КАЗАНСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ

ОСНОВНЫЕ АЛГОРИТМЫ СЕГМЕНТАЦИИ ИЗОБРАЖЕНИЙ

Учебное пособие

КАЗАНЬ

2024

УДК 004.93(075.8)
ББК 16.63я73
О-75

Печатается по рекомендации
Редакционно-издательского совета
Казанского (Приволжского) федерального университета

Авторы:
Д.Н. Тумаков, З.Д. Каюмов, А.Г. Маркина,
Д.И. Хайруллина, А.А. Егорчев

Научный редактор
кандидат физико-математических наук, доцент **Ф.М. Гафаров**

Рецензенты:
доктор физико-математических наук, профессор **Н.Б. Плещинский**;
кандидат физико-математических наук, доцент **А.А. Колчев**

О-75 **Основные алгоритмы сегментации изображений** [Электронный ресурс]: учебное пособие / Д.Н. Тумаков, З.Д. Каюмов, А.Г. Маркина и др. – Электронные текстовые данные (1 файл: 12,2 Мб). – Казань: Издательство Казанского университета, 2024. – 76 с. – Системные требования: Adobe Acrobat Reader. – Загл. с титул. экрана.

ISBN 978-5-00130-837-9

В учебном пособии описаны основные методы и алгоритмы выделения объектов и их границ на изображениях. Подробно рассмотрены классические методы сегментации с математическим описанием самих методов. Приведены примеры программ на языке Python. Показаны результаты сегментации различными алгоритмами на макроснимках зерен горных пород. В конце каждой главы имеются контрольные вопросы и задания для лабораторных работ.

Пособие предназначено для студентов ИТ-специальностей с математическим уклоном.

УДК 004.93(075.8)
ББК 16.63я73

ISBN 978-5-00130-837-9

© Издательство Казанского университета, 2024

Введение

Сегментация является одним из важных этапов обработки изображений, который заключается в выделении объектов и их границ на изображениях. Сегментация широко применяется в таких областях, как компьютерное зрение, медицинская диагностика, автоматическое распознавание образов и многих других.

В настоящем учебном пособии рассмотрены различные методы сегментации изображений, прежде всего такие классические методы, как пороговая сегментация и кластерный анализ. Выделены основные проблемы, с которыми можно столкнуться при проведении сегментации изображений, такие как шумы, размытие, слабые контрасты и другие, предложены методы устранения этих проблем.

Все практические примеры сегментации представлены на языке программирования Python, который широко используется в области обработки изображений. Приведены краткие описания некоторых библиотечных функций, которые используются при обработке изображений. В пособии использованы следующие библиотеки.

1. *NumPy*: Библиотека для работы с многомерными массивами данных. Эта библиотека позволяет эффективно выполнять математические операции и манипуляции с данными.
2. *OpenCV*: Open Source Computer Vision Library – мощный инструмент для обработки изображений и видео. Предоставляет набор функций для чтения, обработки и анализа визуальных данных.
3. *Scikit-image*: Библиотека, специализирующаяся на обработке изображений. Содержит функции, реализующие базовые операции над изображениями, такие как фильтрация, морфологические операции и сегментация.
4. *Matplotlib*: Библиотека для визуализации и создания графиков. Используется для визуализации результатов сегментации.

Сегментация изображений – это процесс выделения на изображении нескольких областей (сегментов), которые содержат объекты с определенными свойствами. В пособии рассматриваются следующие наиболее распространенные методы сегментации изображений.

1. *Пороговая сегментация* – это простой метод, который основан на выборе порогового значения для интенсивности или цвета пикселей, чтобы разделить изображение на две или более областей. Наиболее распространенными алгоритмами пороговой сегментации являются Global Thresholding, Adaptive Thresholding, Otsu's Thresholding.
2. *Методы на основе регионов* – это методы, которые сегментируют изображение путём объединения соседних пикселей со схожими параметрами (цвет, яркость и т.д.) в регионы. К таким методам относятся Region Growing, Split-and-Merge, Watershed.
3. *Методы сегментации на основе границ* – это методы, которые используют границы объектов на изображении для выделения различных областей. Наиболее известными методами выделения границ являются Sobel filter и Canny filter.
4. *Методы сегментации на основе кластеризации* – это методы, которые используют алгоритмы кластеризации для группировки пикселей в кластеры схожих свойств. Примерами таких алгоритмов являются K-Means Clustering, Mean Shift, DBSCAN.
5. *Сегментация изображений на графах* – это методы, которые используют граф для представления изображения, а затем находят минимальный разрез в графе, чтобы разделить изображение на несколько областей. К таким алгоритмам относятся Min-Cut, Normalized Cut, GrabCut.
6. *Сегментация методом активных контуров* – методы, которые используют кривые, называемые контурами, для выделения объектов на изображении. Активный контур – это кривая, которая "активно" перемещается на изображении в направлении границы объекта и следует за контурами объекта, пока не будет полностью его охватывать.

На сегодняшний день для сегментации изображений также активно используют подходы на основе глубокого обучения. К этой группе методов относят сверточные нейронные сети (U-Net, Mask R-CNN, Panoptic FPN и другие), обучение которых проводят на больших наборах данных с размеченными изображениями. Методы с использованием нейронных сетей не будут рассмотрены в настоящем пособии.

Оглавление

Глава 1. Пороговая сегментация

Пороговая бинаризация

Пороговая (глобальная) бинаризация – это метод обработки изображений, который позволяет преобразовать цветное изображение или изображение в оттенках серого (цвета) в черно-белое. Белый цвет соответствует пикселям, яркость которых выше определенного порога, а черный цвет – пикселям, яркость которых ниже этого порога. Процесс пороговой бинаризации состоит из следующих шагов.

1. Конвертирование изображения в оттенки серого, если оно не было в таком формате.
2. Определение порогового значения, по которому будет происходить бинаризация.
3. Перебор всех пикселей изображения и сравнение их яркости с пороговым значением.
4. Если яркость пикселя выше порогового значения, то присвоение ему белого цвета, а если ниже – черного цвета.

Популярным методом глобальной бинаризации изображений принято считать *метод Оцу (Otsu's method)*. Более подробно с этим методом можно ознакомиться в оригинальной статье [1]. Метод Оцу фактически представляет собой алгоритм вычисления оптимального порогового значения, который разделяет пиксели на два класса (объекты и фон).

Для расчёта оптимального порога методом Оцу используют гистограмму изображения. Под гистограммой изображения здесь и в дальнейшем будем понимать график распределения пикселей с различной яркостью, где по оси абсцисс указывают значения яркости пикселей от 0 до 255, а по оси ординат – относительное (в редких случаях абсолютное) число пикселей с конкретным значением яркости.

Также гистограмму можно представить с вероятностной точки зрения: на оси ординат отображать не сумму пикселей с определенной яркостью, а их вероятность (это значение можно получить, разделив эти суммы на общее число пикселей). В этом случае значения пикселей изображения – это случайные величины, а их гистограмма – оценка плотности распределения вероятностей. На рисунке 1 представлены изображение в серых тонах (классический пример – дама в шляпе), а также гистограмма распределения яркостей пикселей этого изображения.

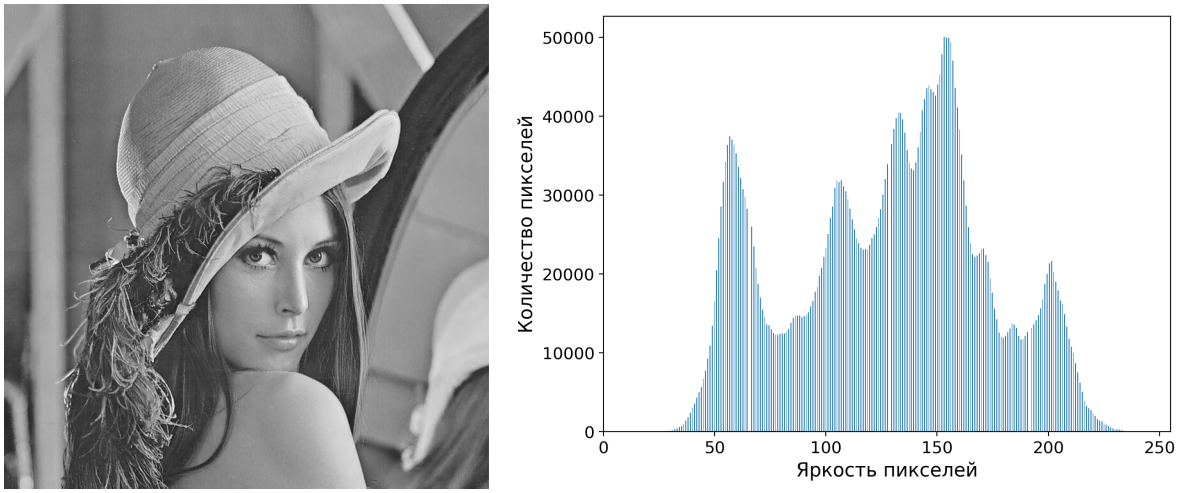


Рисунок 1 – Изображение в градациях серого и гистограмма яркостей пикселей этого изображения

Предположим, что

- изображение описывается с помощью уровней яркости L (изменяется от 0 до 255);
- n_i – число пикселей изображения с уровнем яркости i, $i = 0,1,\dots,L-1$;
- N – общее число пикселей изображения.

Тогда гистограмму изображения можем рассматривать как распределение вероятностей

$$p_i = \frac{n_i}{N}, i = 0,1,\dots,L-1, \quad \sum_{i=0}^{L-1} p_i = 1.$$

Так как теперь известны плотности распределения вероятностей, то можем вычислить оптимальный порог t для сегментации изображения на два

класса C_0 и C_1 (объекты и фон) таких, что класс C_0 содержит пиксели с уровнями яркости [0, 1, …, t - 1], а класс C_1 – пиксели с уровнями яркости [t, t + 1, …, L - 1].

Вероятности P_0, P_1 и средние значения μ_0, μ_1 яркости классов C_0, C_1 вычислим по следующим формулам:

$$P_0 = \sum_{i=0}^{t-1} p_i = P(t), \qquad P_1 = \sum_{i=t}^{L-1} p_i = 1 - P(t),$$

$$\mu_0 = \frac{1}{P_0}\sum_{i=0}^{t-1} ip_i = \frac{\mu(t)}{P(t)}, \qquad \mu_1 = \frac{1}{P_1}\sum_{i=t}^{L-1} ip_i = \frac{\mu_T - \mu(t)}{1 - P(t)},$$

$$\mu_T = \sum_{i=0}^{L-1} ip_i\,,$$

где μ_T – средняя яркость всего изображения, и для любого t справедливы равенства $P_0\mu_0 + P_1\mu_1 = \mu_T$ и $P_0 + P_1 = 1$.

Дисперсии σ_0^2 и σ_1^2 классов C_0 и C_1 вычислим по формулам

$$\sigma_0^2 = \sum_{i=0}^{t-1} \frac{(i-\mu_0)^2 p_i}{P_0}, \qquad \sigma_0^2 = \sum_{i=t}^{L-1} \frac{(i-\mu_1)^2 p_i}{P_1}.$$

Для определения «лучшего» порога (на уровне t) используем следующие меры разделимости классов, зависящие от порога t:

$$\lambda = \frac{\sigma_B^2}{\sigma_P^2}, \qquad \kappa = \frac{\sigma_T^2}{\sigma_P^2}, \qquad \eta = \frac{\sigma_B^2}{\sigma_T^2}, \tag{1}$$

где

$$\sigma_P^2 = P_0\sigma_0^2 + P_1\sigma_1^2, \tag{2}$$

$$\sigma_B^2(t) = P_0(\mu_0 - \mu_T)^2 + P_1(\mu_1 - \mu_T)^2 = P_0P_1(\mu_1 - \mu_0)^2, \tag{3}$$

$$\sigma_T^2 = \sum_{i=0}^{L} (i - \mu_T)^2 p_i. \tag{4}$$

Здесь по формуле (2) определена дисперсия внутри класса (взвешенная сумма дисперсий двух классов, разделенных порогом t), по формуле (3) – дисперсия

между классами и по формуле (4) – общая (полная) дисперсия (не зависит от *t*), для которой справедливо равенство

$$\sigma_T^2 = \sigma_P^2 + \sigma_B^2.$$

Тогда исходную задачу сегментации изображения на два класса C_0 и C_1 (объекты и фон) сведем к задаче оптимизации значения порога *t*, находящей максимум одной из целевых функций (1). Такая задача удовлетворяет гипотезе о том, что классы с «лучшим» пороговым значением будут разделены по уровням серого, и наоборот, порог, обеспечивающий наилучшее разделение классов по уровням серого, будет «лучшим» порогом.

Здесь стоит отметить следующее:

1) целевые функции (1) являются эквивалентными друг к другу и могут быть выражены через λ, например, $\kappa = \lambda + 1$ и $\eta = \lambda/(\lambda + 1)$;
2) так как для вычисления дисперсии внутри класса σ_P^2 необходимо использовать статистику второго порядка (дисперсии классов), а для вычисления дисперсии между классами σ_B^2 необходима статистика первого порядка (средние классов), то целевая функция η является наиболее простой мерой, зависящей от величины порога *t*.

Таким образом, оптимальный порог t^*, который максимизирует целевую функцию η, определим как

$$t^* = \arg\Big(\max_{0<t<L-1} \eta(t)\Big) = \arg\Big(\max_{0<t<L-1} \sigma_B^2(t)\Big), \qquad (5)$$

где

$$\eta(t) = \frac{\sigma_B^2(t)}{\sigma_T^2}, \quad \sigma_B^2(t) = \frac{\big(\mu_T P(t) - \mu(t)\big)^2}{P(t)\big(1 - P(t)\big)}.$$

Дисперсия в данном контексте служит показателем того, насколько сильно разнятся значения яркости пикселей на изображении от среднего значения яркости. Если значение дисперсии велико, то это означает, что пиксели на изображении существенно различаются в яркости.

При решении задачи (5) – определение оптимального порога – выбирают такой порог, который максимально увеличит различия между объектами и

фоном на изображении. Другими словами, определяют такой порог, где разница в яркости между объектами и фоном наибольшая, и получают два изолированных класса пикселей в одном бинаризованном изображении.

Таким образом, по методу Оцу для бинарных изображений порог размещают между средними значениями яркости объектов и фона так, чтобы максимизировать межклассовую дисперсию (3).

В итоге алгоритм метода Оцу состоит из следующих шагов.

1. *Построение гистограммы яркостей*. Вычислить гистограмму яркостей пикселя изображения и вероятность для каждого уровня яркости $p_i,\ i = 0, \dots, L-1$.
2. *Определение начальных значений вероятностей и средних*. Вычислить начальные значения P_0, P_1 и μ_0, μ_1 при $t = 0$.
3. *Вычисление новых значений всех параметров*. Для каждого значения порога от $t = 1$ до максимальной яркости (равной 255):
   a) Обновить значения P_0, P_1 и μ_0, μ_1.
   b) Вычислить $\sigma_B^2(t)$.
   c) Если новое значение $\sigma_B^2(t)$ больше, чем предыдущее, запомнить $\sigma_B^2(t)$ и значение порога t.
4. *Определение оптимального порога бинаризации*. Найденный оптимальный порог t будет соответствовать максимуму $\sigma_B^2(t)$.

Пороговую бинаризацию часто используют в обработке изображений для решения задачи сегментации. Например, применяют пороговую бинаризацию для выделения текста на странице или для выделения объектов с одинаковым цветом фона на изображении.

Для применения глобальной бинаризации и автоматического выбора порога методом Оцу можем воспользоваться функцией cv2.threshold из библиотеки OpenCV. Первый аргумент – это исходное изображение, которое должно быть изображением в градациях серого. Второй аргумент – это пороговое значение, которое используется для классификации значений

пикселей. Третий аргумент – это максимальное значение, которое присваивается значениям пикселей, превышающим пороговое значение. Четвертый аргумент – тип порогового значения, который определяется встроенными типами библиотеки OpenCV. Метод возвращает два значения. Первое – это использованный порог, а второе значение – изображение с пороговым значением.

