_highlight(False)

        if update:
            self.update()
        super(GraphWidgetFrame, self).mouseMoveEvent(event)

    def request_serialization(self, save_filename=None):
        message = dict(
            action='serialize',
            save_filename=save_filename
        )
        if self.is_remote_mode:
            message = json.dumps(message)
        self.ws_client.sendTextMessage(message)

    def serialize(self, message):
        serialized = message['serialized']
        serialized['pan'] = [self.pan.x(), self.pan.y()]
        for node in serialized['nodes']:
            geometry = self.uuid_object_map[node['node_id']].geometry()
            node['geometry'] = [geometry.x(), geometry.y(), geometry.width(),
geometry.height()]

        save_filename = message['save_filename']
        if save_filename is None:
            save_filename, _ =
QtWidgets.QFileDialog.getSaveFileName(filter='graph(*.graph)')
        if save_filename is '':
            return
        if not save_filename.endswith('.graph'):
            save_filename = save_filename + '.graph'

        with open(save_filename, 'w') as f:
            json.dump(serialized, f)

    def clear(self):
        nodes_to_remove = [node_id for node_id, v in self.uuid_object_map.items() if
isinstance(v, NodeWidget)]
        for node_id in nodes_to_remove:
            message = dict(
                action='remove_node',
                node_id=node_id
            )
            if self.is_remote_mode:
                message = json.dumps(message)
            self.ws_client.sendTextMessage(message)

        self.pan.setX(0)
        self.pan.setY(0)

    def restore(self, load_filename=None):
```

```
        if load_filename is None:
            load_filename, _ =
QtWidgets.QFileDialog.getOpenFileName(filter='graph(*.graph)')
        if not os.path.exists(load_filename):
            return

        self.clear()

        with open(load_filename, 'r') as f:
            serialized = json.load(f)

        self.pan.setX(serialized['pan'][0])
        self.pan.setY(serialized['pan'][1])

        for node in serialized['nodes']:
            message = node
            message['action'] = 'add_node'
            if self.is_remote_mode:
                message = json.dumps(message)
            self.ws_client.sendTextMessage(message)

        for edge in serialized['edges']:
            message = edge
            message['action'] = 'add_edge'
            if self.is_remote_mode:
                message = json.dumps(message)
            self.ws_client.sendTextMessage(message)


class GraphWidget(QtWidgets.QMainWindow):
    def __init__(self, image_view, *args, local_ws=None,
                 host='127.0.0.1', port=1302, **kwargs):
        super(GraphWidget, self).__init__()

        self.graph_widget_frame = GraphWidgetFrame(
            image_view, *args, local_ws=local_ws,
            host=host, port=port, **kwargs
        )

        self.setCentralWidget(self.graph_widget_frame)

        menubar = self.menuBar()
        file_menu = menubar.addMenu('File')

        new_graph_action = QtGui.QAction('New', self)
        new_graph_action.triggered.connect(self.new)
        file_menu.addAction(new_graph_action)

        save_graph_action = QtGui.QAction('Save', self)
        save_graph_action.triggered.connect(self.save)
        file_menu.addAction(save_graph_action)

        load_graph_action = QtGui.QAction('Load', self)
        load_graph_action.triggered.connect(self.load)
        file_menu.addAction(load_graph_action)
```

```
    def closeEvent(self, event):
        self.graph_widget_frame.request_serialization(SWAP_FILE)

    def new(self):
        self.graph_widget_frame.clear()

    def save(self):
        self.graph_widget_frame.request_serialization()

    def load(self):
        self.graph_widget_frame.restore()
```

ui\node_widget.py

```
from PyQt5 import QtWidgets
from PyQt5 import QtGui, QtCore

from ui import EmptyTerminal, InTerminal, OutTerminal
from core import DataValue


class NodeWidget(QtWidgets.QWidget):

    remove_signal = QtCore.pyqtSignal()
    paramChanged = QtCore.pyqtSignal(str)

    def __init__(self, name, in_types=dict(), out_types=dict(),
                 parameters=dict(), *args, **kwargs):
        super(NodeWidget, self).__init__(*args, **kwargs)
        self.name = name
        self.parameters = parameters

        self.setup_ui()

        self.label.setText(self.name)

        def _cb(name):
            return lambda: self.paramChanged.emit(name)

        for param in self.parameters.values():
            new_control = param.new_control()
            if new_control is not None:
                self.config_layout.addWidget(new_control)
            param.valueChanged.connect(_cb(param.name))

        self.__mousePressPos = None

        self.in_terminals = in_types
        self.out_terminals = out_types
        self.out_values = {
            k: DataValue.from_datatype(v) for k,
            v in out_types.items()}

        for key, dtype in self.in_terminals.items():
            if isinstance(self.in_terminals[key], EmptyTerminal):
                in_terminal = self.in_terminals[key]
```

```
            else:
                in_terminal = InTerminal(key, dtype, self)
                self.in_terminals[key] = in_terminal
            self.in_terminals_layout.addWidget(in_terminal)
        for key, dtype in self.out_terminals.items():
            if isinstance(self.out_terminals[key], EmptyTerminal):
                out_terminal = self.out_terminals[key]
            else:
                out_terminal = OutTerminal(key, dtype, self)
            self.out_terminals[key] = out_terminal
            self.out_terminals_layout.addWidget(out_terminal)

        if '' in self.in_terminals:
            del self.in_terminals['']
        if '' in self.out_terminals:
            del self.out_terminals['']

        if 'in_image' in self.in_terminals:
            def update_in_image():
                image = self.in_terminals['in_image'].value
                self.config_inp_vis.setImage(image.transpose(1, 0, 2))

            self.value_changed.connect(update_in_image)
        if 'out_image' in self.out_terminals:
            def update_out_image():
                image = self.out_terminals['out_image'].value
                self.config_out_vis.setImage(image.transpose(1, 0, 2))

            self.value_changed.connect(update_out_image)

    def setup_ui(self):
        self.verticalLayout_2 = QtWidgets.QVBoxLayout(self)
        self.verticalLayout_2.setContentsMargins(0, 0, 0, 0)
        self.verticalLayout_2.setSpacing(0)
        self.verticalLayout_2.setObjectName("verticalLayout_2")
        self.verticalLayout = QtWidgets.QVBoxLayout()
        self.verticalLayout.setSpacing(10)
        self.verticalLayout.setObjectName("verticalLayout")
        self.horizontalLayout = QtWidgets.QHBoxLayout()
        self.horizontalLayout.setContentsMargins(6, -1, 6, -1)
        self.horizontalLayout.setObjectName("horizontalLayout")
        self.label = QtWidgets.QLineEdit(self)
        self.label.setStyleSheet("background-color: #3c3c3c;\n"
                                 "border: 8px solid #3c3c3c;\n"
                                 "border-top-left-radius: 12px;\n"
                                 "border-top-right-radius: 12px;\n"
                                 "color: \"#FFFFFF\";\n"
                                 "padding: 0px;\n"
                                 "font: 10pt \"cm sans serif 2012\";")
        self.label.setObjectName("label")
        self.horizontalLayout.addWidget(self.label)
        self.verticalLayout.addLayout(self.horizontalLayout)
        self.config_layout = QtWidgets.QVBoxLayout()
        self.config_layout.setObjectName("config_layout")
        self.verticalLayout.addLayout(self.config_layout)
        self.horizontalLayout_3 = QtWidgets.QHBoxLayout()
```