Однако, следует учитывать, что пороговая бинаризация может не дать хорошего результата, когда на изображении есть области с разной освещенностью. В этом случае пороговые значения для различных областей изображения будут отличаться, и тогда более эффективным методом может быть использование *адаптивной бинаризации*, который позволяет определять пороговое значение для каждой области изображения отдельно.

Адаптивная бинаризация

Адаптивная бинаризация – метод обработки изображений, который определяет пороговое значение для каждой области изображения отдельно, в зависимости от особенностей освещения и контрастности в каждой из них. Этот подход обеспечивает более точные результаты по сравнению с пороговой бинаризацией, где одно и то же пороговое значение применяют для всего изображения.

Метод адаптивной бинаризации изображения впервые описан в работе «Adaptive Thresholding for the DigitalDesk» в 1993 году [2]. Позже в 2007 году предложен усовершенствованный алгоритм в работе [3].

Пусть изображение в оттенках серого представлено в виде двумерного массива яркостей f, где каждый пиксель с координатами (i, j) принимает значение из отрезка от 0 до 255. Исходное изображение разделено на непересекающиеся k-блоков размером $(2D + 1) \times (2D + 1)$ с центром в точках (i_k, j_k). Тогда для каждого блока можем вычислить пороговое значение T_k, как среднее значение локального распределения яркости по формуле

$$T_k = \frac{1}{(2D+1)^2} \sum_{i=-D}^{D} \sum_{j=-D}^{D} f\,(i_k + i, j_k + j)$$

или как среднее значение минимального и максимального значений яркости по формуле

$$T_k = \frac{M_k + m_k}{2},$$

$$M_k = \max_{-D \le i,j \le D} f(i_k + i, j_k + j),$$

$$m_k = \min_{-D \le i,j \le D} f(i_k + i, j_k + j),$$

где M_k – максимальное значение яркости пикселей в блоке k, m_k – минимальное значение яркости пикселей в блоке k.

Так как для каждого блока k известно пороговое значение T_k, то можем осуществить операцию бинаризации внутри каждого блока по следующей формуле:

$$F(i,j) = \begin{cases} 1, \text{если } f(i,j) < T_k - C, \\ 0, \text{иначе.} \end{cases}$$

где C – некоторая константа, которая используется для точной настройки порогового значения T_k, $F(i,j)$ – новое бинаризованное изображение.

Процесс адаптивной бинаризации состоит из следующих шагов.

1. *Разделение на блоки*. Разделить изображение на небольшие неперекрывающиеся блоки.
2. *Вычисление порогового значения для каждого блока*. Обычно для этого используются методы вычисления статистических параметров яркости пикселей в блоке, таких как среднее значение или медиана.
3. *Определение относительной яркости пикселей*. Рассмотреть каждый пиксель изображения и сравнить его яркость с пороговым значением блока, к которому он принадлежит.
4. «*Бинаризация» пикселей*. Если яркость пикселя выше порогового значения блока, то ему присвоить белый цвет, а если ниже – черный цвет.

Одним из главных преимуществ адаптивной бинаризации является возможность учитывать неоднородность освещения на изображении, что обычно является проблемой для пороговой бинаризации.

Однако необходимо отметить, что метод адаптивной бинаризации требует вычислительных ресурсов, поскольку для каждого блока изображения необходимо вычислить порог и выполнить бинаризацию. Поэтому выбор размера блоков и метода вычисления порога может влиять на точность и эффективность данного метода.

Для выполнения адаптивной бинаризации применим метод cv2.adaptiveThreshold. Помимо переменных, используемых в cv2.threshold, этот метод принимает три входных параметра:

- cv2.ADAPTIVE_THRESH_MEAN_C – пороговое значение, равное разности среднего значения площадей окрестностей (в пикселях) и константы C,
- blockSize – размер рассматриваемой области (в пикселях),
- C – величина, которая вычитается из средней или взвешенной суммы пикселей области.

На рисунке 2 приведены примеры изображений, обработанных различными методами бинаризации.

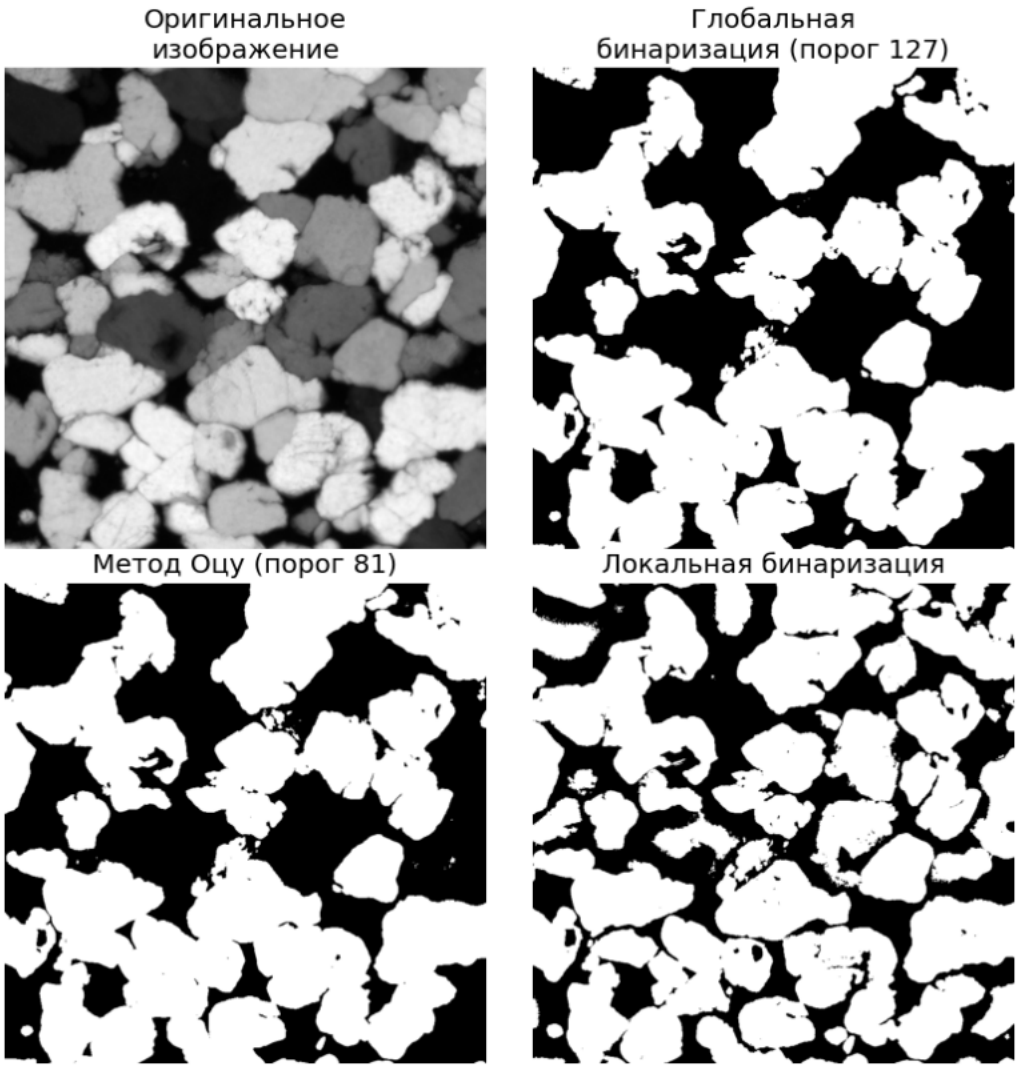


Рисунок 2 – Пример работы методов бинаризации

Контрольные вопросы

1. Чем отличается глобальная бинаризация от локальной?
2. В чем преимущество использования метода Оцу?
3. Перечислите основные шаги метода Оцу.
4. На каких типах изображений имеет преимущество адаптивная бинаризация?

Лабораторная работа

1. Напишите программу для бинаризации изображения, используя функции cv2.threshold и cv2.adaptiveThreshold. Сравните визуально полученные

результаты (библиотека matplotlib) и сделайте выводы о методах бинаризации.

2. Напишите программу, которая будет выполнять глобальную бинаризацию изображения без использования метода cv2.threshold. Программа должна принимать на вход путь к файлу-изображению и пороговое значение, а сохранять бинаризованное изображение в другом файле.
3. Создайте простое графическое приложение с использованием библиотеки Tkinter, которое позволит пользователю выбирать изображение, метод бинаризации и соответствующие параметры, а затем выводить результат на экран.
4. Реализуйте глобальную бинаризацию изображений по представленному выше алгоритму. Сравните результаты работы алгоритма и встроенной функции cv2.threshold.
5. Реализуйте адаптивную бинаризацию изображений по представленному выше алгоритму. Проанализируйте результаты для различных значений C. Сравните результаты работы алгоритма и встроенной функции cv2.adaptiveThreshold.

Глава 2. Методы сегментации на основе регионов

Методы сегментации на основе регионов применяют для разделения изображения на несколько связанных областей (регионов). Пусть X обозначает всю область изображения (множество пикселей изображения), $f(x, y)$ – функцию яркости, определенную на X, а P – логический предикат, определенный на подмножествах $S = \{S_i\}, i = \overline{1,n}$ множества X как

$$P(S) = \begin{cases} true, & \text{если } \exists \text{ const } a\text{: } |f(x,y) - a| < \varepsilon, \text{для всех } (x,y) \in S, \\ false, & \text{иначе,} \end{cases}$$

где ε – заданная допустимая ошибка. Тогда сегментацию изображения можем определить как разбиение X на подмножества (регионы) S_1, S_2,…, S_n – такие, что

1) каждый пиксель изображения должен находиться в определенном регионе, т.е. вся область X должна быть сегментирована
   i. $\bigcup_{i=1}^{n} S_i = X$;
2) регионы представляют собой связанные области, т.е. каждый регион $S_i, i = \overline{1,n}$ состоит из смежных пикселей;
3) регионы не пересекаются
4) $S_i \cap S_j = \emptyset$ для всех $i, j = \overline{1,n},\ i \neq j$;
5) пиксели одного региона однородны $P(S_i) = true$ для $i = \overline{1,n}$, т.е. все пиксели в S_i имеют одинаковые оттенки серого для изображения, представленного в градациях серого;
6) регионы S_i и S_j являются различными в смысле предиката
   $$P(S_i \cup S_j) = false \text{ для } i \neq j.$$

Далее рассмотрим основные методы сегментации изображений на основе регионов, а именно Region Growing, Split-and-Merge и Watershed Segmentation.

Метод сегментации Region Growing

Метод сегментации Region Growing (разрастания или роста региона) впервые предложен в работе [4] в 1968 году и основан на идее объединения

пикселей в область (регион), которая содержит схожие элементы (пиксели) и имеет достаточно однородное статистическое распределение яркости пикселей.

Алгоритм начинают с выбора начального пикселя (зерна) в изображении и пошагово добавляют к нему соседние пиксели, которые подходят по какому-то критерию сходства. Таким образом, получают растущий регион, состоящий из пикселей «схожего содержания».

Алгоритм метода Region Growing содержит следующие этапы.

1. *Выбор начальной точки*. Выбрать один пиксель (x_c, y_c), например, как центральный пиксель всего изображения или самый максимальный пик на гистограмме изображения. Этот пиксель будет служить отправной точкой для роста региона.
2. *Выбор «критерия сходства»*. Выбрать критерий, который определяет, насколько рассмотренные пиксели похожи на пиксели растущего региона S. Например, на основе заданного порогового значения T для яркости пикселя (x', y')

$$P(x', y') = \begin{cases} true, & \text{если } f(x', y') < T, \\ false, & \text{иначе,} \end{cases}$$

или заданного порогового значения T на отклонение яркости любого пикселя $f(x', y')$ относительно средней яркости пикселей в регионе S

$$P\big(S, (x', y')\big) = \begin{cases} true, & \text{если } \left| f(x', y') - \dfrac{1}{D} \sum\limits_{(x,y) \in S} f(x, y) \right| < T, \\ false, & \text{иначе,} \end{cases}$$

где D – количество пикселей в регионе S. Также в качестве критерия сходства можно использовать заданное пороговое значение T для соседних пикселей (x', y') и $(x, y) \in S$:

$$P\big(S, (x', y')\big) = \begin{cases} true, & \text{если } |f(x', y') - f(x, y)| < T, \forall (x, y) \in S \text{ и } (x', y') \in N(x, y), \\ false, & \text{иначе,} \end{cases}$$

где $N(x, y)$ – множество соседних пикселей вокруг (x, y). Если значение предиката P будет истинным, то пиксель добавляется к региону S.

3. *Итерационный процесс*. Процесс (этапы 1-2) продолжить до тех пор, пока регион не перестанет расширяться, то есть, пока не перестанут добавляться новые точки в регион или не будет достигнут заданный размер региона. При этом на каждой новой итерации алгоритма необходимо рассматривать пиксели, лежащие на границе региона S, но не принадлежащие самому этому региону.

Вышеизложенный алгоритм метода Region Growing используют для выделения одного региона, однако применяя его последовательно или одновременно для нескольких регионов, можем получить разбиение всего изображения. Такой подход к сегментации изображений называют методом Seed Region Growing (разрастания областей из зёрен) [5].

Процесс сегментации изображения начинают проводить от первоначально заданного набора зёрен, а именно от начального состояния множеств S_1, S_2,…, S_n, которые содержат, например, по одному зерну (пикселю). Однако, если изображение содержит шумы, то одиночные точечные зерна могут попасть на атипичный пиксель (т.е. на статистический выброс). Это может привести к плохой начальной оценке среднего значения этого региона и неправильной сегментации. Для устранения такой проблемы рекомендуют использовать небольшие зоны зерна размером 2x2 или 4x4 (вместо одиночных пикселей), при этом каждая зона зерна должна быть достаточно большой, чтобы обеспечить стабильную оценку среднего значения ее региона.

Каждый шаг алгоритма предполагает добавление одного пикселя к одному из регионов. Предположим, что было выполнено m шагов алгоритма. Теперь рассмотрим состояние множеств S_i на шаге m+1. Пусть B – набор всех еще нераспределенных пикселей, граничащих хотя бы с одним из регионов

$$B = \left\{ (x,y) \notin \bigcup_{i=1}^{n} S_i \mid N(x,y) \cap \bigcup_{i=1}^{n} S_i \neq \emptyset \right\},$$

где $N(x,y)$ – множество «соседей» пикселя (x,y). Если для пикселя $(x,y) \in B$ множество $N(x,y)$ соответствует только одному S_i, то определим значение

$i(x,y)$ из интервала 1..n так, чтобы $N(x,y) \cap S_{i(x,y)} \neq \emptyset$ (i представляет собой индекс S_i), и определим $\delta(x,y)$ как меру того, насколько пиксель (x,y) отличается от региона, к которому он примыкает. Например, $\delta(x,y)$ можем вычислить как

$$\delta(x,y) = \left| f(x,y) - \underset{(\bar{x},\bar{y}) \in S_{i(x,y)}}{\text{mean}} [f(\bar{x},\bar{y})] \right|. \quad (6)$$

Если $N(x,y)$ соответствует двум или более S_i, то выберем $i(x,y)$ с минимальным $\delta(x,y)$. Затем найдём пиксель $(x',y') \in B$ такой, что

$$(x',y') = \underset{(x,y) \in B}{\text{argmin}}\, \delta(x,y) \quad (7)$$

и добавим (x',y') к $S_{i(x',y')}$.

На этом шаг m+1 считаем завершенным. Процесс повторяем до тех пор, пока не будут выделены все пиксели. Формулы (6) и (7) гарантируют, что окончательная сегментация на регионы будет максимально однородной с учетом ограничения связности.

Псевдокод алгоритма выглядит следующим образом.

Пометить начальные точки согласно их принадлежности регионам $S_i, i = \overline{1,n}$.

Добавить соседние пиксели, граничащие с регионами, в список SB, отсортированный по значению функции δ.