```
        self.horizontalLayout_3.setObjectName("horizontalLayout_3")
        self.in_terminals_layout = QtWidgets.QVBoxLayout()
        self.in_terminals_layout.setObjectName("in_terminals_layout")
        self.in_terminals_layout.setSpacing(8)
        self.horizontalLayout_3.addLayout(self.in_terminals_layout)
        spacerItem = QtWidgets.QSpacerItem(
            40, 20, QtWidgets.QSizePolicy.Expanding, QtWidgets.QSizePolicy.Minimum)
        self.horizontalLayout_3.addItem(spacerItem)
        spacerItem1 = QtWidgets.QSpacerItem(
            20, 40, QtWidgets.QSizePolicy.Minimum, QtWidgets.QSizePolicy.Expanding)
        self.horizontalLayout_3.addItem(spacerItem1)
        self.verticalLayout.addLayout(self.horizontalLayout_3)
        self.horizontalLayout_2 = QtWidgets.QHBoxLayout()
        self.horizontalLayout_2.setSpacing(0)
        self.horizontalLayout_2.setObjectName("horizontalLayout_2")
        spacerItem2 = QtWidgets.QSpacerItem(
            20, 40, QtWidgets.QSizePolicy.Minimum, QtWidgets.QSizePolicy.Expanding)
        self.horizontalLayout_2.addItem(spacerItem2)
        spacerItem3 = QtWidgets.QSpacerItem(
            40, 20, QtWidgets.QSizePolicy.Expanding, QtWidgets.QSizePolicy.Minimum)
        self.horizontalLayout_2.addItem(spacerItem3)
        self.out_terminals_layout = QtWidgets.QVBoxLayout()
        self.out_terminals_layout.setObjectName("out_terminals_layout")
        self.out_terminals_layout.setSpacing(8)
        self.horizontalLayout_2.addLayout(self.out_terminals_layout)
        self.verticalLayout.addLayout(self.horizontalLayout_2)
        self.verticalLayout_2.addLayout(self.verticalLayout)

        self.border_pen = QtGui.QPen(QtGui.QColor('#3c3c3c'))
        self.border_pen.setWidth(3)
        self.border_pen.setJoinStyle(QtCore.Qt.MiterJoin)
        self.background_brush = QtGui.QBrush(QtGui.QColor('#787878'))

    def paintEvent(self, event):
        qp = QtGui.QPainter()
        qp.begin(self)

        qp.setPen(self.border_pen)
        qp.setBrush(self.background_brush)
        w, h = self.width(), self.height()

        qp.drawRect(7, 7, w - 15, h - 16)

    def mousePressEvent(self, event):
        self.__mousePressPos = None
        self.__mouseMovePos = None
        if event.button() == QtCore.Qt.LeftButton:
            self.__mousePressPos = event.globalPos()
            self.__mouseMovePos = event.globalPos()
        elif event.button() == QtCore.Qt.RightButton:
            self.remove()

        super(NodeWidget, self).mousePressEvent(event)

    def mouseMoveEvent(self, event):
```

```
        if event.buttons() == QtCore.Qt.LeftButton and self.__mousePressPos is not
None:
            # adjust offset from clicked point to origin of widget
            currPos = self.mapToGlobal(self.pos())
            globalPos = event.globalPos()
            diff = globalPos - self.__mouseMovePos
            newPos = self.mapFromGlobal(currPos + diff)
            self.move(newPos)

            self.__mouseMovePos = globalPos

        super(NodeWidget, self).mouseMoveEvent(event)

    def mouseReleaseEvent(self, event):
        if self.__mousePressPos is not None:
            moved = event.globalPos() - self.__mousePressPos
            if moved.manhattanLength() > 3:
                event.ignore()
                return

        super(NodeWidget, self).mouseReleaseEvent(event)

    def remove(self):
        self.remove_signal.emit()
```

ui\terminal.py

```
from PyQt5 import QtWidgets
from PyQt5 import QtCore
from core.datatypes import (
    Image,
    Image_1C,
    Image_GRAY,
    Image_3C,
    Image_RGB,
    Image_HSV,
    Image_LAB,
    Image_YCrCb,
    Image_BINARY
)

dtype_to_color = {
    Image: (29, 181, 230),
    Image_1C: (36, 36, 36),
    Image_GRAY: (230, 181, 29),
    Image_3C: (230, 230, 230),
    Image_RGB: (181, 230, 29),
    Image_HSV: (181, 230, 29),
    Image_LAB: (181, 230, 29),
    Image_YCrCb: (181, 230, 29),
    Image_BINARY: (237, 28, 42)
}


class TerminalButton(QtWidgets.QPushButton):

    def __init__(self):
```

```
        super(TerminalButton, self).__init__()

        self.setSizePolicy(
            QtWidgets.QSizePolicy.Fixed,
            QtWidgets.QSizePolicy.Fixed
        )

    def global_pos(self):
        return self.mapToGlobal(
            self.mapFromParent(
                self.pos() + QtCore.QPoint(
                    self.width() // 2,
                    self.height() // 2
                )))


class Terminal(QtWidgets.QWidget):

    clicked = QtCore.pyqtSignal()

    def __init__(self, name, dtype, parent_node, button_left, *args, **kwargs):
        super(Terminal, self).__init__(*args, **kwargs)
        self.name = name
        self.parent_node = parent_node
        self.value = None
        self.edges = 0

        layout = QtWidgets.QHBoxLayout()
        layout.setContentsMargins(0, 0, 0, 0)
        self.setLayout(layout)

        self.button = TerminalButton()
        self.button.clicked.connect(self.clicked)
        self.label = QtWidgets.QLabel()
        self.label.setText(self.name)
        if button_left:
            layout.addWidget(self.label)
            layout.addWidget(self.button)
            self.label.setAlignment(
                QtCore.Qt.AlignRight | QtCore.Qt.AlignVCenter
            )
        else:
            layout.addWidget(self.button)
            layout.addWidget(self.label)
            self.label.setAlignment(
                QtCore.Qt.AlignLeft | QtCore.Qt.AlignVCenter
            )

        stylesheet = """
        QPushButton {
            width: 11px;
            height: 11px;
            border-radius: 8px;
            border: 3px solid #3c3c3c;
            background-color: <button_color>;
        }
```

```
        QPushButton[occupied="true"] {
            background-color: qradialgradient(spread:pad, cx:0.5, cy:0.5, radius:0.5,
fx:0.5, fy:0.500455, stop:0.33 rgba(60, 60, 60, 255), stop:0.66 <button_color>)
        }
        QPushButton:hover:!pressed {
            border: 2px solid #ffffff;
        }
        QPushButton[occupied="false"] {
            background-color: <button_color>
        }
        """.replace(
            '<button_color>',
            'rgba(%d, %d, %d, 255)' % dtype_to_color[dtype]
        )
        self.button.setStyleSheet(stylesheet)
        self.label.setStyleSheet("""
            color: #ffffff;
            font: 8pt \"cm sans serif 2012\"
            """)

    def resizeEvent(self, event):
        QtWidgets.QWidget.resizeEvent(self, event)

    def global_pos(self):
        return self.button.global_pos()

    def add_edges_count(self, v):
        self.edges += v
        if self.edges == 0:
            self.button.setProperty('occupied', False)
        else:
            self.button.setProperty('occupied', True)
        self.button.style().unpolish(self.button)
        self.button.style().polish(self.button)
        self.button.update()


class EmptyTerminal(Terminal):
    def __init__(self, parent_node):
        super(EmptyTerminal, self).__init__(
            name='',
            dtype=Image_3C,
            parent_node=parent_node,
            button_left=False
        )

        self.setStyleSheet('background-color: "#00000000"')
        self.setEnabled(False)


class InTerminal(Terminal):
    def __init__(self, *args, **kwargs):
        super(InTerminal, self).__init__(*args, button_left=False, **kwargs)

        self.connected_edge = None
```

```
    def remove_edges(self):
        if self.connected_edge is not None:
            self.connected_edge.remove()
        super(InTerminal, self).remove_edges()


class OutTerminal(Terminal):
    def __init__(self, *args, **kwargs):
        super(OutTerminal, self).__init__(*args, button_left=True, **kwargs)

        self.connected_edges = list()

    def remove_edges(self):
        for edge in self.connected_edges:
            edge.remove()
        super(OutTerminal, self).remove_edges()
```

ui\utils.py

```
from PyQt5 import Qt, QtGui
import numpy as np


def draw_connection(start, end, qp):
    if not isinstance(start, Qt.QPointF):
        start = Qt.QPointF(start)
    if not isinstance(end, Qt.QPointF):
        end = Qt.QPointF(end)

    path = QtGui.QPainterPath(start)
    path.cubicTo(
        Qt.QPointF(start.x() + (end - start).x() / 2, start.y()),
        Qt.QPointF(start.x() + (end - start).x() / 2, end.y()),
        end
    )
    qp.drawPath(path)


def distance_to_connection(p, start, end):
    M = np.array([
        [-1, 3, -3, 1],
        [3, -6, 3, 0],
        [-3, 3, 0, 0],
        [1, 0, 0, 0],
    ])
    P = np.array([
        [start.x(), start.y()],
        [start.x() + (end - start).x() / 2, start.y()],
        [start.x() + (end - start).x() / 2, end.y()],
        [end.x(), end.y()],
    ])
    t = np.linspace(0, 1, max(int(np.linalg.norm(P[3] - P[0]) / 30), 3))
    pts = np.stack([t ** 3, t ** 2, t, np.ones_like(t)], axis=1) @ M @ P
    return np.linalg.norm(pts - [p.x(), p.y()], axis=1).min()


class bidict(dict):
```

```
    def __setitem__(self, key, val):
        dict.__setitem__(self, key, val)
        dict.__setitem__(self, val, key)

    def __delitem__(self, key):
        dict.__delitem__(self, self[key])
        dict.__delitem__(self, key)
```

ui__init__.py

```
from .terminal import *
from .node_widget import NodeWidget
from .edge_widget import EdgeWidget
from .graph_widget import GraphWidget
```

core\nodes\alpha_blending.py

```
from core import Node
from core.datatypes import Image
from core.parameters import FloatParameter

import numpy as np
import cv2


def blend(data_in):
    in_image_1 = data_in['First image']['image']
    in_image_2 = data_in['Second image']['image']
    alpha = data_in['Alpha']

    if in_image_1.ndim == 2 and in_image_2.ndim == 3:
        in_image_1 = np.expand_dims(in_image_1, axis=-1)
    elif in_image_1.ndim == 3 and in_image_2.ndim == 2:
        in_image_2 = np.expand_dims(in_image_2, axis=-1)

    return {'Output image': dict(image=(in_image_1 * alpha + in_image_2 * (
        1 - alpha)).astype(np.uint8), colorspace=data_in['First
image']['colorspace'])}


class AlphaBlending(Node):

    def __init__(self):
        super(AlphaBlending, self).__init__(
            in_types={'First image': Image, 'Second image': Image},
            out_types={'Output image': Image},
            parameters={'Alpha': FloatParameter('Alpha', 0.5, 0, 1)},
            fn=blend
        )
```

core\nodes\brightness_normalization.py

```
from core import Node
from core.datatypes import Image_GRAY
from core.parameters import IntParameter

import numpy as np
import cv2
```

```
def brightness_normalization(data_in):
    in_image = data_in['Input image']['image']
    kernel_size = data_in['Kernel size']
    if kernel_size % 2 == 0:
        kernel_size -= 1

    brightness_mask = cv2.GaussianBlur(in_image, (kernel_size, kernel_size), 0)
    out_image = cv2.subtract(in_image.astype(np.float32),
brightness_mask.astype(np.float32))
    m = out_image.min()
    out_image = (255 * (out_image - m) / (out_image.max() - m)).astype(np.uint8)

    return {
        'Brightness mask': dict(image=brightness_mask, colorspace=data_in['Input
image']['colorspace']),
        'Output image': dict(image=out_image, colorspace=data_in['Input
image']['colorspace'])
    }


class BrightnessNormalization(Node):

    def __init__(self):
        super(BrightnessNormalization, self).__init__(
            in_types={'Input image': Image_GRAY},
            out_types={
                'Brightness mask': Image_GRAY,
                'Output image': Image_GRAY
            },
            parameters={
                'Kernel size': IntParameter(
                    'Kernel size', 49, 3, 255, 2)},
            fn=brightness_normalization
        )

core\nodes\change_colorspace.py
from core import Node
from core.datatypes import *

import numpy as np
import cv2

# Get fn dict for changing colorspaces
colorspaces = ['RGB', 'HSV', 'LAB', 'YCrCb']


def idn(x): return x.copy()


change_fns = dict(
    RGB=dict(
        RGB=idn,
        HSV=lambda x: cv2.cvtColor(x, cv2.COLOR_RGB2HSV),
        LAB=lambda x: cv2.cvtColor(x, cv2.COLOR_RGB2LAB),
        YCrCb=lambda x: cv2.cvtColor(x, cv2.COLOR_RGB2YCrCb),
```

```
    ),
    HSV=dict(
        RGB=lambda x: cv2.cvtColor(x, cv2.COLOR_HSV2RGB),
        HSV=idn,
        LAB=lambda x: cv2.cvtColor(
            cv2.cvtColor(
                x,
                cv2.COLOR_HSV2RGB),
            cv2.COLOR_RGB2LAB),
        YCrCb=lambda x: cv2.cvtColor(
            cv2.cvtColor(
                x,
                cv2.COLOR_HSV2RGB),
            cv2.COLOR_RGB2YCrCb),
    ),
    LAB=dict(
        RGB=lambda x: cv2.cvtColor(x, cv2.COLOR_LAB2RGB),
        HSV=lambda x: cv2.cvtColor(
            cv2.cvtColor(
                x,
                cv2.COLOR_LAB2RGB),
            cv2.COLOR_RGB2HSV),
        LAB=idn,
        YCrCb=lambda x: cv2.cvtColor(
            cv2.cvtColor(
                x,
                cv2.COLOR_LAB2RGB),
            cv2.COLOR_RGB2YCrCb),
    ),
    YCrCb=dict(
        RGB=lambda x: cv2.cvtColor(x, cv2.COLOR_YCrCb2RGB),
        HSV=lambda x: cv2.cvtColor(
            cv2.cvtColor(
                x,
                cv2.COLOR_YCrCb2RGB),
            cv2.COLOR_RGB2HSV),
        LAB=lambda x: cv2.cvtColor(
            cv2.cvtColor(
                x,
                cv2.COLOR_YCrCb2RGB),
            cv2.COLOR_RGB2LAB),
        YCrCb=idn,
    ),
)


def change_colorspace(data_in):
    image = data_in['Color image']['image']
    c1 = data_in['Color image']['colorspace']
    return {
        c2: dict(image=change_fns[c1][c2](image), colorspace=c2)
        for c2 in colorspaces
    }


class ChangeColorspace(Node):
```

```
    def __init__(self):
        super(ChangeColorspace, self).__init__(
            in_types={
                'Color image': Image_3C
            },
            out_types={
                'RGB': Image_RGB,
                'HSV': Image_HSV,
                'LAB': Image_LAB,
                'YCrCb': Image_YCrCb,
            },
            fn=change_colorspace,
            all_inputs_required=False
        )
```

```
core\nodes\compose.py
from core import Node
from core.datatypes import Image_RGB, Image_HSV, Image_LAB, Image_YCrCb, Image_GRAY
from core.parameters import IntParameter

import numpy as np
import cv2


def compose(data_in):
    channel_1 = data_in['Channel 1']['image']
    channel_2 = data_in['Channel 2']['image']
    channel_3 = data_in['Channel 3']['image']
    try:
        out_image = np.stack([channel_1, channel_2, channel_3], axis=-1)
    except ValueError:
        return {
            'RGB': None,
            'HSV': None,
            'LAB': None,
            'YCrCb': None
        }

    return {
        'RGB': dict(image=out_image, colorspace='RGB'),
        'HSV': dict(image=out_image, colorspace='HSV'),
        'LAB': dict(image=out_image, colorspace='LAB'),
        'YCrCb': dict(image=out_image, colorspace='YCrCb')
    }


class Compose(Node):

    def __init__(self):
        super(Compose, self).__init__(
            in_types={
                'Channel 1': Image_GRAY,
                'Channel 2': Image_GRAY,
                'Channel 3': Image_GRAY
            },
```