Пока список listB не пуст:

извлечь из списка первую точку (x',y');

если все соседи точки (x',y') принадлежат единому региону S_i, **то:**

пометить точку (x',y') как принадлежащую региону S_i;

обновить среднее значение яркости пикселей региона S_i;

добавить в список listB сохраняя порядок всех соседей (x',y'), которые не принадлежат ни одному из регионов и не являются граничными точками;

в противном случае

пометить точку (x',y') граничной точкой.

Метод Region Growing показывает хорошие результаты на простых изображениях с однородными регионами, однако могут возникнуть проблемы при сегментации более сложных изображений, где есть много перекрывающихся регионов.

На рисунке 3 приведен пример выделения одного объекта методом Region Growing. В качестве отправной точки алгоритма выбран центральный пиксель, а критерием сходства – пороговое значение равное 80 (значение выбрано экспериментально).

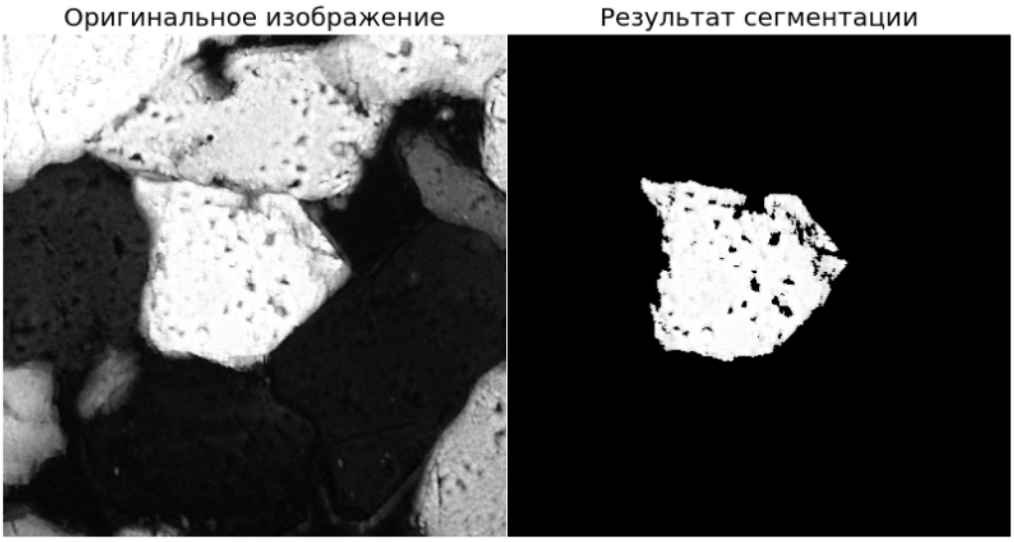


Рисунок 3 – Пример выделения одного региона с помощью Region Growing

Метод сегментации изображений Split-and-Merge

Метод сегментации изображений Split-and-Merge впервые разработан в 1976 году и опубликован в работе [6]. Данный метод предназначен для разделения изображения на мелкие сегменты, а затем их объединения в более крупные сегменты, основываясь на определенных критериях.

Основная идея метода Split-and-Merge заключается в следующем.

1. *Разделение изображения (Split).* На первом этапе изображение X размером $W \times H$ представляет собой одну область, которую необходимо разделить на четыре равные части (сегменты). Для этого найдем координаты центральной точки по формуле

$$x_c = \frac{x_{top-left} + x_{bottom-right}}{2}, \qquad y_c = \frac{y_{top-left} + y_{bottom-right}}{2}, \quad (8)$$

где $(x_{top-left}, y_{top-left})$ – координаты левого верхнего угла изображения, $(x_{bottom-right}, y_{bottom-right})$ – координаты правого нижнего угла, и определим координаты верхнего левого и правого нижнего углов четырех сегментов следующим образом:

$$\begin{gathered} S_1: (x_{top-left}, y_{top-left}) \to (x_c, y_c), \\ S_2: (x_c, y_{top-left}) \to (x_{bottom-right}, y_c), \\ S_3: (x_{top-left}, y_c) \to (x_c, y_{bottom-right}), \\ S_4: (x_c, y_c) \to (x_{top-left}, y_{top-left}), \end{gathered} \quad (9)$$

Таким образом, каждый сегмент будет иметь размер $W/2 \times H/2$, как на рисунке ниже

S_1	S_2
S_3	S_4

Далее каждый сегмент проверим на однородность. Для этого используем критерий однородности: стандартное отклонение яркости пикселей сегмента S_i меньше некоторого порогового значения T

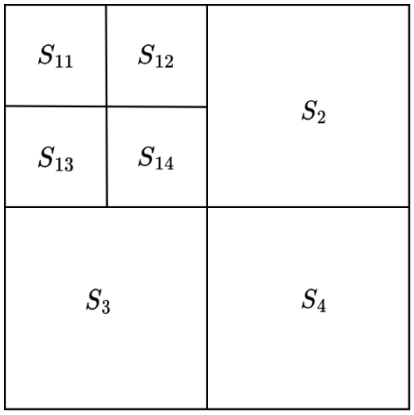
$$\sigma(S_i) = \sqrt{\frac{1}{D} \sum_{(x,y) \in S_i} [f(x,y) - \mu(S_i)]^2} \leq T,$$

где D – количество пикселей в сегменте S_i, $\mu(S_i)$ – среднее значение яркости пикселей в сегменте S_i; или другой критерий однородности: модуль разности значения яркости центрального пикселя и среднего

значения яркости пикселей всего сегмента меньше некоторого порогового значения T_{diff}

$$|f(x_c, y_c) - \mu(S_i)| < T_{diff}, \qquad (x_c, y_c) \in S_i,$$

где T_{diff} – допустимый порог различия яркостей $f(x_c, y_c)$ и $\mu(S_i)$. Если критерий однородности не выполняется ($P(S_i) = false$), то сегмент делится на более мелкие части по формулам (8) и (9). Допустим, что для сегмента S_1 этот критерий $P(S_1) = false$, тогда S_1 будет повторно разделен на более мелкие сегменты как на рисунке ниже.



Этот процесс повторяем для каждого созданного подсегмента до тех пор, пока не будут получены сегменты, в которых все пиксели имеют близкую интенсивность.

2. *Объединение изображений (Merge).* На втором этапе происходит объединение соседствующих сегментов, если два или более связанных сегментов удовлетворяют определенному критерию схожести. Пусть S_i и S_j, $i \neq j$ – два соседних сегмента, тогда объединение $S_i \cup S_j$ можем выполнить, если модуль разности средних значений яркости пикселей этих двух сегментов не превышает некоторого порога T_m

$$\left|\mu(S_i) - \mu\left(S_j\right)\right| < T_m, \qquad i \neq j.$$

Таким образом, окончательно получим сегментированное изображение.

Приведем псевдокод алгоритма Split and Merge.

// этап Split

Поместить в стек всё изображение (считаем, что изображение – самый большой сегмент).

Пока стек не пуст:

взять сегмент S_i из стека;

вычислить критерий однородности S_i;

если $P(S_i)$ == false (сегмент S_i неоднородный), то:

разделить S_i на более малые сегменты и **поместить** их в стек;

в противном случае

оставить сегмент таким.

// этап Merge

Пока стек не пуст:

взять из стека сегмент S_i для всех соседних сегментов S_j:

вычислить критерий однородности S_i и S_j;

если сегменты S_i и S_j однородные, **то:**

объединить $S' = S_i \cup S_j$;

поместить S' в стек;

удалить S_i и S_j из стека;

в противном случае

взять другой соседний сегмент S_j.

На рисунке 4 представлен пример сегментации изображения методом Split-and-Merge.

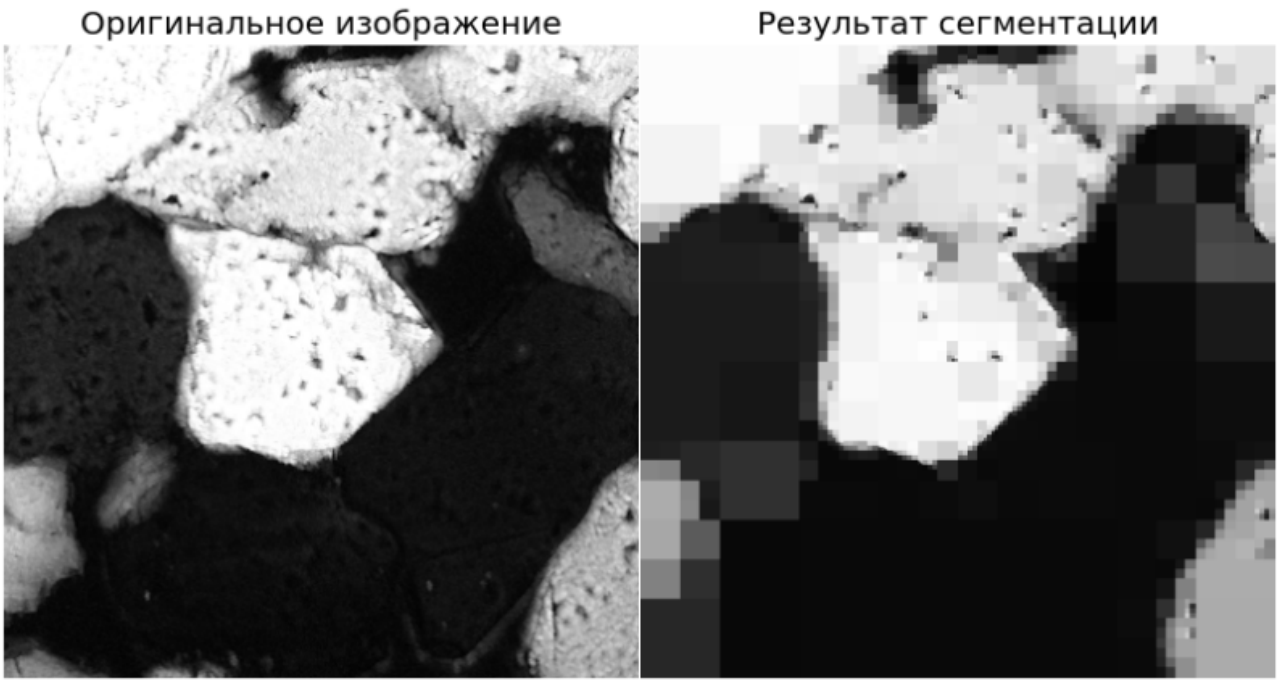


Рисунок 4 – Пример сегментации изображения методом Split-and-Merge

Сегментация методом водораздела

Метод водораздела (англ. Watershed Segmentation) [7] считают одним из самых популярных алгоритмов сегментации изображений и используют для разделения похожих соприкасающихся друг с другом объектов на изображении.

Объясним подробно суть данного алгоритма. Изображение в градациях серого образно представим как топографическую карту. На этой карте (изображении) значения пикселей с высокой интенсивностью обозначают возвышенности (белые области), тогда как значения с низкой интенсивностью обозначают долины – локальные минимумы (черные области).

Теперь начнем заполнять всю долину водой, и через некоторое время вода, поступающая из разных долин, начинает сливаться. Чтобы этого избежать, нужно построить барьеры в местах слияния воды, поэтому барьеры называют линиями бассейна и используют для определения границ сегмента. Когда вода наполнит всю территорию, вплоть до самого высокого пика, то заливку водой остановим. В конце процесса будут видны только линии водораздела (построенные барьеры), и это будет окончательным результатом сегментации.

Таким образом, целью алгоритма водораздела является построение линий водораздела, которые и будут сегментировать изображение.

Алгоритм водораздела содержит следующие этапы:

1. Получить бинарное изображение.
2. Вычислить преобразование расстояния.
3. Найти локальные точки максимума.
4. Обозначить метки.

Рассмотрим каждый этап на примерах кода, реализованного на языке Python. Сначала импортируем необходимые библиотеки.

```
import numpy as np
import cv2
import matplotlib.pyplot as plt
from scipy import ndimage as ndi
from skimage.segmentation import watershed
from skimage.feature import peak_local_max
from skimage import morphology
from skimage.measure import label
```

Для получения бинарного изображения, которое будет «удобно» сегментировать, необходимо провести предобработку исходного изображения. С этой целью реализуем необходимые для предобработки изображения функции. Ниже приведены методы для выполнения морфологических операций: эрозии (уменьшения изображения) и дилатации (увеличения изображения).

```
def erode(img, k_s=3, it=1):
    kernel = np.ones((k_s, k_s), 'uint8')
    erode_img = cv2.erode(img, kernel, iterations=it)
    return erode_img
def dilate(img,k_s=3,it=1):
    kernel = np.ones((k_s, k_s), 'uint8')
    dilate_img = cv2.dilate(img, kernel, iterations=it)
```

```
    return dilate_img
```

Эрозия – это математическая операция, которую используют для сужения или уменьшения объектов на изображении X. Пусть задан структурный элемент Z. Эрозией множества X называют множество Y, состоящее их тех элементов исходного множества X, для которых выполняется условие $Z_x \subseteq X$, т. е.

$$Y = X \ominus Z = \{x: Z_x \subseteq X\},$$

где $\ominus$ – логическая операция AND между изображением и структурным элементом. Центр структурного элемента помещают во все точки $x \in X$, если элемент полностью принадлежит X, тогда точка $x \in Y$.

Другими словами, структурный элемент Z представляет собой небольшую геометрическую фигуру, такую как точка или квадрат, который проходит по изображению. Если яркости всех пикселей объекта на изображении совпадают с яркостью пикселей структурного элемента, то эти пиксели сохраняются, в противном случае они удаляются. Таким образом, результатом эрозии является сужение объектов на изображении.

Дилатация – это математическая операция, которую используют для расширения или увеличения объектов на изображении X. Пусть задан структурный элемент Z. Дилатацией множества X называют множество Y и определяют формулой

$$Y = X \oplus Z = \{x: Z_x \cap X = \emptyset\},$$

где $\oplus$ – логическая операция OR между изображением и структурным элементом. Центр структурного элемента помещают во все точки $x \in X$, если хотя бы один пиксель элемента принадлежит X, тогда точка $x \in Y$. Объединение полученных точек определяет множество Y.

Другими словами, структурный элемент также проходит по всему изображению, и если хотя бы яркость одного пикселя объекта на изображении совпадает с яркостью пикселя структурного элемента, то все пиксели на изображении, которые покрывает структурный элемент, считают частью этого объекта. Таким образом, результатом дилатации является расширение объектов на изображении.

Эрозию и дилатацию широко используют в области обработки изображений для улучшения качества изображения или выделения интересующих областей. Например, эрозию применяют для удаления шума или неправильных пикселей, а дилатацию – для заполнения пробелов или увеличения размера объектов.

Эти операции часто комбинируют вместе для достижения желаемого результата. Например, применение эрозии перед дилатацией позволяет удалить мелкие дефекты и затем расширить оставшиеся области. Эту операцию называют открытием. С другой стороны, применение дилатации перед эрозией может помочь заполнить пробелы и затем сузить области. Такую операцию называют закрытием.

Эрозия и дилатация являются основными операциями в математической морфологии[1]. Они играют важную роль во многих приложениях, таких как распознавание образов, сегментация изображений, а также в медицинском и биологическом анализе данных.

Ниже приведена функция для удаления небольших объектов на бинарном изображении. Используем метод morphology.remove_small_objects, который на вход принимает изображение с метками областей от 1 до n (количество областей на изображении).

```
    def del_small_areas(thresh,area_black = 100, area_white=100):
        result = morphology.remove_small_objects(label(thresh),
area_white,)
        result[result>0]=255
        result = morphology.remove_small_objects(label(255-
result), area_black,)
        result[result>0]=255
        result = 255 - result
        return result
```

[1] Морфология - техника обработки и анализа изображений.

Проводим локальную бинаризацию изображения с последующей обработкой (эрозия, дилатация и удаление мелких объектов).

```
img = cv2.imread('data/image.png', cv2.IMREAD_GRAYSCALE)
mask_image = cv2.imread('data/mask.png',cv2.IMREAD_GRAYSCALE)
thresh_local_1 = cv2.adaptiveThreshold(img, 255,
cv2.ADAPTIVE_THRESH_MEAN_C, cv2.THRESH_BINARY_INV, 101, 10)
thresh_local = del_small_areas(erode(dilate(thresh_local_1)))
```

На рисунке 5 показаны исходное изображение, результат локальной бинаризации до и после предобработки, а также маска изображения.