```
            out_types={
                'RGB': Image_RGB,
                'HSV': Image_HSV,
                'LAB': Image_LAB,
                'YCrCb': Image_YCrCb,
            },
            fn=compose
        )

core\nodes\gamma_contrast.py
from core import Node
from core.datatypes import Image
from core.parameters import FloatParameter

import numpy as np


def gamma_contrast(data_in):
    in_image = data_in['Input image']['image']
    gamma = data_in['Gamma']

    return {'Output image': dict(image=(np.power(in_image / 255, gamma) *
255).astype(
        np.uint8), colorspace=data_in['Input image']['colorspace'])}


class GammaContrast(Node):

    def __init__(self):
        super(GammaContrast, self).__init__(
            in_types={'Input image': Image},
            out_types={'Output image': Image},
            parameters={'Gamma': FloatParameter('Gamma', 1, 0, 2)},
            fn=gamma_contrast
        )

core\nodes\gaussian_blur.py
from core import Node
from core.datatypes import Image_3C
from core.parameters import IntParameter

import cv2


def gaussian_blur(data_in):
    in_image = data_in['Input image']['image']
    kernel_size = data_in['Kernel size']
    if kernel_size % 2 == 0:
        kernel_size -= 1
    return {'Output image': dict(image=cv2.GaussianBlur(
        in_image, (kernel_size, kernel_size), 0), colorspace=data_in['Input
image']['colorspace'])}


class GaussianBlur(Node):
```

```
    def __init__(self):
        super(GaussianBlur, self).__init__(
            in_types={'Input image': Image_3C},
            out_types={'Output image': Image_3C},
            parameters={
                'Kernel size': IntParameter(
                    'Kernel size', 3, 1, 11, 2)},
            fn=gaussian_blur
        )
```

core\nodes\gray_kmeans.py

```
from core.parameters import IntParameter
from core.datatypes import Image_GRAY
from core import Node
import numpy as np


def gray_kmeans(data_in):
    in_image = data_in['Input image']['image']
    clusters = data_in['Clusters']

    def kmeans(X, n_clusters=2, max_iter=300, tol=1e-4):
        uniques, counts = np.unique(X, return_counts=True)
        clusters = np.random.choice(uniques, n_clusters)
        clusters = clusters + np.random.uniform(low=-0.5, high=0.5,
size=clusters.shape)

        mass = (uniques * counts)

        for i in range(max_iter):
            dists = np.abs(clusters.reshape(-1, 1) - uniques.reshape(1, -1))
            argmin = dists.argmin(axis=0)
            clust = argmin.reshape(1, -1) == np.arange(n_clusters).reshape(-1, 1)
            new_clust = np.array([
                mass[clust[c]].sum() / counts[clust[c]].sum()
                for c in range(n_clusters)
            ])
            if np.all(np.abs(clusters - new_clust) < tol):
                break
            clusters = np.sort(new_clust)

        dists = np.abs(clusters.reshape(-1, 1) - X.reshape(1, -1))
        argmin = dists.argmin(axis=0)

        return clusters, argmin, clusters[argmin]

    clust, segmap, clustered = kmeans(in_image.reshape(-1, 1), n_clusters=clusters)

    segmap = segmap.reshape(in_image.shape)
    clustered = clustered.reshape(in_image.shape)

    return {
        'Clustered': dict(image=clustered.astype(np.uint8), colorspace='GRAY'),
        'Segmentation map': dict(image=clustered.astype(np.int32), colorspace='GRAY')
    }
```

```
class GrayKMeans(Node):

    def __init__(self):
        super(GrayKMeans, self).__init__(
            in_types={'Input image': Image_GRAY},
            out_types={
                'Clustered': Image_GRAY,
                'Segmentation map': Image_GRAY
            },
            parameters={'Clusters': IntParameter('Clusters', 2, 2, 20, 1)},
            fn=gray_kmeans
        )

core\nodes\kmeans.py
from core.parameters import IntParameter
from core.datatypes import Image
from core import Node
import numpy as np
from sklearn.cluster import KMeans as SK_KMeans


def kmeans(data_in):
    in_image = data_in['Input image']['image']
    clusters = data_in['Clusters']

    clust = SK_KMeans(
        clusters,
        n_init=1
    )
    if in_image.ndim == 3:
        idx = clust.fit_predict(in_image.reshape(
            np.prod(in_image.shape[:-1]), -1))
    else:
        idx = clust.fit_predict(in_image.reshape(np.prod(in_image.shape), -1))
    clustered = clust.cluster_centers_[idx].reshape(in_image.shape)

    return {'Output image': dict(image=clustered.astype(
        np.uint8), colorspace=data_in['Input image']['colorspace'])}


if __name__ == '__main__':
    data_in = dict(
        in_image=np.random.uniform(
            0, 255, size=(
                256, 256, 3)).astype(
            np.uint8),
        clusters=2
    )
    print(kmeans(data_in))


class KMeans(Node):

    def __init__(self):
        super(KMeans, self).__init__(
```

```
            in_types={'Input image': Image},
            out_types={'Output image': Image},
            parameters={'Clusters': IntParameter('Clusters', 2, 2, 20, 1)},
            fn=kmeans
        )

core\nodes\linear_contrast.py
from core import Node
from core.datatypes import Image
from core.parameters import FloatParameter

import numpy as np


def linear_contrast(data_in):
    in_image = data_in['Input image']['image']
    alpha = data_in['Alpha']

    return {'Output image': dict(image=np.clip(128 + (in_image.astype(np.int32) -
128)
                                               * alpha, 0, 255).astype(np.uint8),
colorspace=data_in['Input image']['colorspace'])}


class LinearContrast(Node):

    def __init__(self):
        super(LinearContrast, self).__init__(
            in_types={'Input image': Image},
            out_types={'Output image': Image},
            parameters={'Alpha': FloatParameter('Alpha', 1, 1, 2)},
            fn=linear_contrast
        )

core\nodes\morphlogy.py
from core import Node
from core.datatypes import Image_BINARY
from core.parameters import IntParameter, OptionParameter

import cv2

option_to_op = {
    'Open': cv2.MORPH_OPEN,
    'Close': cv2.MORPH_CLOSE,
    'Gradient': cv2.MORPH_GRADIENT,
    'Top hat': cv2.MORPH_TOPHAT,
    'Black': cv2.MORPH_BLACKHAT
}
option_to_element = {
    'Rectangle': cv2.MORPH_RECT,
    'Ellipse': cv2.MORPH_ELLIPSE,
    'Cross': cv2.MORPH_CROSS
}


def morphology(data_in):
```

```
    in_image = data_in['BINARY image']['image']
    kernel_size = data_in['Kernel size']

    op = data_in['Operation']
    element = option_to_element[data_in['Element']]
    element = cv2.getStructuringElement(element, (kernel_size, kernel_size))

    if op == 'Erode':
        out_image = cv2.erode(in_image, element)
    elif op == 'Dilate':
        out_image = cv2.dilate(in_image, element)
    else:
        out_image = cv2.morphologyEx(in_image, option_to_op[op], element)

    return {
        'BINARY image': dict(image=out_image, colorspace='BINARY')
    }


class Morphology(Node):

    def __init__(self):
        super(Morphology, self).__init__(
            in_types={'BINARY image': Image_BINARY},
            out_types={'BINARY image': Image_BINARY},
            parameters={
                'Kernel size': IntParameter('Kernel size', 3, 1, 11, 1),
                'Operation': OptionParameter(
                    'Operation',
                    'Erode',
                    ['Erode', 'Dilate'] + list(option_to_op.keys())
                ),
                'Element': OptionParameter(
                    'Element',
                    'Rectangle',
                    list(option_to_element.keys())
                ),
            },
            fn=morphology
        )

core\nodes\random_image.py
from core import Node
from core.datatypes import *

import numpy as np


def read(data_in):
    return {
        'RGB image': dict(image=np.random.uniform(0, 255, (256, 256,
3)).astype(np.uint8), colorspace='RGB'),
        'GRAY image': dict(image=np.random.uniform(0, 255, (256,
256)).astype(np.uint8), colorspace='GRAY'),
        'HSV image': dict(image=np.random.uniform(0, 2551, (256, 256,
3)).astype(np.uint8), colorspace='HSV'),
```