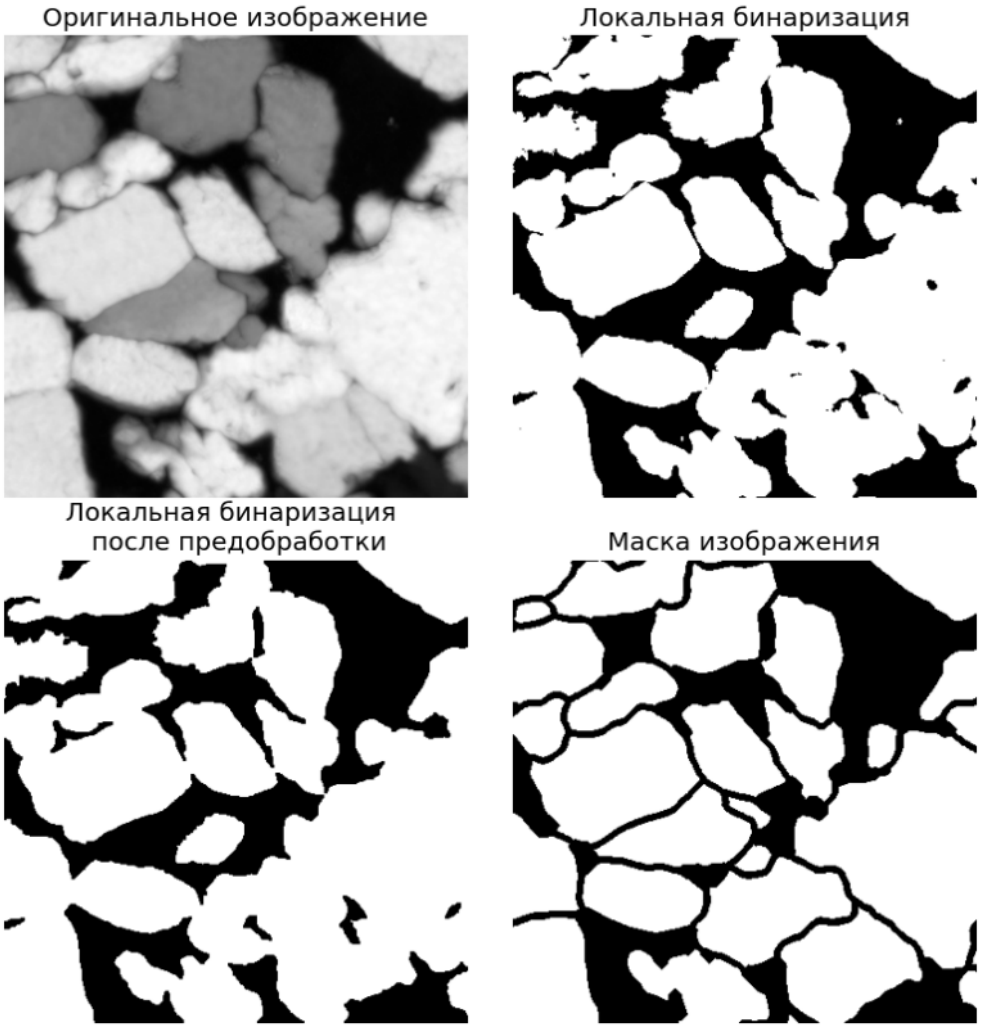


Рисунок 5 – Результат локальной бинаризации

Теперь разделим соприкасающиеся объекты и создадим границу между ними так, чтобы она располагалась как можно дальше от центров перекрывающихся объектов. Для этого используем алгоритм, называемый преобразованием расстояния.

Пусть I – бинарное изображение, тогда преобразованное изображение D_I в каждой своей точке (x, y) определяется по формуле

$$D_I(x,y) = \min_{(\bar{x},\bar{y})\in I} \sqrt{(x-\bar{x})^2 + (y-\bar{y})^2}, \quad (10)$$

где $(\bar{x}, \bar{y})$ – фоновые пиксели изображения I.

Таким образом, алгоритм на входе получает двоичное изображение I, а на выходе формирует изображение D_I, которое содержит информацию (в виде интенсивности пикселей) о расстоянии от текущего пикселя до ближайшего пикселя с нулевой интенсивностью (фоновый пиксель) согласно формуле (10).

Используем из библиотеки SciPy функцию distance_transform_edt(), которая преобразует исходное изображение в изображение «преобразование расстояния», используя классическую евклидову метрику:

```
distance = ndi.distance_transform_edt(thresh_local).
```

Приведем на рисунке 6 исходное изображение, бинарное изображение и «преобразование расстояния».

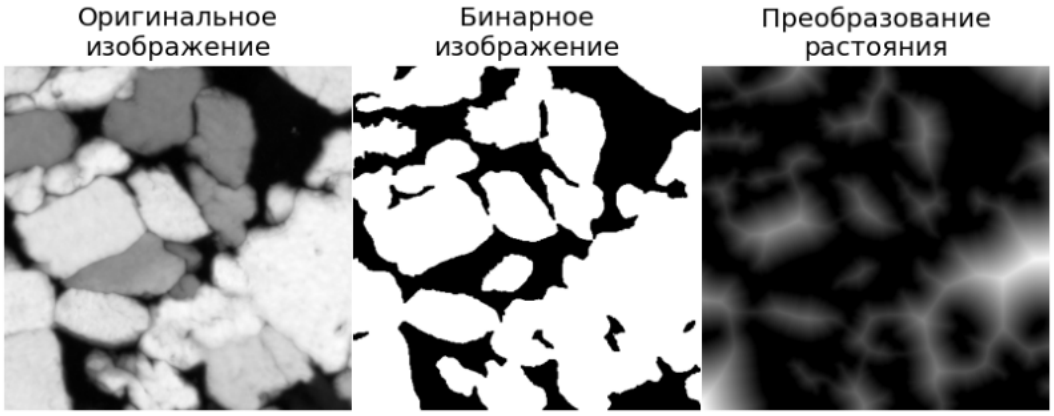


Рисунок 6 – Преобразование расстояния

Здесь стоит отметить, что в «преобразовании расстояния» можно увидеть «белые пиксели», эти пиксели имеют большие значения преобразования расстояния (другими словами, большие расстояния до фоновых пикселей).

Теперь найдем координаты пиков (локальных максимумов) белых областей на изображении. Для этого применим функцию peak_local_max() из библиотеки scikit-image к построенному изображению и получим маркеры, которые будут использованы в функции водосбора.

```
local_max = peak_local_max(distance, min_distance=50,
footprint=np.ones((3, 3)), labels=thresh_local)
```

На следующем шаге выполним разметку полученных маркеров для функции водораздела. Для этого используем функцию ndi.label() из библиотеки SciPy. Эта функция содержит один входной параметр, который представляет собой массив. Любые ненулевые значения этого параметра считаются признаками, а нулевые значения – фоном. В нашей программе будем использовать координаты рассчитанного локального максимума. Переведем эти координаты в двумерный массив и функцией ndi.label() случайным образом пометим все локальные максимумы разными положительными значениями, начиная с 1. Каждый из *N* полученных объектов на изображении помечается целым значением от 1 до *N*.

```
mask = np.zeros(distance.shape, dtype=bool)
mask[tuple(local_max.T)] = True
markers, _ = ndi.label(mask)
```

На последнем шаге применим функцию skimage.segmentation.watershed() из библиотеки scikit-image. В качестве входных параметров зададим инвертированное изображение преобразования расстояния и маркеры, которые вычислены ранее в коде, приведенном выше. Поскольку алгоритм водораздела предполагает, что маркеры представляют собой локальные минимумы, то нужно инвертировать изображение «преобразования расстояния». Таким образом, светлые пиксели будут представлять больш**и**е возвышения, а темные пиксели – низкие возвышения для преобразования водораздела:

```
labels = watershed(-distance, markers, mask=thresh_local).
```

Далее при помощи функций cv2.findContours и cv2.drawContours найдем и выделим границы полученных областей на исходном изображении.

```
image_countours = cv2.cvtColor(img.copy(),cv2.COLOR_GRAY2BGR)
for label in np.unique(labels):
   if label == 0:
       continue
   mask = np.zeros(thresh_local.shape, dtype="uint8")
   mask[labels == label] = 255
```

```
        cnts = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
        cnts = cnts[0] if len(cnts) == 2 else cnts[1]
        cv2.drawContours(image_countours, cnts, -1, (0,255,0), 1)
```

В следующих строках выполним визуализацию полученных результатов сегментации с помощью библиотеки Matplotlib:

```
    fig, axes = plt.subplots(ncols=4, figsize=(10, 5),
sharex=True, sharey=True)
    ax = axes.ravel()
    ax[0].imshow(gray, cmap=plt.cm.gray)
    ax[0].set_title('Оригинальное\nизображение')
    ax[1].imshow(labels, cmap=plt.cm.nipy_spectral)
    ax[1].set_title('Сегментация\nводораздела')
    ax[2].imshow(image_countours, cmap=plt.cm.nipy_spectral)
    ax[2].set_title('Найденные\nграницы')
    ax[3].imshow(mask_image, cmap=plt.cm.gray)
    ax[3].set_title('Исходная маска\nсегментации')
    for a in ax:
        a.set_axis_off()
    fig.tight_layout()
    plt.show()
```

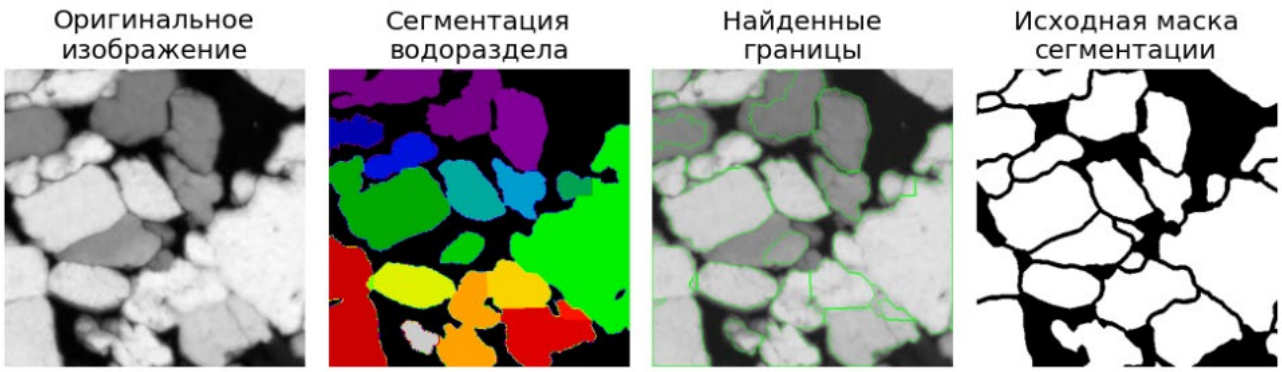


Рисунок 7 – Результат сегментации методом водораздела

На рисунке 7 приведены исходное изображение, изображение водораздела, найденные границы на изображении и (исходную) маску сегментации.

Контрольные вопросы

1. Чем метод Region Growing отличается от метода Split-and-Merge?
2. Какой принцип лежит в основе метода Watershed Segmentation?
3. Какие этапы включает алгоритм водораздела?
4. Что такое эрозия и дилатация. Для чего используются эти морфологические операции?
5. Какие библиотеки используются в предоставленном коде для обработки изображений?

Лабораторная работа

1. Реализуйте функцию, которая будет применять метод Region Growing к заданному изображению. Начните с выбора начального пикселя и итеративного добавления пикселей на основе заданного порога.
2. Изучите метод Split-and-Merge и напишите функцию, которая будет разделять и объединять сегменты на основе заданных критериев. Примените этот метод к изображению и сравните результаты с другими методами.
3. Примените метод Watershed Segmentation к изображениям. Используйте функции из предоставленного кода для обработки и анализа. Исследуйте влияние изменения параметров и методов предобработки (например, размера окна адаптивной бинаризации, пороговых значений, min_distance в методе peak_local_max и т.д.) на результаты сегментации. Сделайте выводы о том, какие параметры оказывают наибольшее влияние на результаты.

Глава 3. Методы сегментации изображений на основе границ

Оператор Собеля

Оператор Собеля (оператор Собеля-Фельдмана или фильтр Собеля) – это классический метод обработки изображений, который используют для выделения границ. Данный метод впервые представлен в 1968 году И. Собелем и Г. Фельдманом в докладе «Оператор изотропного градиента размера 3 x 3 для обработки изображений» («A 3 x 3 isotropic gradient operator for image processing» [8]) в рамках Стэнфордского проекта по искусственному интеллекту, и с тех пор стал одним из наиболее распространенных методов для обработки изображений.

Оператор Собеля – дискретный дифференциальный оператор, который применяют для вычисления приближенного значения градиента яркости в каждой точки изображения.

Пусть X – исходное изображение в оттенках серого и каждому пикселю изображения с координатами x и y соответствует некоторое значение функции яркости $f(x,y)$. Градиентом яркости $\mathrm{grad}\, f(x,y)$, т. е. градиентом функции яркости $f(x,y)$ изображения X, называют двумерный вектор вида

$$\vec{g} = (g_x, g_y) = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right),$$

где компоненты g_x и g_y представляют собой частные производные функции яркости изображения по аргументу x и по аргументу y, соответственно. Этот вектор $\vec{g}$ в каждой точке изображения показывает направление наибольшего увеличения яркости, а его длина $|\vec{g}|$ – скорость изменения яркости.

Следовательно, в результате применения оператора Собеля можем выявить пиксели изображения с резкими или плавными изменениями яркости, что в свою очередь указывает на наличие границ или контуров. Таким образом получим, что пикселям в однородной области изображения соответствует нулевой вектор, а пикселям, где происходит переход между областями

различной яркости, отвечает вектор, показывающий направление и величину изменения яркости.

Сначала для вычисления приближенных значений производных по горизонтали и вертикали выполним операцию свертки (конволюции) исходного изображения X с ядрами G_x и G_y размера 3 x 3 следующего вида:

$$G_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}, \quad G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}.$$

Операцией свертки двух матриц называют сумму произведений элементов этих матриц с одинаковыми индексами. Таким образом, свертку $g_x(x, y)$ изображения X с ядром G_x можем вычислить по формуле

$$g_x(x, y) = G_x * f(x, y) = \sum_{i=-1}^{1} \sum_{j=-1}^{1} G_x(i, j) f(x + i, x + j), \quad \forall (x, y) \in X, \qquad (11)$$

а свертку $g_y(x, y)$ изображения X с ядром G_y – согласно формуле

$$g_y(x, y) = G_y * f(x, y) = \sum_{i=-1}^{1} \sum_{j=-1}^{1} G_y(i, j) f(x + i, x + j), \quad \forall (x, y) \in X. \qquad (12)$$

В результате получим два изображения: первое изображение содержит информацию об изменении яркости пикселей в горизонтальном направлении, а второе изображение – информацию об изменении яркости пикселей в вертикальном направлении.

Далее сформируем новое изображение, для каждого пикселя которого вычислим приближенное значение величины градиента

$$|\vec{g}(x, y)| = \sqrt{g_x^2(x, y) + g_y^2(x, y)}. \qquad (13)$$

Также можем определить и направление градиента по формуле

$$\theta(x, y) = \operatorname{arctg}\left(\frac{g_y(x, y)}{g_x(x, y)}\right). \qquad (14)$$

Это новое изображение представляет собой карту градиента, где пикселям, имеющим большое приближённое значение величины градиента, соответствуют участки высокой яркости. В результате, границы объектов и

другие структуры, характеризующиеся резкими изменениями яркости, будут выделены и легко заметны на изображении.

На последнем шаге проведем бинаризацию изображения для обработки границ с помощью некоторого порогового значения T:

$$Y(x,y) = \begin{cases} 1, \text{если } |\vec{g}(x,y)| > T, \\ 0, \text{иначе.} \end{cases}$$

Таким образом получим бинаризованное изображение Y, где пиксели со значением, превышающим пороговое значение, считаем граничными.