```
        'LAB image': dict(image=np.random.uniform(0, 255, (256, 256,
3)).astype(np.uint8), colorspace='LAB'),
        'YCrCb image': dict(image=np.random.uniform(0, 255, (256, 256,
3)).astype(np.uint8), colorspace='YCrCb')
    }


class RandomImage(Node):

    def __init__(self):
        super(RandomImage, self).__init__(
            out_types={
                'RGB image': Image_RGB,
                'GRAY image': Image_GRAY,
                'HSV image': Image_HSV,
                'LAB image': Image_LAB,
                'YCrCb image': Image_YCrCb,
            },
            fn=read
        )

core\nodes\read_from_file_roi.py
from core import Node
from core.datatypes import *
from core.parameters import ImageROIParameter

import numpy as np
import cv2
import os


def propagate(data_in):
    image_rgb = data_in['Image']['image']
    if image_rgb is not None:
        return {
            'RGB': dict(image=image_rgb, colorspace='RGB'),
            'GRAY': dict(image=cv2.cvtColor(image_rgb, cv2.COLOR_RGB2GRAY),
colorspace='GRAY'),
            'HSV': dict(image=cv2.cvtColor(image_rgb, cv2.COLOR_RGB2HSV),
colorspace='HSV'),
            'LAB': dict(image=cv2.cvtColor(image_rgb, cv2.COLOR_RGB2LAB),
colorspace='LAB'),
            'YCrCb': dict(image=cv2.cvtColor(image_rgb, cv2.COLOR_RGB2YCrCb),
colorspace='YCrCb')
        }
    else:
        return {
            'RGB': None,
            'GRAY': None,
            'HSV': None,
            'LAB': None,
            'YCrCb': None
        }

class ReadFromFileROI(Node):
```

```
    def __init__(self):
        super(ReadFromFileROI, self).__init__(
            out_types={
                'RGB': Image_RGB,
                'GRAY': Image_GRAY,
                'HSV': Image_HSV,
                'LAB': Image_LAB,
                'YCrCb': Image_YCrCb,
            },
            parameters={'Image': ImageROIParameter('Image')},
            fn=propagate
        )

    def _params_to_serialize(self):
        return dict()
```

core\nodes\show.py

```
from core import Node
from core.datatypes import Image
from core.parameters import ColormapParameter, BoolParameter

import numpy as np
import matplotlib


def show(data_in):
    image = data_in['Input image']['image']
    colormap = data_in['Colormap']
    if image.shape[-1] != 3:
        colormap = matplotlib.cm.get_cmap(colormap)
        if data_in['Normalize']:
            m = image.min()
            image = (image - m) / (image.max() - m)
        else:
            image = image / 255
        image = (colormap(image)[..., :3] * 255).astype(np.uint8)
    return {'-': dict(image=image, colorspace='RGB')}


class Show(Node):

    def __init__(self):
        super(Show, self).__init__(
            in_types={'Input image': Image},
            out_types={'-': Image},
            out_values_to_return=set('-'),
            parameters={
                'Colormap': ColormapParameter('Colormap'),
                'Normalize': BoolParameter('Normalize', False)
            },
            fn=show
        )
```

core\nodes\split.py

```
from core import Node
from core.datatypes import Image_3C, Image_GRAY
```

```
from core.parameters import IntParameter

import numpy as np
import cv2


def split(data_in):
    in_image = data_in['Color image']['image']
    return {
        'Channel 1': dict(image=in_image[..., 0], colorspace='GRAY'),
        'Channel 2': dict(image=in_image[..., 1], colorspace='GRAY'),
        'Channel 3': dict(image=in_image[..., 2], colorspace='GRAY'),
    }


class Split(Node):

    def __init__(self):
        super(Split, self).__init__(
            in_types={'Color image': Image_3C},
            out_types={
                'Channel 1': Image_GRAY,
                'Channel 2': Image_GRAY,
                'Channel 3': Image_GRAY},
            fn=split
        )

core\nodes\threshold.py
from core import Node
from core.datatypes import Image_GRAY, Image_BINARY
from core.parameters import FloatParameter, OptionParameter

import cv2

option_to_alg_flag = {
    '-': 0,
    'Otsu': cv2.THRESH_OTSU,
    'Triangle': cv2.THRESH_TRIANGLE
}
option_to_type_flag = {
    'Binary': cv2.THRESH_BINARY,
    'Binary inverted': cv2.THRESH_BINARY_INV,
    'Truncate': cv2.THRESH_TRUNC,
    'ToZero': cv2.THRESH_TOZERO,
    'ToZero inverted': cv2.THRESH_TOZERO_INV
}


def threshold(data_in):
    in_image = data_in['GRAY image']['image']
    t = data_in['Threshold']

    algorithm = option_to_alg_flag[data_in['Algorithm']]
    type = option_to_type_flag[data_in['Type']]

    _, out_image = cv2.threshold(in_image, t * 255, 255, type + algorithm)
```

```
    if 'Binary' in data_in['Type']:
        return {
            'BINARY image': dict(image=out_image, colorspace='BINARY'),
            'GRAY image': dict(image=out_image, colorspace='GRAY')
        }
    return {
        'BINARY image': None,
        'GRAY image': dict(image=out_image, colorspace='GRAY')
    }


class Threshold(Node):

    def __init__(self):
        super(Threshold, self).__init__(
            in_types={'GRAY image': Image_GRAY},
            out_types={
                'BINARY image': Image_BINARY,
                'GRAY image': Image_GRAY
            },
            parameters={
                'Threshold': FloatParameter('Threshold', 0.5, 0, 1),
                'Algorithm': OptionParameter('Algorithm', '-', ['-', 'Otsu',
'Triangle']),
                'Type': OptionParameter('Type', 'Binary', ['Binary', 'Binary
inverted', 'Truncate', 'ToZero', 'ToZero inverted']),
            },
            fn=threshold
        )

core\nodes\watershed.py
from core import Node
from core.datatypes import Image_GRAY, Image_BINARY
from core.parameters import FloatParameter, OptionParameter

import numpy as np
import cv2


def watershed(data_in):
    thresholded = data_in['Thresholded']['image']

    dist_transform = cv2.distanceTransform(thresholded, cv2.DIST_L2, 5)
    ret, sure_fg = cv2.threshold(dist_transform, data_in['Distance threshold'] *
dist_transform.max(), 255, 0)
    sure_fg = sure_fg.astype(np.uint8)

    ret, markers = cv2.connectedComponents(sure_fg)
    markers = markers + 1
    markers[cv2.subtract(thresholded, sure_fg)] = 0

    #markers = cv2.watershed(img, markers)

    dist_transform = (dist_transform - dist_transform.min()) / (dist_transform.max()
- dist_transform.min())
```