Пример программы для выделения границ на изображении с помощью оператора Собеля представлен ниже.

```
import cv2
# загружаем изображение
img = cv2.imread('image.jpg', cv2.IMREAD_GRAYSCALE)
# применяем оператор Собеля для обнаружения границ
sobelx = cv2.Sobel(img, cv2.CV_64F, 1, 0, ksize=3)
sobely = cv2.Sobel(img, cv2.CV_64F, 0, 1, ksize=3)
# объединяем результаты
sobel = cv2.sqrt(cv2.addWeighted(cv2.pow(sobelx, 2.0), 1.0,
cv2.pow(sobely, 2.0), 1.0, 0.0))
# применяем пороговую бинаризации
sobel_thresh = cv2.threshold(sobel, 100, 255,
cv2.THRESH_BINARY)[1]
```

В этом примере загружаем изображение с помощью функции cv2.imread(), применяем оператор Собеля для обнаружения границ с помощью функций cv2.Sobel() и объединяем результаты с помощью функций cv2.sqrt() и cv2.addWeighted(). Далее применяем пороговую бинаризацию для окончательной обработки выделенных границ. На рисунке 8 приведен результат обработки изображения оператором Собеля.

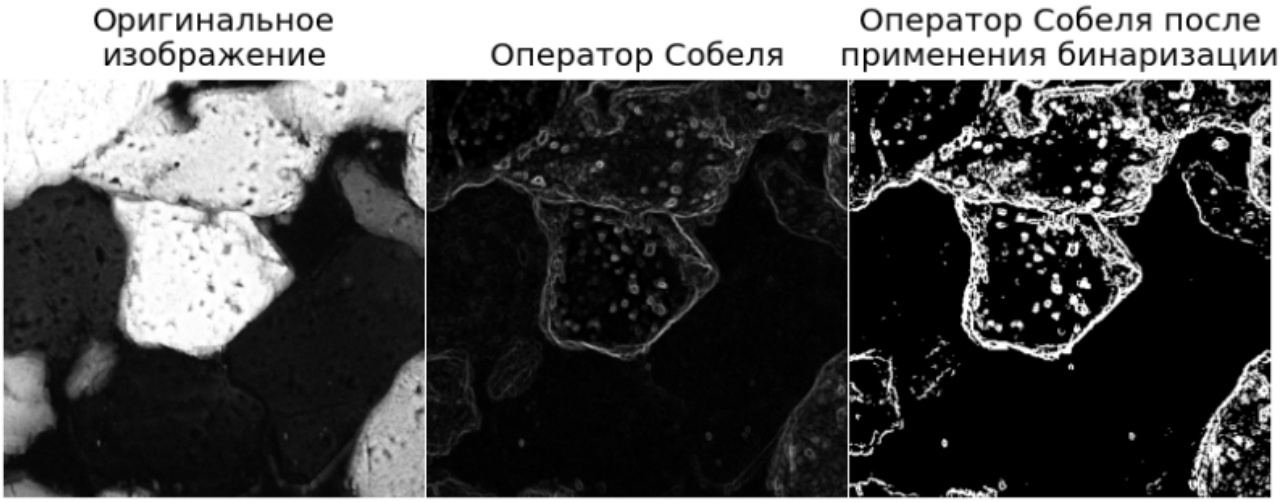


Рисунок 8 – Результат работы оператора Собеля

Следует отметить, что оператор Собеля является простым и эффективным методом обработки изображений для выделения границ. Он может использоваться как самостоятельный метод, так и в комбинации с другими методами сегментации изображений.

Однако, несмотря на свою эффективность в обнаружении границ, этот метод имеет несколько недостатков.

1. Чувствительность к шуму. Шумные пиксели могут быть обработаны как ложные границы на изображении, что приводит к неправильному результату.
2. Потеря тонких границ. Из-за использования небольших ядер (3x3) могут быть упущены тонкие детали или мелкие объекты, так как оператор Собеля предназначен для выделения более крупных границ.
3. Определение порога. Процесс определения порога для бинаризации результата может быть сложным и зависеть от конкретного контекста задачи.

Перечисленные недостатки оператора Собеля необходимо учитывать при обработке изображений.

Оператор Кэнни

Оператор Кэнни (детектор границ Кэнни, алгоритм Кэнни) – это метод обнаружения границ на изображении, который разработан Джоном Кэнни в

1986 году и изложен в работе [9]. Этот оператор является более сложным и точным, чем оператор Собеля, и отвечает трем основным критериям.

1. Хорошее обнаружение (минимизирует влияние шума на процесс обнаружения границ).
2. Хорошая локализация (точно определяет положение границ, обеспечивая высокую пространственную точность в выделении контуров объектов).
3. Единственный отклик на одну границу (не допускает появление лишних или повторяющихся выделений одной и той же границы).

Оператор Кэнни включает несколько этапов обработки изображения.

1. *Сглаживание изображения.* На первом этапе на изображение накладывают фильтр низких частот – фильтр Гаусса.

   Пусть X – исходное изображение в оттенках серого, $f(x, y)$ – функция, определяющая значение яркости пикселя с координатами x и y, а $G(x, y)$ – ядро фильтра Гаусса. Тогда сглаженное изображение получим с помощью операции свертки исходного изображения с фильтром Гаусса по формуле

$$s(x, y) = \frac{1}{2\pi\sigma^2} \sum_{i=-l}^{l} \sum_{j=-l}^{l} e^{-\frac{(i^2+j^2)}{2\sigma^2}} f(x+i, y+j), \quad \forall\, (x, y) \in X,$$

   где σ – среднеквадратичное отклонение нормального распределения, l – половинный размер фильтра, который задается как 3σ.

   В результате произойдет размытие изображения таким образом, что уменьшится контрастность между соседними пикселями, изменится распределения яркости (на изображении будут более плавные переходы яркости), что означает подавление высокочастотных компонентов (шума) на изображении.

2. *Вычисление градиентов.* На втором этапе происходит обнаружение горизонтальных, вертикальных и диагональных границ путем обработки сглаженного изображения оператором Собеля.

Для каждого пикселя изображения вычислим частные производные функции яркости в горизонтальном и вертикальном направлениях по формулам (11) и (12). После определим приближенное значение величины градиента по формуле (13), его направление по формуле (14) и полученные значения $\theta(x, y)$ округлим следующим образом:

$$\theta(x,y) \approx \begin{cases} 0°, & \text{если } 0° \le \theta < 22.5° \text{ или } 157.5° \le \theta \le 180°, \\ 45°, & \text{если } 22.5° \le \theta < 67.5°, \\ 90°, & \text{если } 67.5 \le \theta < 112.5°, \\ 135°, & \text{если } 112.5° \le \theta < 157.5°. \end{cases}$$

3. *Подавление не-максимумов.* На третьем этапе выделяют локальные максимумы по направлению градиента и подавляют пиксели, которые не являются максимальными в своих окрестностях.

Для каждого пикселя (x, y) проверим, является ли его величина градиента максимумом в направлении градиента. Если это не так, то значение пикселя уменьшим до нуля. Формально этот процесс опишем следующим образом:

- рассмотреть окрестность пикселя (x, y) вдоль направления градиента. Например, если $\theta(x, y)$ указывает на горизонтальное направление, то сравнивать значения величины градиента в соседних пикселях слева и справа;
- если значение величины градиента в пикселе (x, y) является максимальным в данном направлении, то оставить его без изменений. В противном случае, установить значение в нуль, что можем вычислить по следующей формуле:

$$\bar{g}(x,y) = \begin{cases} |\vec{g}(x,y)|, & \text{если } |\vec{g}(x,y)| \ge |\vec{g}(x+\Delta x_1, y+\Delta y_1)| \\ & \text{и } |\vec{g}(x,y)| \ge |\vec{g}(x+\Delta x_2, y+\Delta y_2)|, \\ 0, & \text{иначе,} \end{cases}$$

где Δx_1, Δy_1, Δx_2, Δy_2 определяются углом наклона $\theta(x, y)$ согласно таблице 1.

Таблица 1. Определение соседних пикселей к пикселю (x, y) по направлению градиента $\theta(x, y)$

$\boldsymbol{\theta(x, y)}$	$\mathbf{0°}$	$\mathbf{45°}$	$\mathbf{90°}$	$\mathbf{135°}$
$\boldsymbol{\Delta x_1}$	-1	-1	0	-1
$\boldsymbol{\Delta y_1}$	0	-1	-1	1
$\boldsymbol{\Delta x_2}$	1	1	0	1
$\boldsymbol{\Delta y_2}$	0	1	1	-1

Таким образом, на данном этапе уменьшим количество границ и улучшим их четкость на изображении.

4. *Выполнение двойной пороговой фильтрации и уточнение границ.* На последнем этапе формируют бинаризованное изображение Y для выделения значимых границ и определения связных компонент.

Для этого зададим два порога: верхний порог T_{high} и нижний порог T_{low}. Пиксели с величиной градиента, превышающей верхний порог, считаем граничными пикселями и обозначим как «сильные» границы, а пиксели с величиной градиента меньше нижнего порога отбросим. Отдельно рассмотрим пиксели с величиной градиента между верхним и нижним порогами. Если такой пиксель соединен с «сильной границей», то его считаем «слабой» границей. Все остальные пиксели считаем фоновыми и обнуляем. Математически это можем записать следующим образом:

$$Y(x, y) = \begin{cases} 1\ (\text{«сильная» граница}), & \text{если } \bar{g}(x, y) \geq T_{high}, \\ 1\ (\text{«слабая» граница}), & \text{если } T_{low} \leq \bar{g}(x, y) < T_{high} \\ & \text{и связан с «сильной» границей}, \\ 0, & \text{иначе.} \end{cases}$$

Ниже приведен пример кода на Python, который демонстрирует использование оператора Кэнни для сегментации изображений.

```
import cv2
# загрузка изображения и преобразование в оттенки серого
img = cv2.imread('image.jpg')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
# применение оператора Кэнни
edges = cv2.Canny(gray, 100, 200)
# поиск контуров
contours, _ = cv2.findContours(edges, cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)
# отрисовка контуров на изображении
cv2.drawContours(img, contours, -1, (0, 255, 0), 1)
# отображение результата
```

В приведенном примере загружаем изображение и преобразуем его в оттенки серого с помощью функции cv2.cvtColor(). Затем применяем оператор Кэнни с помощью функции cv2.Canny() и находим контуры с помощью функции cv2.findContours(). После этого отрисовываем контуры на исходном изображении с помощью функции cv2.drawContours(). Результат обработки изображения оператором Кэнни приведен на рисунке 9.

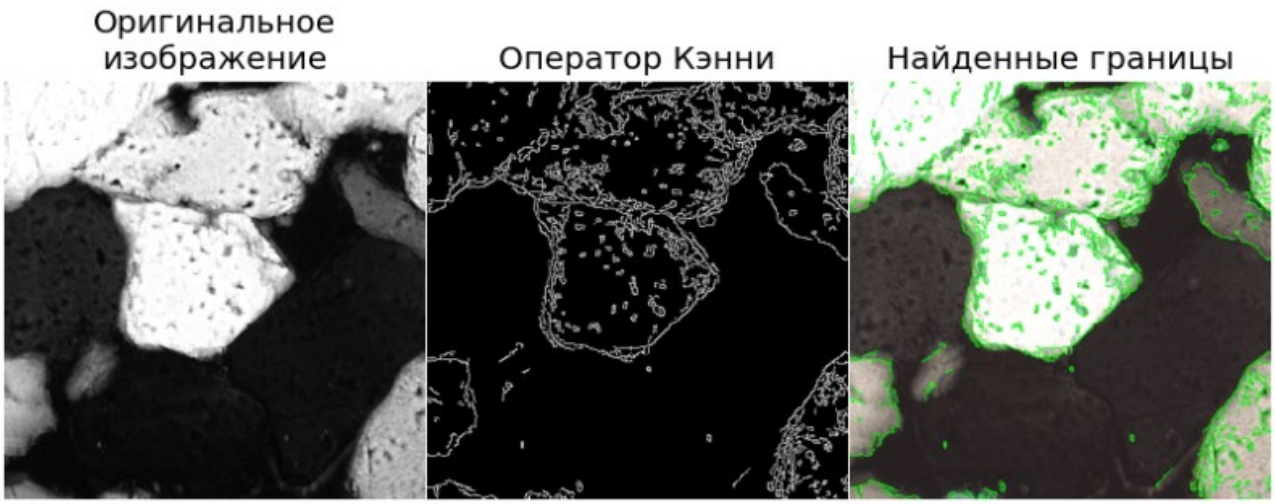


Рисунок 9 – Результат работы оператора Кэнни

Стоит отметить, что оператор Кэнни требует настройки двух пороговых значений: нижнего и верхнего. Эти значения могут быть различными в зависимости от конкретной задачи и изображения. Нижний порог определяет минимальное значение величины градиента, необходимое для того, чтобы

пиксель был рассмотрен как часть границы, а верхний порог определяет минимальное значение величины градиента, необходимое для того, чтобы пиксель был рассмотрен как часть «сильной» границы.

Одним из главных преимуществ оператора Кэнни является то, что он способен находить границы объектов с высокой точностью и четкостью. Однако его недостатком является то, что он может быть чувствителен к шуму на изображении. Поэтому перед применением оператора Кэнни рекомендуется выполнить сглаживание изображения для удаления зашумленных пикселей.

Контрольные вопросы

1. Какие два ядра используются в операторе Собеля для обнаружения границ?
2. Каким образом объединяются результаты применения ядер G_x и G_y в операторе Собеля?
3. На какие этапы разделяется алгоритм Кэнни для обнаружения границ?

Лабораторная работа

1. Напишите код с использованием оператора Собеля для выделения границ на изображении. Загрузите изображение и примените оператор Собеля по горизонтали и по вертикали.
2. Возьмите любое изображение и напишите программу для обнаружения границ с помощью оператора Кэнни. Исследуйте влияние изменения пороговых значений оператора Кэнни на результат сегментации.
3. Примените оператор Собеля и оператор Кэнни к разным типам изображений (например, фотографии пейзажей и изображения с объектами). Сравните результаты и опишите, какой оператор дал более четкие и точные границы для каждого типа изображения.

Глава 4. Методы сегментации на основе кластеризации

K-Means Clustering

K-Means Clustering – это алгоритм кластеризации, который используют для разбиения изображения на сегменты, основываясь на их сходстве в пространстве цветовых и пространственных характеристик. В случае сегментации двумерного RGB-изображения кластеризацию проводят в пятимерном пространстве, где две координаты – это положения пикселя на плоскости, а три другие – его цветовые характеристики (красный, зеленый и синий). Отметим, что также часто изображение перед кластеризацией переводят в оттенки серого или рассматривают только одну цветовую характеристику, в этом случае кластеризацию проводят в трехмерном пространстве признаков.

Основная идея метода K-Means Cluster заключается в следующем.

1. *Задание количества кластеров*. Выбрать количество кластеров, на которые будет разделено изображение.
2. *Определение центров кластеров*. Определить случайным образом K центров кластеров $C_1, C_2, \dots, C_K$, соответствующих кластерам $S_1, S_2, \dots, S_K$. Заметим, что часто вместо термина «центр» используют термин «центроид».
3. *Вычисление расстояний между всеми пикселями и центрами*. Вычислить для каждого *i*-го пикселя расстояние $d_j^{(i)}$ его вектора характеристик $u^{(i)}$ до каждого центра по формуле
$$d_j^{(i)} = \|u^{(i)} - C_j\|_2 .$$
4. *Определение кластера для каждого пикселя*. «Прикрепить» каждый пиксель к ближайшему кластеру j, основываясь на полученных расстояниях до центров:
$$j = \underset{j}{\operatorname{argmin}}\, d_j^{(i)} .$$

5. *Вычисление новых координат центров*. Обновить координаты центров кластеров, вычислив как средние значения всех характеристик пикселей, принадлежащих данному кластеру:

$$C_j = \frac{1}{N_j} \sum_{u^{(i)} \in S_j}^{N_j} u^{(i)} ,$$

где N_j – количество пикселей (мощность) j-го кластера.

6. *Итерации*. Повторять шаги 3-5 до тех пор, пока не будет выполнен один из двух критериев остановки алгоритма:
   - границы кластеров и расположения центров кластеров перестанут изменяться от итерации к итерации, т.е., начиная с какой-то итерации в каждом кластере будет оставаться один и тот же набор пикселей;
   - достигнут критерий сходимости. Чаще всего используют критерий суммы расстояний между центрами кластеров и принадлежащим им пикселям

$$\sum_{j=1}^{K} \sum_{u^{(i)} \in S_j}^{N_j} \|u^{(i)} - C_j\|_2 .$$

После выполнения алгоритма каждый пиксель будет отнесен к одному из кластеров, при этом, каждый кластер будет представлять собой группу пикселей с близкими значениями цветовых характеристик. Эти кластеры могут быть использованы для разделения изображения на сегменты. В конечном результате будут выведены (получены) исходное изображение и сегментированное изображение с помощью метода K-Means Clustering (рисунок 10). Ниже приведен код программы на языке Python

```
import numpy as np
import cv2
def kmeans_segmentation(image, k):
    # Преобразование изображения в массив пикселей
    pixels = image.reshape(-1, 3).astype(np.float32)
```

```
        # Выполнение K-Means сегментации
        criteria = (cv2.TERM_CRITERIA_EPS +
cv2.TERM_CRITERIA_MAX_ITER, 100, 0.2)
        _, labels, centers = cv2.kmeans(pixels, k, None,
criteria, 10, cv2.KMEANS_RANDOM_CENTERS)

        # Преобразование меток обратно в размеры изображения
        segmented_image =
centers[labels.flatten()].reshape(image.shape)
        return segmented_image

    # Загрузка изображения
    image = cv2.imread('img.png')

    # Применение K-Means сегментации с 2 кластерами
    k = 2
    segmented_image = kmeans_segmentation(image,
k).astype('uint8')
```

В данной программе используем функцию cv2.kmeans для выполнения сегментации методом K-Means. Эта функция принимает на вход несколько параметров:

- pixels: массив пикселей изображения;
- k: количество сегментов или кластеров;
- None: параметр для передачи начальных центров кластеров. В данном случае центры выбираем случайным образом;
- criteria (type, max_iter, epsilon): критерии остановки для алгоритма K-Means. Здесь используем комбинацию критериев cv2.TERM_CRITERIA_EPS и cv2.TERM_CRITERIA_MAX_ITER. Первый критерий cv2.TERM_CRITERIA_EPS означает, что алгоритм завершается, если отклонение между двумя последовательными итерациями становится меньше определенного значения eps. Второй критерий

cv2.TERM_CRITERIA_MAX_ITER ограничивает максимальное количество итераций алгоритма;

– 10: количество запусков алгоритма K-Means с разными начальными центрами;
– cv2.KMEANS_RANDOM_CENTERS: флаг для выбора начальных центров кластеров случайным образом.

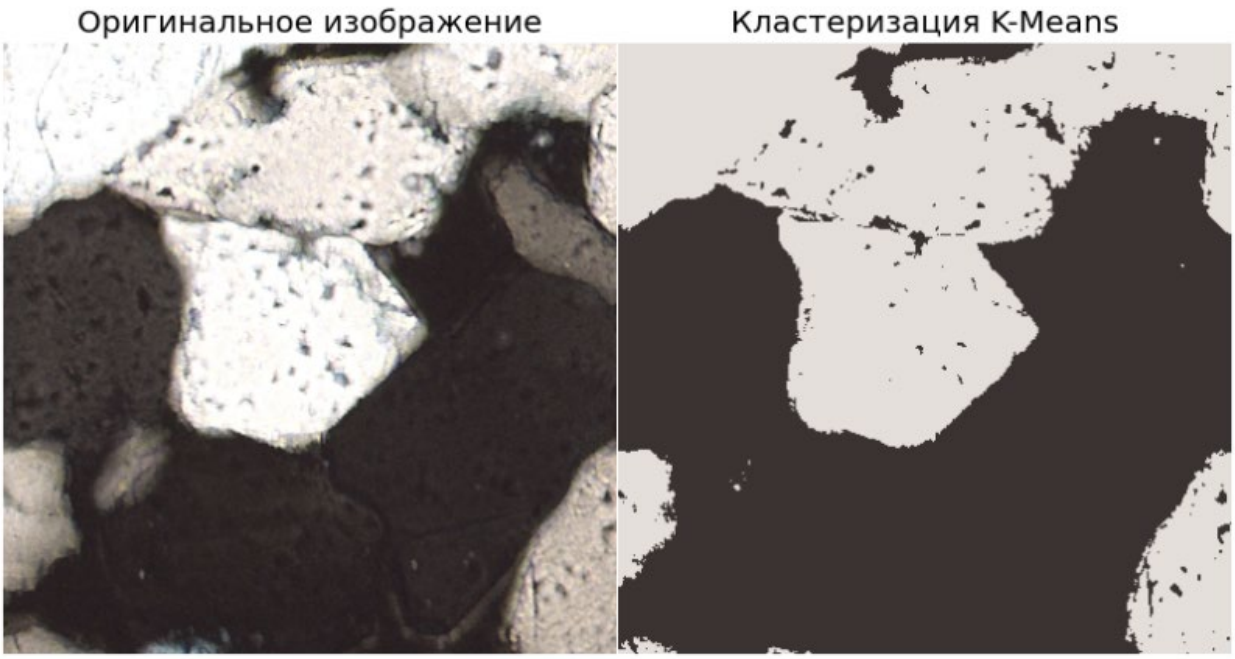


Рисунок 10 – Результат сегментации K-Means с двумя кластерами

Функция cv2.kmeans возвращает:

– labels: массив меток, где каждый пиксель помечен соответствующим кластером;
– centers: массив центроидов каждого кластера.

Цветовые пространства CIELab и CIELuv

Люди по-разному воспринимают различные цвета. Поэтому важно иметь объективный способ характеризовать цвета, а также количественно определять различия (расстояния) между цветами. Цветовая палитра RGB для этого не подходит, однако существует стандартная система цветов CIELab, которую используют во всем мире.

Трехмерное цветовое пространство CIELab (рисунок 11) натянуто на три взаимно перпендикулярных вектора (оси). На оси L^* откладывают значения яркости таким образом, что белый объект имеет значение L^* равное 100, а черный объект – равное 0. Ахроматические цвета, т.е. оттенки серого, задают на оси L^*: Хроматические («реальные») цвета описывают с помощью двух осей в горизонтальной плоскости. Ось a^* представляет собой зелено-красную ось, а ось b^* идет от синего $(-b^*)$ к желтому $(+b^*)$. Значения a^* и b^* изменяют от -127 до 128 (в некоторых приложения от 0 до 255). Чем дальше лежит точка от оси L^*, тем цвет более насыщенный.

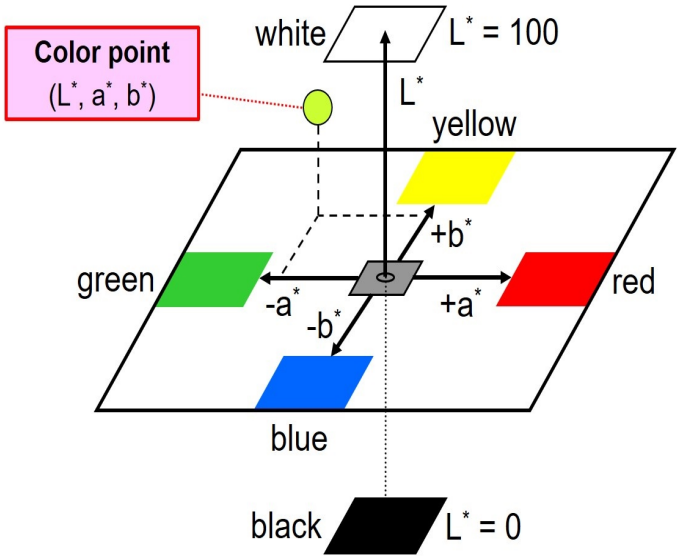


Рисунок 11 – Схема трехмерного цветового пространства CIELab

Таким образом, каждый цвет представляют точкой (L^*, a^*, b^*). Символ звездочки (*) для L^*, a^* и b^* указывает, что это новая цветовая система, которую считают продолжением старой системы CIELab. Новую систему теперь повсеместно применяют для количественной оценки цветов, хотя часто используют упрощенное обозначение, а именно без символа звездочки. Заметим, что также используют и систему CIELuv.

Расстояние d между двумя цветовыми точками (L_1^*, a_1^*, b_1^*) и (L_2^*, a_2^*, b_2^*) вычисляется как евклидова норма между двумя векторами в пространстве CIELab:

$$d = \sqrt{((L_2^*)^2 - (L_1^*)^2)^2 + ((a_2^*)^2 - (a_1^*)^2)^2 + ((b_2^*)^2 - (b_1^*)^2)^2}.$$

Теперь, когда есть цветовая система, позволяющая количественно оценивать расстояния между различными цветами, необходимо определить формулы трансформации между RGB-палитрой и цветовой системой CIELab. Сначала нормируем RGB-компоненты:

$$(R^*, G^*, B^*) = (R/255, G/255, B/255)\,.$$

Затем преобразование от RGB к CIELab проведем через промежуточную цветовую систему CIEXYZ:

$$R_l = \begin{cases} \dfrac{R^*}{12.92}\,, & R^* \le 0.04045\,, \\ \dfrac{R^* + 0.055}{1.055}\,, & R^* > 0.04045\,. \end{cases}$$

По аналогичным формулам, как и для красной составляющей, определим другие составляющие: G_l и B_l через значения G и B. Затем вычислим координаты (X_D, Y_D, Z_D) палитры CIEXYZ:

$$\begin{pmatrix} \mathrm{X_D} \\ \mathrm{Y_D} \\ \mathrm{Z_D} \end{pmatrix} = \begin{pmatrix} 0.4124 & 0.3576 & 0.1805 \\ 0.2126 & 0.7152 & 0.0722 \\ 0.0193 & 0.1192 & 0.9505 \end{pmatrix} \begin{pmatrix} R_l \\ G_l \\ B_l \end{pmatrix}.$$

И, наконец, по координатам (X_D, Y_D, Z_D) получим координаты (L*, a*, b*):

$$L^* = 116\, f\left(\frac{\mathrm{Y_D}}{\mathrm{Y_n}}\right) - 16\,,$$

$$a^* = 500\left(f\left(\frac{\mathrm{X_D}}{\mathrm{X_n}}\right) - f\left(\frac{\mathrm{Y_D}}{\mathrm{Y_n}}\right)\right),$$

$$b^* = 200\left(f\left(\frac{\mathrm{Y_D}}{\mathrm{Y_n}}\right) - f\left(\frac{\mathrm{Z_D}}{\mathrm{Z_n}}\right)\right),$$

где

$$f(t) = \begin{cases} \sqrt[3]{t}\,, & \text{если } t > \delta^3\,, \\ \dfrac{t}{3\delta^2 + \dfrac{4}{29}}\,, & \text{иначе}\,, \end{cases}$$

$$\delta = \frac{6}{29}\,.$$

Здесь X_n, Y_n и Z_n представляют собой трехцветные значения CIEXYZ эталонной белой точки. Эти значения выберем следующими (для белого цвета согласно эталону D50): $X_n = 95.0489$, $X_n = 100$ и $X_n = 82.5188$. Однако, в случае

других эталонов, можно задать значения, немного отличающиеся от приведенных.

После того, как для всех пикселей получены значения (L^*, a^*, b^*), проведем кластеризацию в данной цветовой системе. Затем вернемся в стандартную систему RGB, чтобы отобразить полученные кластеры.

Сначала вычислим (X_D, Y_D, Z_D):

$$X_D = X_n f^{-1}\left(\frac{L^* + 16}{116} + \frac{a^*}{500}\right),$$

$$Y_D = Y_n f^{-1}\left(\frac{L^* + 16}{116}\right),$$

$$Z_D = Z_n f^{-1}\left(\frac{L^* + 16}{116} - \frac{b^*}{500}\right),$$

где

$$f^{-1}(t) = \begin{cases} t^3, & \text{если } t > \delta^3, \\ 3\delta^2\left(t - \frac{4}{29}\right), & \text{иначе}. \end{cases}$$

Затем получим

$$\begin{pmatrix} R_l \\ G_l \\ B_l \end{pmatrix} = \begin{pmatrix} 3.2406 & -1.5372 & -0.4986 \\ -0.9689 & 1.8758 & 0.0415 \\ 0.0557 & -0.2040 & 1.0570 \end{pmatrix} \begin{pmatrix} X_D \\ Y_D \\ Z_D \end{pmatrix}.$$

На последнем этапе используем гамма-коррекцию для красной

$$R^* = \begin{cases} 12.92\, R_l, & R_l \le 0.0031308, \\ 1.055\, R - 0.055, & R_l > 0.0031308. \end{cases}$$

и других нормированных составляющих RGB. В результате получим

$$(R, G, B) = (255\, R^*, 255\, G*, 255\, B*).$$

Помимо цветового пространства CIELab используют также очень похожее пространство CIELuv (называемое также CIE 1976 UCS). Не останавливаясь подробно на этом пространстве, приведем его цветовое поле на рисунке 12. На данном рисунке по контуру приведены значения длин волн от фиолетового (380-440 нм) до красного (625-770 нм) цветов. В центре изображения показаны значения цветовых температур (2000 К, 3000 К, …, 10000 К).

Сравним на рисунке 13 результаты сегментации, полученные методом K-Means для цветовых наборов RGB и CIELab в случае снимка зерён пород. По рисунку можем заметить, что сегментация для обоих наборов практически одинаковая. Это, прежде всего, обусловлено тем, что цвета изображения близки к серому.