```
    dist_transform = (dist_transform * 255).astype(np.uint8)
    out_image = sure_fg

    return {
        'Distance transform': dict(image=dist_transform, colorspace='GRAY'),
        'GRAY image': dict(image=markers, colorspace='GRAY')
    }


class Watershed(Node):

    def __init__(self):
        super(Watershed, self).__init__(
            in_types={
                'Thresholded': Image_BINARY
            },
            parameters={
                'Distance threshold': FloatParameter('Distance threshold', 0.7, 0.,
1.)
            },
            out_types={
                'GRAY image': Image_GRAY,
                'Distance transform': Image_GRAY,
            },
            fn=watershed
        )

core\nodes\write_to_file.py
from core import Node
from core.datatypes import Image_RGB
from core.parameters import FilenameParameter

import numpy as np
import cv2


def write(data_in):
    filename = data_in['File']
    image = data_in['Input image']['image']
    print(data_in)
    cv2.imwrite(
        filename,
        cv2.cvtColor(
            (image *
             255).astype(
                np.uint8),
            cv2.COLOR_BGR2RGB))
    return dict()


class WriteToFile(Node):

    def __init__(self):
        super(WriteToFile, self).__init__(
            in_types={'Input image': Image_3C},
            parameters={'File': FilenameParameter('File', open=False)},
```

```
            fn=write
        )
```

core\nodes__init__.py

```
from .write_to_file import WriteToFile
from .show import Show
from .gaussian_blur import GaussianBlur
from .linear_contrast import LinearContrast
from .gamma_contrast import GammaContrast
from .kmeans import KMeans
from .gray_kmeans import GrayKMeans
from .alpha_blending import AlphaBlending
from .change_colorspace import ChangeColorspace
from .random_image import RandomImage
from .threshold import Threshold
from .split import Split
from .morphlogy import Morphology
from .read_from_file_roi import ReadFromFileROI
from .compose import Compose
from .watershed import Watershed
from .brightness_normalization import BrightnessNormalization

str_to_classname = {
    'ReadFromFileROI': ReadFromFileROI,
    'WriteToFile': WriteToFile,
    'Show': Show,
    'GaussianBlur': GaussianBlur,
    'LinearContrast': LinearContrast,
    'GammaContrast': GammaContrast,
    'KMeans': KMeans,
    'GrayKMeans': GrayKMeans,
    'AlphaBlending': AlphaBlending,
    'ChangeColorspace': ChangeColorspace,
    'RandomImage': RandomImage,
    'Threshold': Threshold,
    'Morphology': Morphology,
    'Split': Split,
    'Compose': Compose,
    'Watershed': Watershed,
    'BrightnessNormalization': BrightnessNormalization
}

node_tree = {
    'ReadFromFileROI': ['I/O'],
    'WriteToFile': ['I/O'],
    'Show': ['I/O'],
    'RandomImage': ['I/O'],
    'GaussianBlur': ['Preprocessing', 'Blur'],
    'LinearContrast': ['Preprocessing', 'Normalization'],
    'GammaContrast': ['Preprocessing', 'Normalization'],
    'KMeans': ['Segmentation'],
    'GrayKMeans': ['Segmentation'],
    'Watershed': ['Segmentation'],
    'AlphaBlending': ['Misc'],
    'ChangeColorspace': ['Misc'],
    'Threshold': ['Preprocessing'],
```

```
    'Morphology': ['Preprocessing'],
    'Split': ['Preprocessing'],
    'Compose': ['Preprocessing'],
    'BrightnessNormalization': ['Preprocessing', 'Normalization']
}

core\parameters\bool_parameter.py
from PyQt5 import QtWidgets
from core import Parameter


class BoolParameter(Parameter):

    def __init__(self, name, init):
        super(BoolParameter, self).__init__(name)

        self._set_value(init)

    def format_value(self):
        return str(self.get_value())

    def _new_control(self):
        checkbox = QtWidgets.QCheckBox()
        checkbox.setStyleSheet("""

        """)

        def _(v):
            self._set_value(v == 2)
            self.valueChanged.emit()
        checkbox.stateChanged.connect(_)
        checkbox.setCheckState(self.get_value())
        self.valueChanged.connect(
            lambda: checkbox.setCheckState(
                2 if self.get_value() else 0))
        return checkbox

core\parameters\colormap_parameter.py
from core.parameters import OptionParameter

import matplotlib.pyplot as plt


class ColormapParameter(OptionParameter):

    def __init__(self, name):
        init = 'gray'
        colormaps = plt.colormaps()

        super(
            ColormapParameter,
            self).__init__(
            name,
            init=init,
            options=colormaps)
```

```
core\parameters\filename_parameter.py
from PyQt5 import QtWidgets
from core import Parameter
import os


class FilenameParameter(Parameter):
    def __init__(self, name, open):
        super(FilenameParameter, self).__init__(name)
        self.open = open

    def _new_control(self):
        open_filechooser = QtWidgets.QPushButton()
        open_filechooser.setStyleSheet("""
            background: #3c3c3c;
            border-radius: 4px;
            height: 25px;
            color: #c8c8c8; font: 11pt \"cm sans serif 2012\";
            margin-right: 8px;
        """)
        open_filechooser.setText('File...')

        def choose_file(v):
            if self.open:
                file, _ = QtWidgets.QFileDialog().getOpenFileName()
            else:
                file, _ = QtWidgets.QFileDialog().getSaveFileName()
            self._set_value(file)
            self.valueChanged.emit()
        open_filechooser.clicked.connect(choose_file)

        self.valueChanged.connect(
            lambda: open_filechooser.setText(
                os.path.basename(
                    self.get_value())))

        return open_filechooser

core\parameters\float_parameter.py
from PyQt5 import QtCore, QtGui, QtWidgets
from core import Parameter


class FloatSlider(QtWidgets.QWidget):

    valueChanged = QtCore.pyqtSignal(float)

    def __init__(self, init, min, max, *args, **kwargs):
        super(FloatSlider, self).__init__()

        self.min, self.max = min, max

        self.slider = QtGui.QSlider(*args, **kwargs)
        label = QtGui.QLabel()

        layout = QtGui.QHBoxLayout()
```

```
        self.setLayout(layout)
        layout.addWidget(label)
        layout.addWidget(self.slider)
        self.slider.valueChanged.connect(
            lambda *args: label.setText('%.2f' % self.get_value()))
        self.slider.sliderReleased.connect(
            lambda: self.valueChanged.emit(
                self.get_value()))

        self.setValue = self.slider.setValue
        self.slider.setValue(int(99 * (init - min) / (max - min)))

        self.valueChanged.emit(init)

    def get_value(self):
        return (self.max - self.min) * self.slider.value() / 99 + self.min


class FloatParameter(Parameter):

    def __init__(self, name, init, min, max):
        super(FloatParameter, self).__init__(name)

        self.name, self.init, self.min, self.max = name, init, min, max
        self._set_value(init)

    def _new_control(self):
        slider = FloatSlider(
            self.init,
            self.min,
            self.max,
            QtCore.Qt.Horizontal)

        def _(v):
            self._set_value(v)
            self.valueChanged.emit()
        slider.valueChanged.connect(_)
        self.valueChanged.connect(
            lambda: slider.setValue(int(self.datavalue.get_data())))
        slider.setStyleSheet("""
            QSlider::handle::horizontal  {
                border-radius: 3px;
                background: #c8c8c8;
                height: 12px;
                width: 10px;
            }

            QSlider::groove::horizontal {
                background: qlineargradient(spread:pad, x1:0.506, y1:0.75, x2:0.5,
y2:0.25, stop:0.375 rgba(0, 0, 0, 0), stop:0.376 rgba(60, 60, 60, 255), stop:0.675
rgba(60, 60, 60, 255), stop:0.676 rgba(0, 0, 0, 0));
            }

            QLabel {
                color: #c8c8c8; font: 11pt \"cm sans serif 2012\";
            }
```

```
        """)
        return slider

    def format_value(self):
        return '%.2f' % self.get_value()

    def get_value(self):
        return self.min + (self.max - self.min) * \
            self.datavalue.get_data() / 100

    def _set_value(self, value):
        self.datavalue.set_data(
            int(99 * (value - self.min) / (self.max - self.min)))
```

core\parameters\image_roi_parameter.py