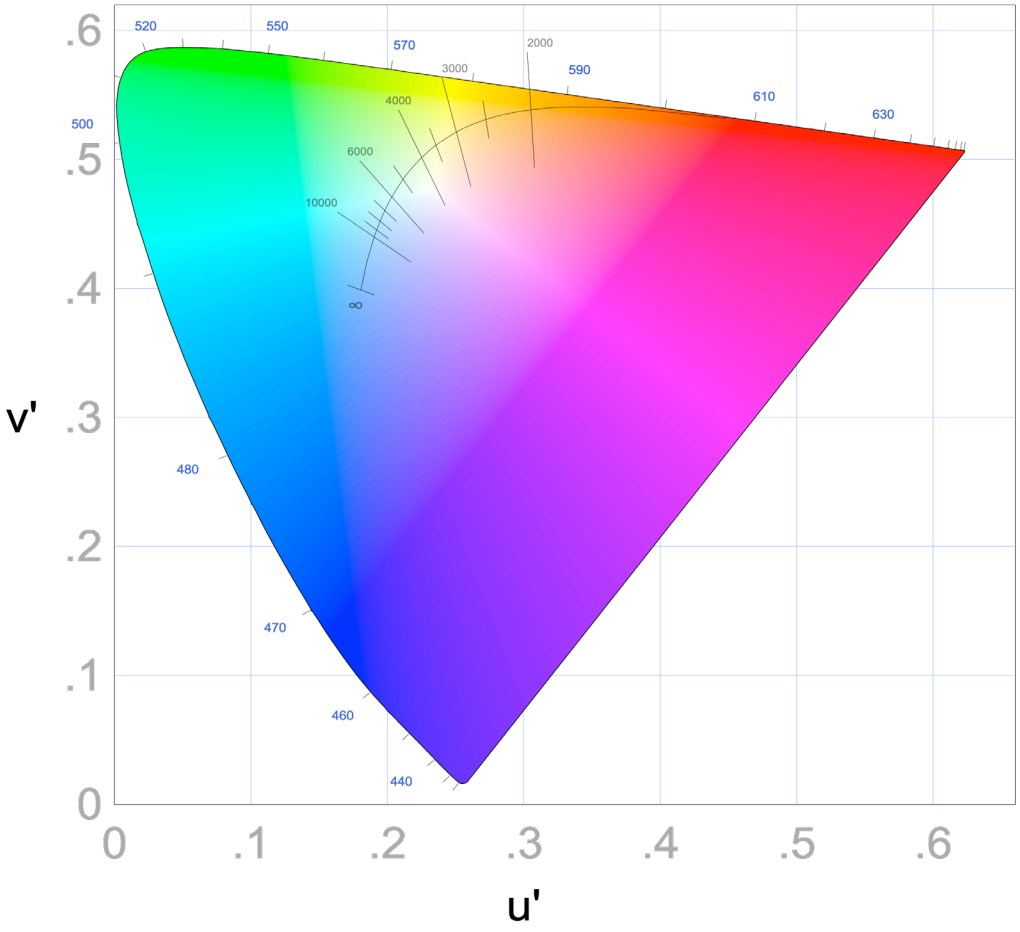


Рисунок 12 – Цветовое пространство CIELuv

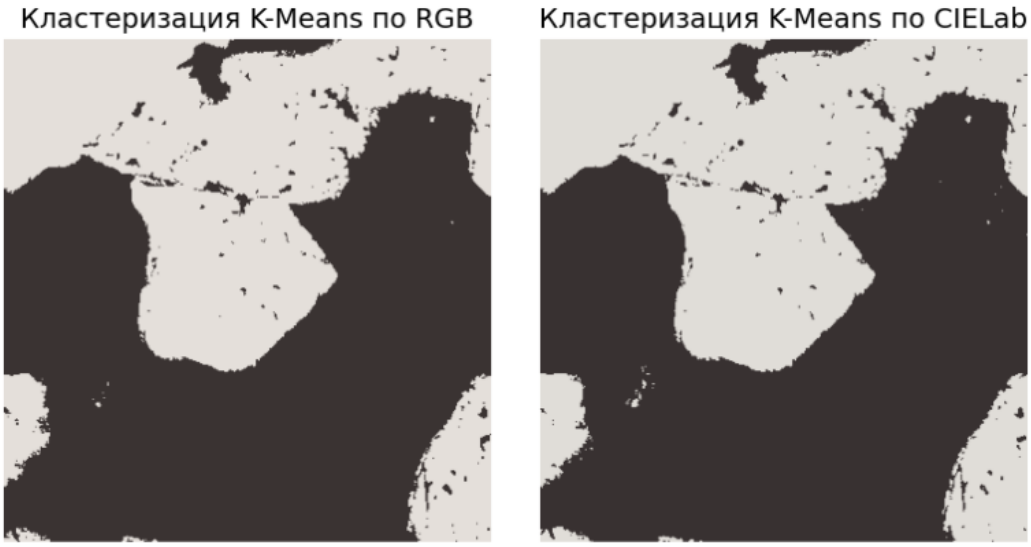


Рисунок 13 – Трехмерное цветовое пространство CIELab

Проведём теперь кластеризацию для насыщенного изображения (красный автомобиль на серо-зелено-голубом фоне) по различным цветовым пространствам. Полученные результаты приведены на рисунке 14. В этом случае видим существенные отличия для цветового пространства RGB с одной стороны и CIELab и CIELuv с другой. Если для цветовых пространств CIE проведём кластеризацию только по L (L^*), то результат получим схожим с кластеризацией по всем компонентам.

Рисунок 14 – Сравнение результатов сегментации K-Means в цветовых пространствах RGB, CIELab и CIELuv

Приведём код, который преобразует изображения в нужную палитру, кластеризует, затем преобразует кластеры в RGB и выводит полученные «кластеризованные» изображения на экран.

```
    import cv2

    def kmeans_segmentation(image, k):
        # Преобразование изображения в массив пикселей
        pixels = image.reshape(-1, 1 if len(image.shape)==2 else
3).astype(np.float32)

        # Выполнение K-Means сегментации
        criteria = (cv2.TERM_CRITERIA_EPS +
cv2.TERM_CRITERIA_MAX_ITER, 100, 0.2)
        _, labels, centers = cv2.kmeans(pixels, k, None,
criteria, 10, cv2.KMEANS_RANDOM_CENTERS)

        # Преобразование меток обратно в размеры изображения
        segmented_image =
centers[labels.flatten()].reshape(image.shape)

        return segmented_image.astype('uint8')

    # Загрузка изображения
    image = cv2.imread('car.jpg') # Изначально в формате BGR

    # Перевод изображения в другие цветовые пространства
    image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    image_lab = cv2.cvtColor(image, cv2.COLOR_BGR2LAB)
    image_luv = cv2.cvtColor(image, cv2.COLOR_BGR2LUV)

    # Применение K-Means сегментации с 3 кластерами
    k = 3
    segmented_image_rgb = kmeans_segmentation(image_rgb, k)
    segmented_image_lab = kmeans_segmentation(image_lab, k)
    segmented_image_lab_only_l =
kmeans_segmentation(image_lab[:,:,0], k)
    segmented_image_luv = kmeans_segmentation(image_luv, k)
```

```
    segmented_image_luv_only_l =
kmeans_segmentation(image_luv[:,:,0], k)

    # Перевод результатов сегментации в RGB
    segmented_image_lab = cv2.cvtColor(segmented_image_lab,
cv2.COLOR_LAB2RGB)
    segmented_image_luv = cv2.cvtColor(segmented_image_luv,
cv2.COLOR_LUV2RGB)

    # Визуализация результатов
    images = [image_rgb, segmented_image_rgb,
segmented_image_lab, segmented_image_lab_only_l,
segmented_image_luv, segmented_image_luv_only_l]
    labels = ['Оригинальное изображение', 'Кластеризация K-Means
по RGB', 'Кластеризация K-Means по LAB', 'Кластеризация K-
Means\nпо LAB (канал L)', 'Кластеризация K-Means по LUV',
'Кластеризация K-Means\nпо LUV (канал L)']

    # Создание сетки 3x2
    fig, axes = plt.subplots(nrows=3, ncols=2, figsize=(8, 12))

    # Заполнение каждой ячейки сетки изображением и подписью
    for i, ax in enumerate(axes.flat):
        ax.imshow(images[i], cmap='gray')
        ax.set_title(labels[i])
        ax.axis('off')

    plt.show()
```

Mean Shift

Mean Shift – это алгоритм кластеризации, который относят к методам машинного обучения без учителя и используют для разбиения изображения на сегменты в пространстве цветовых характеристик. Идея метода состоит в

группировке пикселей по цвету в заданное количество кластеров (групп) данных в многомерном пространстве.

Принцип работы метода Mean Shift следующий.

1. *Инициализация центров*. Выберем начальные центры, которые будем использовать как стартовые точки для процесса сдвига среднего значения.
2. *Вычисление плотности*. Для каждой точки данных вычислим «плотность», которая определяется на основе расстояния между этой точкой и остальными точками. Это можем выполнить через функцию-ядро (например, гауссово ядро).
3. *Вычисление сдвига среднего значения*. Для каждой точки данных найдем сдвиг среднего значения в направлении более плотной области данных. Этот сдвиг вычислим путем взвешенной суммы векторов направлений от точки данных до других точек, пропорционально их плотности.
4. *Обновление значений центров*. Новые центры вычислим путем сдвига старых центров на величину, равную сдвигу среднего значения.
5. *Итерационный процесс*. Шаги 3 и 4 выполним итерационно до тех пор, пока центры не перестанут смещаться или пока не будет достигнуто максимальное количество итераций.
6. *Распределение точек по кластерам*. Каждую точку данных присваиваем к ближайшему центру, формируя таким образом кластеры.
7. *Последующая обработка*. Выполним постпроцессинг, чтобы объединить слишком близкие кластеры или удалить выбросы.

В контексте сегментации изображений пиксели рассматривают как данные (точки в многомерном пространстве), определяемые их цветом, текстурой и другими характеристиками. Применение метода Mean Shift к этим

пикселям позволяет выявить группы пикселей со схожими характеристиками и выделить сегменты объектов и фоновых областей на изображении.

Преимущества метода Mean Shift.

1. Автоматическое определение количества кластеров. Mean Shift не требует указания числа кластеров заранее. Метод сам определяет количество кластеров.
2. Обработка различных размеров кластеров. Алгоритм способен обнаруживать кластеры различных размеров и форм.
3. Устойчивость к выбросам. Благодаря процессу сдвига среднего значения алгоритм менее подвержен влиянию выбросов.

Однако метод Mean Shift также имеет некоторые ограничения.

1. Вычислительная сложность. Алгоритм может быть вычислительно затратным, особенно для больших наборов данных.
2. Зависимость от параметров. Для определения параметров (например, размер окна для вычисления сдвига среднего значения) может быть необходимо провести некоторые эксперименты.
3. Плотность данных. Если данные имеют неравномерное распределение плотности, то алгоритм может давать менее точные результаты.

Метод Mean Shift считают эффективным инструментом для сегментации изображений, но его успешное применение требует тщательного подбора параметров и обработки результатов. Для реализации данного метода на языке Python используем функцию MeanShift() из библиотеки scikit-learn. На рисунке 15 приведен результат кластеризации.

```
import numpy as np
from sklearn.cluster import MeanShift, estimate_bandwidth
image = cv2.imread('image.png')

# Преобразование изображения в массив пикселей
pixels = np.reshape(image, (-1, 3))
```

```
    # Оценка параметра bandwidth
    bandwidth = estimate_bandwidth(pixels, quantile=0.2,
n_samples=500)
    # Создание объекта MeanShift с найденным параметром bandwidth
    ms = MeanShift(bandwidth=bandwidth, bin_seeding=True)

    # Обучение модели на данных
    ms.fit(pixels)

    # Получение меток кластеров
    labels = ms.labels_

    # Преобразование меток кластеров в 2D-матрицу
    labels_2d = np.reshape(labels, image.shape[:2])
```

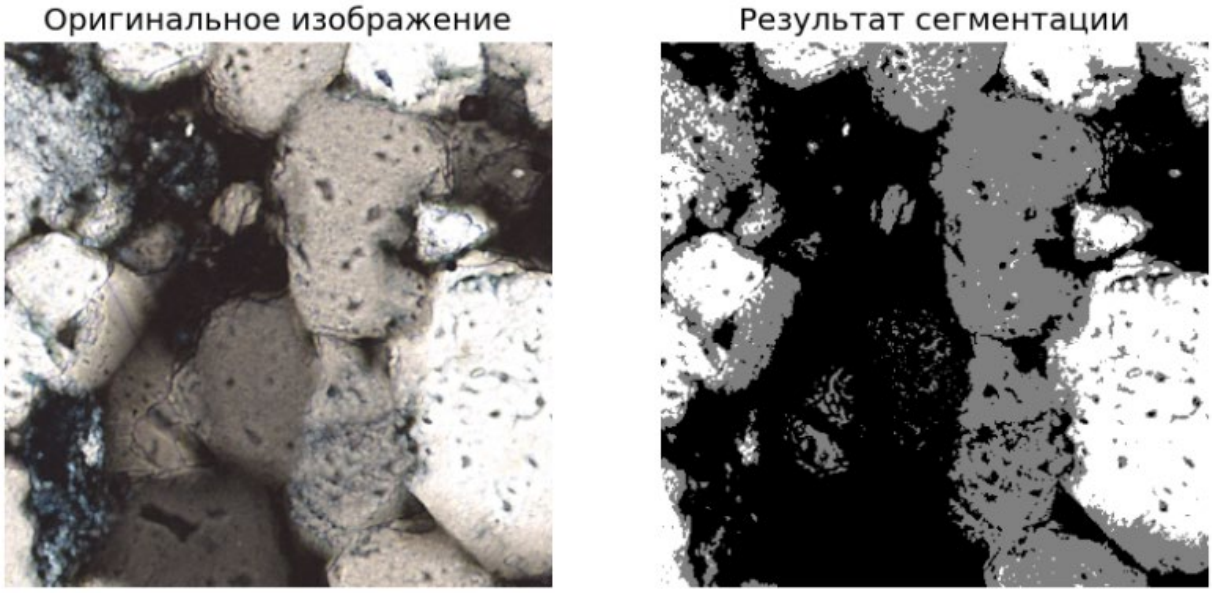


Рисунок 15 – Результат кластеризации пикселей с помощью алгоритма Mean Shift (было найдено три кластера)

Параметр bandwidth и bin_seeding – это два важных параметра метода кластеризации MeanShift(), которые влияют на его производительность и результаты.

Параметр bandwidth определяет размер окрестности, используемой для оценки плотности точек в пространстве при кластеризации, и влияет на масштаб и гладкость сегментации. Для получения оптимального значения

bandwidth можно использовать функцию `estimate_bandwidth` из модуля sklearn.cluster. В параметре quantile можно указать, какую долю точек использовать при оценке параметра bandwidth. Меняя это значение, можно получить различные уровни детализации сегментации.

Параметр bin_seeding влияет на скорость сегментации. Если bin_seeding установлен в True, то начальные точки берутся из бинарной сетки, что может помочь ускорить алгоритм, но может привести к потере точности в определении центроидов.

Выбор и настройка этих параметров должны осуществляться исходя из конкретной задачи и особенностей данных. Рекомендуется проводить эксперименты с разными значениями и наблюдать, как они влияют на качество результатов и производительность алгоритма.

DBSCAN

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) – это алгоритм кластеризации, который применяют для выявления кластеров произвольной формы в пространстве данных на основе плотности распределения точек (данных). Метод определяет кластеры как плотные области точек (точки, которые находятся близко друг к другу), и обнаруживает выбросы как менее плотные области (точки, которые не принадлежат ни одному из кластеров). Процесс сегментации изображений методом DBSCAN состоит из следующих шагов.

1. *Подготовка данных*. Изображение преобразуем в набор точек, таких как пиксели или векторы признаков, для использования алгоритмом DBSCAN.
2. *Определение параметров DBSCAN*. Алгоритм DBSCAN имеет два основных параметра:
   - Эпсилон (eps) – максимальное расстояние между двумя точками. Если расстояние меньше эпсилон, то точки считаются соседними.

– Минимальное количество точек (minsamples) – минимальное количество точек в окрестности размером эпсилон. Если число точек больше или равно этого значения, то точка, вокруг которой построена окрестность, считается базовой (core point). Такие точки лежат в области высокой плотности данных.

3. *Выполнение DBSCAN*. На этом шаге применим алгоритм DBSCAN к данным. Начнем с выбора случайной неисследованной точки данных и найдем всех ее соседей в пределах радиуса eps. Если количество соседей больше или равно minsamples, то принимаем их как ядро и образуем кластер. Затем процесс повторяем с каждой точкой-ядром и ее соседями до тех пор, пока не рассмотрим все точки.
4. *Постобработка кластеров*. После кластеризации считаем, что каждая точка либо принадлежит одному из кластеров, либо является выбросом (outlier). Выбросы – это точки данных, которые не имеют достаточного количества соседей, чтобы стать ядром. В рамках сегментации изображений выбросы представляют фоновые или шумовые пиксели.

Преимущества DBSCAN в сегментации изображений включают способность кластеризовать данные произвольной формы и устойчивость к выбросам. Однако алгоритм является чувствительным к выбору параметров, таким как eps и minsamples, и может потребовать определенной настройки для каждого конкретного изображения или типа данных.