```
from core import Parameter
from core.datatypes import Image


class ImageROIParameter(Parameter):

    def __init__(self, name):
        super(ImageROIParameter, self).__init__(name, dtype=Image)

    def assign_control(self, control):
        def _():
            self.set_value(dict(image=control.get_image(), colorspace='RGB'))
        control.roiChanged.connect(_)
        _()
```

core\parameters\int_parameter.py

```
from PyQt5 import QtWidgets
from core import Parameter


class IntParameter(Parameter):

    def __init__(self, name, init, min, max, step):
        super(IntParameter, self).__init__(name)

        self.init, self.min, self.max, self.step = init, min, max, step
        self._set_value(init)

    def format_value(self):
        return '%d' % self.get_value()

    def _new_control(self):
        spinbox = QtWidgets.QSpinBox()
        spinbox.setMinimum(self.min)
        spinbox.setMaximum(self.max)
        spinbox.setSingleStep(self.step)
        spinbox.lineEdit().setReadOnly(True)
        spinbox.setStyleSheet("""
            QSpinBox {
                background: #3c3c3c;
                border-radius: 4px;
```

```
                height: 25px;
                color: #c8c8c8; font: 11pt \"cm sans serif 2012\";
                margin-right: 8px;
            }
            QSpinBox::up-button, QSpinBox::down-button {
                border-radius: 3px;
                background: #c8c8c8;
                width: 12px;
                margin: 2px;
            }"""
                            )

        def _(v):
            self._set_value(v)
            self.valueChanged.emit()
        spinbox.valueChanged.connect(_)
        spinbox.setValue(self.init)
        self.valueChanged.connect(lambda: spinbox.setValue(self.get_value()))
        return spinbox
```

core\parameters\numerical_parameter.py

```
from PyQt5 import QtCore, QtWidgets
from core import Parameter


class NumericalParameter(Parameter):

    def __init__(self, name):
        super(NumericalParameter, self).__init__(name)

    def format_value(self):
        pass

    def get_slider(self, min=0, max=100, step=1):
        slider = QtWidgets.QSlider()
        slider.setMinimum(min)
        slider.setMaximum(max)
        slider.setSingleStep(step)
        slider.setOrientation(QtCore.Qt.Horizontal)

        def value_changed(v):
            self.set_value(v)
            self.changed_signal.emit()
        slider.valueChanged.connect(value_changed)
        self.changed_signal.connect(lambda: slider.setValue(self.get_value()))
        slider.setValue((min + max) // 2)
        return slider

    def new_control(self):
        widget = QtWidgets.QWidget()
        vl = QtWidgets.QVBoxLayout()
        widget.setLayout(vl)

        name_label = QtWidgets.QLabel()
        name_label.setText(self.name)
        name_label.setAlignment(QtCore.Qt.AlignHCenter)
```

```
        vl.addWidget(name_label)

        hl = QtWidgets.QHBoxLayout()
        vl.addLayout(hl)

        slider = self.get_slider()

        display = QtWidgets.QLabel()
        display.setText(self.format_value())
        self.changed_signal.connect(
            lambda: display.setText(
                self.format_value()))

        hl.addWidget(slider)
        hl.addWidget(display)

        return widget
```

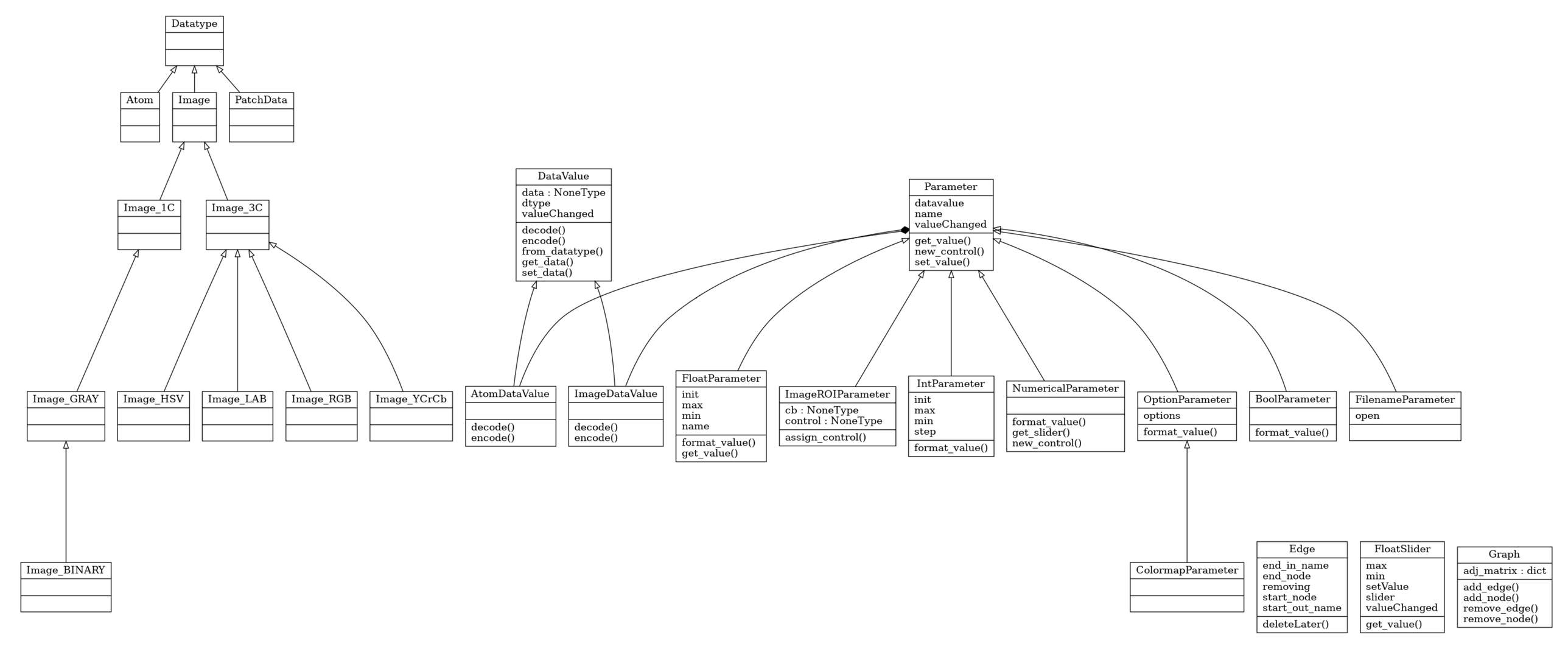
core\parameters\option_parameter.py

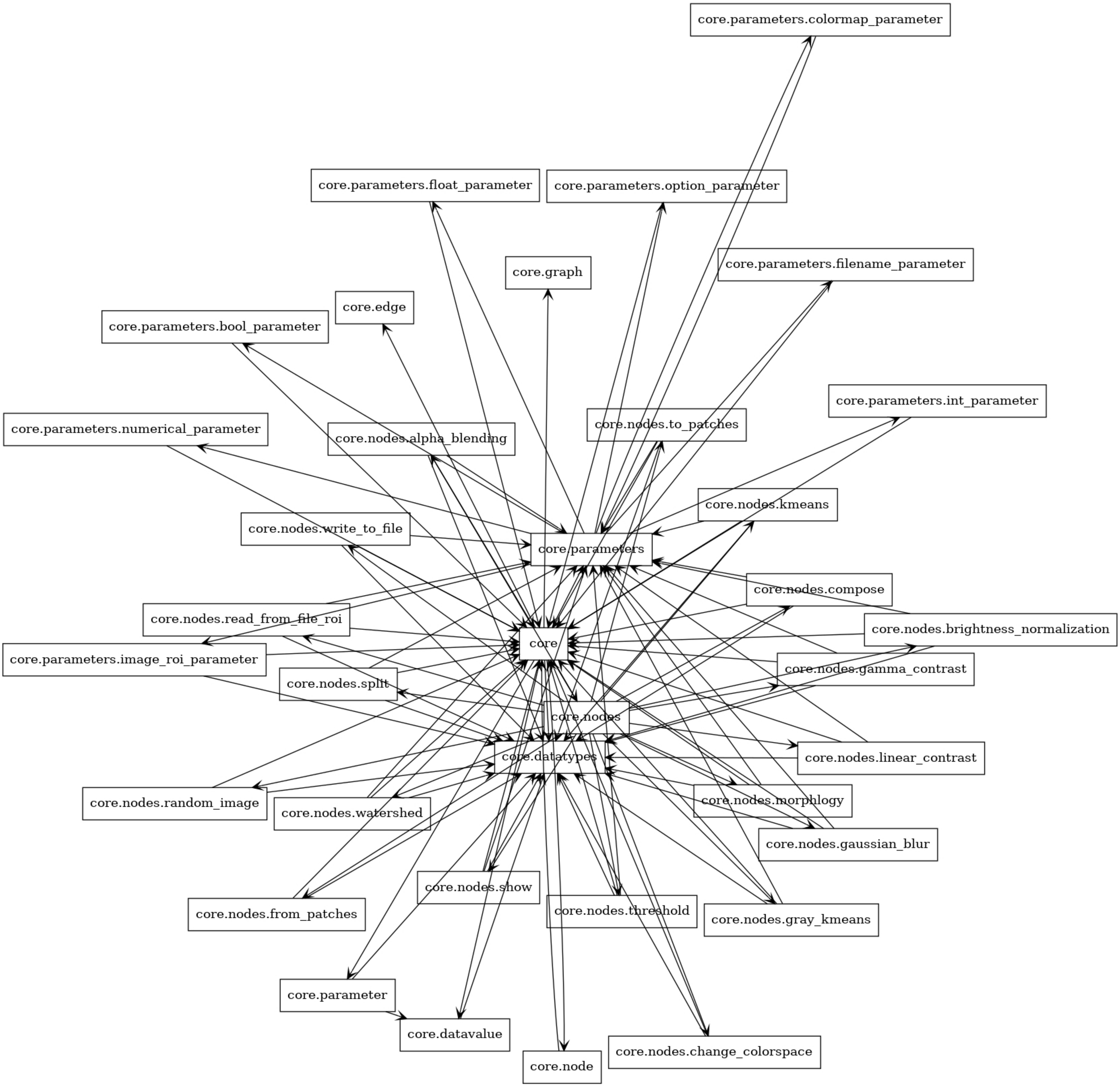
```
from PyQt5 import QtWidgets
from core import Parameter


class OptionParameter(Parameter):

    def __init__(self, name, init, options):
        super(OptionParameter, self).__init__(name)

        self.options = options
        self._set_value(init)

    def format_value(self):
        return str(self.get_value())

    def _new_control(self):
        combobox = QtWidgets.QComboBox()
        for option in self.options:
            combobox.addItem(option)
        combobox.setStyleSheet("""
            background-color: #3c3c3c;
            border: 8px solid #3c3c3c;
            border-radius: 8px;
            color: \"#FFFFFF\";
            padding: 0px;
            font: 10pt \"cm sans serif 2012\";
        """)

        def _(v):
            self.set_value(v)
            self.valueChanged.emit()
        combobox.currentTextChanged.connect(_)
        combobox.setCurrentText(self.get_value())
        self.valueChanged.connect(
            lambda: combobox.setCurrentText(
                self.get_value()))
        return combobox
```

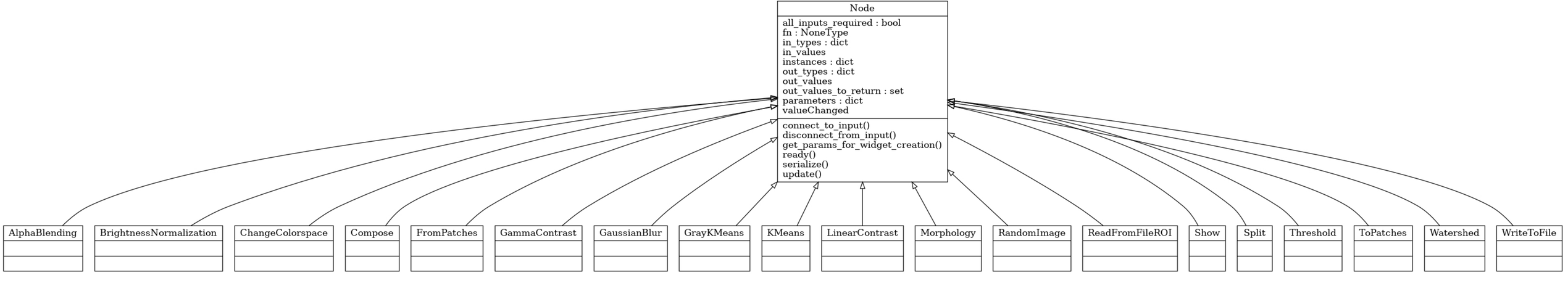
core\parameters__init__.py

```
from .numerical_parameter import NumericalParameter
from .int_parameter import IntParameter
from .float_parameter import FloatParameter
from .filename_parameter import FilenameParameter
from .bool_parameter import BoolParameter
from .option_parameter import OptionParameter
from .colormap_parameter import ColormapParameter
from .image_roi_parameter import ImageROIParameter
```

ДОДАТОК 2 ГРАФІЧНІ МАТЕРІАЛИ


Datatype
Atom
Image
PatchData
Image_1C
Image_3C
Image_GRAY
Image_HSV
Image_LAB
Image_RGB
Image_YCrCb
Image_BINARY
DataValue
data : NoneType
dtype
valueChanged
decode()
encode()
from_datatype()
get_data()
set_data()
AtomDataValue
decode()
encode()
ImageDataValue
decode()
encode()
Parameter
datavalue
name
valueChanged
get_value()
new_control()
set_value()
FloatParameter
init
max
min
name
format_value()
get_value()
ImageROIParameter
cb : NoneType
control : NoneType
assign_control()
IntParameter
init
max
min
step
format_value()
NumericalParameter
format_value()
get_slider()
new_control()
OptionParameter
options
format_value()
BoolParameter
format_value()
FilenameParameter
open
ColormapParameter
Edge
end_in_name
end_node
removing
start_node
start_out_name
deleteLater()
FloatSlider
max
min
setValue
slider
valueChanged
get_value()
Graph
adj_matrix : dict
add_edge()
add_node()
remove_edge()
remove_node()


core.parameters.colormap_parameter
core.parameters.float_parameter
core.parameters.option_parameter
core.graph
core.parameters.filename_parameter
core.edge
core.parameters.bool_parameter
core.parameters.int_parameter
core.parameters.numerical_parameter
core.nodes.alpha_blending
core.nodes.to_patches
core.nodes.kmeans
core.nodes.write_to_file
core.parameters
core.nodes.compose
core.nodes.read_from_file_roi
core.nodes.brightness_normalization
core
core.parameters.image_roi_parameter
core.nodes.gamma_contrast
core.nodes.split
core.nodes
core.datatypes
core.nodes.linear_contrast
core.nodes.random_image
core.nodes.morphlogy
core.nodes.watershed
core.nodes.gaussian_blur
core.nodes.show
core.nodes.from_patches
core.nodes.threshold
core.nodes.gray_kmeans
core.parameter
core.datavalue
core.node
core.nodes.change_colorspace


Node
all_inputs_required : bool
fn : NoneType
in_types : dict
in_values
instances : dict
out_types : dict
out_values
out_values_to_return : set
parameters : dict
valueChanged
connect_to_input()
disconnect_from_input()
get_params_for_widget_creation()
ready()
serialize()
update()
AlphaBlending
BrightnessNormalization
ChangeColorspace
Compose
FromPatches
GammaContrast
GaussianBlur
GrayKMeans
KMeans
LinearContrast
Morphology
RandomImage
ReadFromFileROI
Show
Split
Threshold
ToPatches
Watershed
WriteToFile