Ниже приведем код и результат кластеризации на рисунке 16.

```
import numpy as np
from sklearn.cluster import DBSCAN
import cv2

# Загрузка изображения
image = cv2.imread('image.png')
```

```
    # Преобразование RGB изображения в одномерный массив
    pixels = image.reshape(-1, 3)

    # Создание объекта DBSCAN
    dbscan = DBSCAN(eps=3, min_samples=10)

    # Производим кластеризацию пикселей
    labels = dbscan.fit_predict(pixels)

    # Преобразуем метки в изображение
    segmented_image = labels.reshape(image.shape[0],
image.shape[1])
```

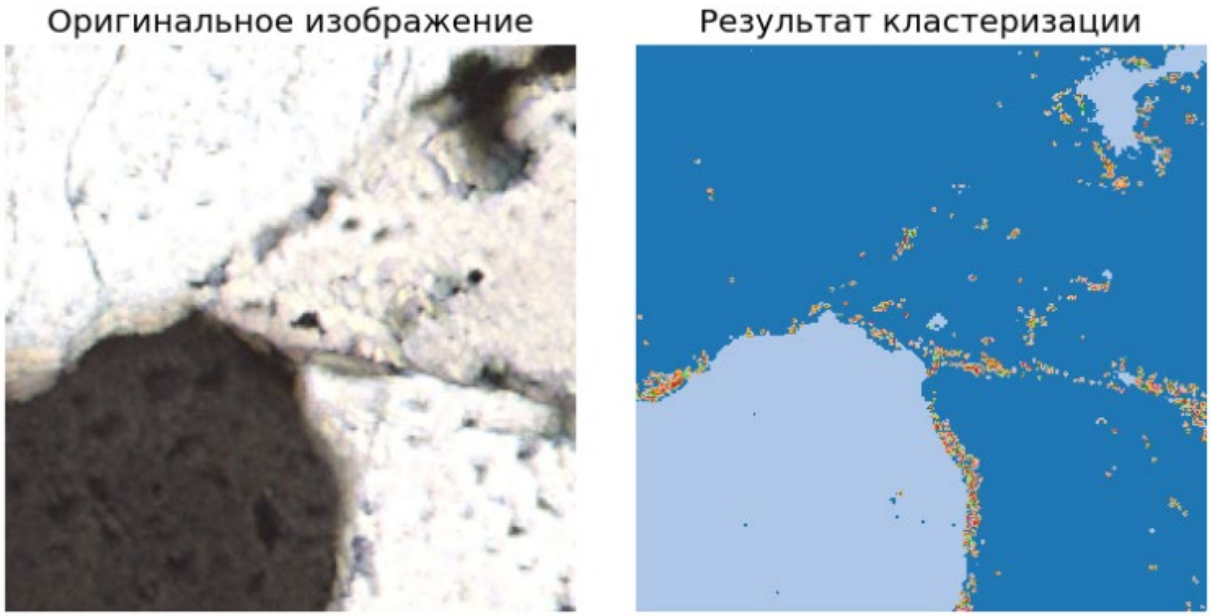


Рисунок 16 – Результат кластеризации методом DBSCAN

Контрольные вопросы

1. Что такое K-Means Clustering?
2. Какие основные шаги включает алгоритм K-Means Clustering?
3. Какие параметры передаются в функцию cv2.kmeans() в OpenCV?
4. Какие преимущества и ограничения имеет метод MeanShift?
5. Какие основные параметры влияют на работу метода DBSCAN?

Лабораторная работа

1. Расширьте функцию kmeans_segmentation из представленного кода, чтобы она принимала дополнительный параметр max_iterations и передавала его в функцию kmeans.
2. Примените kmeans к нескольким изображениям. Изменяя такие параметры, как количество кластеров, максимальное количество итераций попробуйте получить оптимальный результат.
3. Постройте цветовую плоскость CIELab (a*, b*) и объясните по построенному изображению связь между CIELab и RGB.
4. Попробуйте разные комбинации значений eps и min_samples на одном и том же изображении для алгоритма DBSCAN. Приведите визуальные результаты кластеризации для каждой комбинации и оцените, какие параметры лучше подходят для сегментации.
5. Выберите несколько изображений и выполните сравнительный анализ, оценив качество сегментации для трех описанных методов: K-Means, Mean Shift, DBSCAN.
6. Проведите сегментацию нескольких «сильно» цветных изображений в четырех цветовых пространствах (RGB, оттенки серого, CIELub и CIELuv) тремя представленными методами. Сделайте и обоснуйте выводы о сходстве и различии различных пространств и методов сегментации.

Глава 5. Методы сегментации изображений на графах

Min-Cut Segmentation

Min-Cut Segmentation – это эффективный метод компьютерного зрения, в котором решение задачи разбиения изображений на однородные сегменты или области сводят к решению задачи поиска минимального разреза графа изображения. Этот метод часто применяют для выделения объектов и их контуров на изображении.

Суть метода состоит в том, что изображение представляют в виде графа, где пиксели изображения считают вершинами графа, а ребра между ними строят по степени сходства между пикселями (яркостями пикселей). Затем граф разбивают на две части путем разреза ребер. Разрез ребер – это разделение ребер графа на две группы, где одна группа соединяет вершины на одной стороне разреза, а другая – на другой стороне.

Метод Min-Cut Segmentation делает оптимальное разбиение графа на две части с помощью алгоритма поиска минимального разреза. Данный алгоритм определяет, какие ребра графа должны быть разрезаны, чтобы минимизировать суммарное весовое значение ребер. Таким образом получают изображение, разделенное на два сегмента – передний и задний план.

Чтобы получить более точное разбиение изображение и большее число однородных сегментов, метод Min-Cut Segmentation применяют несколько раз.

Normalize-Cut

Normalized Cut – это метод сегментации изображений на основе теории графов, который расширяет идеи метода Min-Cut Segmentation. Вместо того, чтобы разбивать граф на две части, Normalized Cut разбивает его на несколько сегментов.

Основная идея Normalized Cut – разделение графа изображения на несколько сегментов, при котором каждый сегмент содержит схожие пиксели, а

границы между сегментами проходят через области, где различия в пикселях наибольшие.

Как и в методе Min-Cut Segmentation, изображение представляют в виде графа, где каждый пиксель принимают за вершину графа, а ребра между ними строят по степени сходства пикселей. Однако в Normalized Cut ребра графа имеют веса, которые определяют не только степень сходства между пикселями, а также степень связи между вершинами внутри каждого сегмента.

В основе метода Normalized Cut лежит задача минимизации суммарного веса всех границ между сегментами при условии, что каждый сегмент образует достаточно большую область и имеет однородное распределение пикселей.

Ключевой момент в Normalized Cut – использование нормализованного разреза графа, который учитывает как степень различия между пикселями, так и размер каждого сегмента. Это позволяет получить более точные результаты при разбиении графа на несколько сегментов.

В целом, метод Normalized Cut считают более точным и мощным способом сегментации изображений, чем метод Min-Cut Segmentation, поскольку он позволяет разбивать изображение на несколько сегментов с более сложной структурой. Однако метод Normalized Cut также требует мощных вычислительных ресурсов для обработки больших изображений и настройки параметров для достижения наилучших результатов.

Приведем реализацию метода сегментации Normalized Cut с помощью библиотеки skimage. Для получения начальной маски сегментации изображения используем алгоритм кластеризации SLIC (Simple Linear Iterative Clustering). Результат сегментации показан на рисунке 17.

```
# Импорт необходимых библиотек
from skimage import data, segmentation, color
from skimage import graph
from matplotlib import pyplot as plt

# Загрузка тестового изображения (кофейной чашки)
```

```
img = cv2.imread('image.png')

# Сегментация изображения с использованием метода SLIC. Этот метод разбивает изображение на сегменты, учитывая цвета и текстуры.
labels1 = segmentation.slic(img, compactness=15, n_segments=50)

# Перевод изображения в градациях серого в цветовой канал RGB
out1 = color.label2rgb(labels1, img, kind='avg', bg_label=0)

# Построение Region Adjacency Graph (RAG). Строится граф смежности регионов, где веса ребер определяются сходством цветов в смежных регионах.
g = graph.rag_mean_color(img, labels1, mode='similarity')

# Выполнение нормализованного разреза графа
labels2 = graph.cut_normalized(labels1, g)
out2 = color.label2rgb(labels2, img, kind='avg', bg_label=0)

# Визуализация результатов
fig, ax = plt.subplots(ncols=3, sharex=True, sharey=True, figsize=(10, 8))
ax[0].imshow(img)
ax[0].set_title('Оригинальное изображение')
ax[1].imshow(out1)
ax[1].set_title('Сегментация SLIC')
ax[2].imshow(out2)
ax[2].set_title('Сегментация Normalized Cut')
for a in ax:
    a.axis('off')
plt.tight_layout()
plt.show()
```

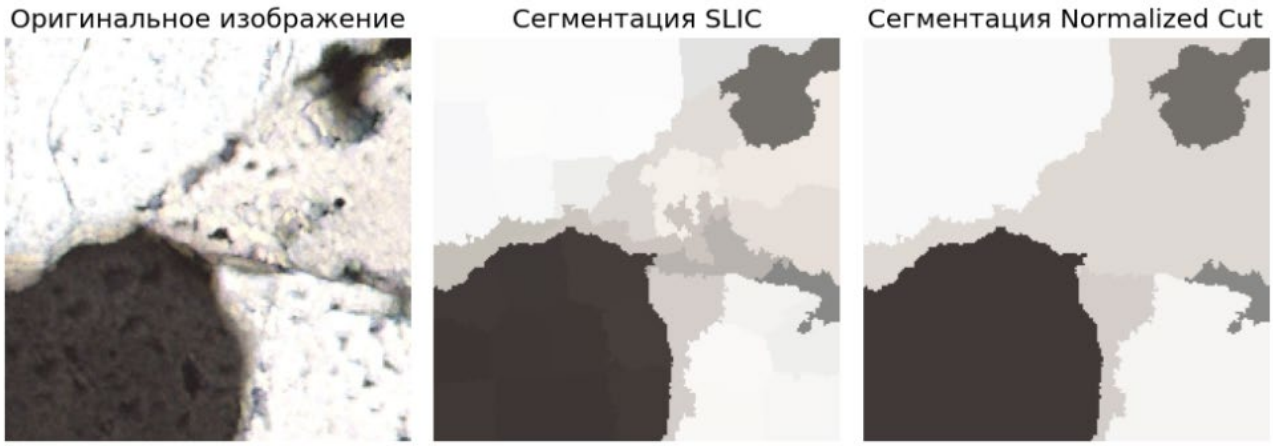


Рисунок 17 – Результат сегментации методом Normalized Cut

GrabCut

GrabCut – это алгоритм сегментации изображений, подробно изложенный в работе «GrabCut: Interactive foreground extraction using iterated graph cuts» (2004 год) авторами Carsten Rother, Vladimir Kolmogorov и Andrew Blake. Этот метод применяют для решения задачи интерактивного выделения переднего плана изображения с минимальным участием пользователя.

Рассмотрим основные этапы метода GrabCut .

1. *Инициализация*. На первом этапе задаём примерную область местоположения объекта на изображении и присваиваем каждому пикселю изображения начальные метки: «фон» и «передний план».
2. *Построение графа*. Изображение представим в виде графа, где вершины соответствуют пикселям, а ребра – связям между пикселями. Определим веса ребер на основе цвета пикселей и их пространственного расположения.
3. *Вычисление энергии*. Задаём энергетическую функцию, которая включает в себя данные о цвете пикселей и их связях, а также вклад от пользовательских меток. Эта функция оценивает, насколько хорошо каждый пиксель соответствует переднему или заднему плану. Минимизация этой функции приведет нас к оптимальной сегментации.

4. *Итерационный процесс*. Решаем задачу оптимизации с помощью алгоритма графового разреза, который итеративно обновляет принадлежность пикселей к переднему или заднему плану, учитывая данные от пользователя и информацию об изображении.
5. *Обновление маски*. После завершения алгоритма обновим маску сегментации на основе результата минимизации энергии. Пиксели, отмеченные как передний план, считаем частью объекта, а пиксели, отмеченные как фон, считаем частью фона.

Ниже приведем программную реализацию описанного метода для выделения объектов на изображении.

```
import cv2
import numpy as np

# Загрузка изображения
image = cv2.imread('image.png')

# Инициализация маски и прямоугольника
mask = np.zeros(image.shape[:2], np.uint8)
rect = (80, 80, 230, 230)  # Прямоугольник, содержащий объект (x, y, width, height)

# Визуализация прямоугольника на изображении
image_with_rect = image.copy()
cv2.rectangle(image_with_rect, (rect[0], rect[1]), (rect[0] + rect[2], rect[1] + rect[3]), (0, 0, 255), 2)

# Инициализация GrabCut
bgd_model = np.zeros((1, 65), np.float64)
fgd_model = np.zeros((1, 65), np.float64)

# Применение GrabCut
cv2.grabCut(image, mask, rect, bgd_model, fgd_model, iterCount=5, mode=cv2.GC_INIT_WITH_RECT)
```

```
    # Получение двоичной маски: 3 - передний план, 2 - задний
план
    mask_new = np.where((mask == 2) | (mask == 0), 0,
1).astype('uint8')

    # Применение маски к оригинальному изображению
    result = image * mask_new[:, :, np.newaxis]
```

Рассмотрим каждый параметр метода cv2.grabCut:

- img: входное изображение, которое GrabCut предполагает как 8-битное 3-канальное изображение.
- mask: маска ввода/вывода.
- rect: заданный пользователем прямоугольник, в котором, вероятно, находится объект.
- bgd_model и fgd_model: это временные массивы, используемые алгоритмом GrabCut для хранения статистики о фоне (background) и переднем плане (foreground) соответственно. Каждый из этих массивов представляет собой одномерный массив с размерностью (1, 65), где 65 – количество элементов, используемых для хранения статистики. В GrabCut, для каждого пикселя в изображении используется 65 параметров для описания его характеристик, таких как цвет, текстура и т.д. Параметры этих моделей обновляются в процессе работы алгоритма.
- iterCount: количество итераций. Чем больше итераций, тем дольше будет работать GrabCut и в идеале результаты будут лучше.
- mode: cv2.GC_INIT_WITH_RECT или cv2.GC_INIT_WITH_MASK.

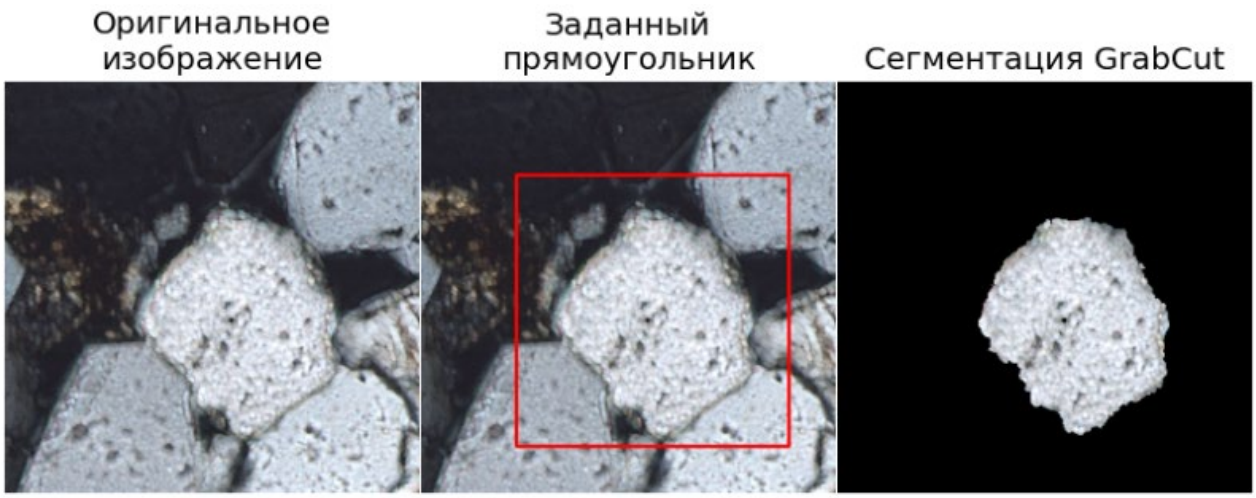


Рисунок 18 - Результат сегментации методом GrabCut

В целом, GrabCut позволяет выделять передний план объектов на изображении, даже в случаях, когда объекты имеют сложную форму или схожие цвета с фоном. Однако этот метод требует некоторого взаимодействия с пользователем для указания начальных меток.

Контрольные вопросы

1. Что представляет собой граф в методах на основе графов?
2. В чем основное отличие между Normalized Cut и Min-Cut Segmentation?
3. Какая оптимизационная задача решается в методе Normalized Cut для разбиения графа на сегменты?
4. Что такое GrabCut? Какие основные шаги включает этот алгоритм?

Лабораторная работа

1. Измените параметры (например, compactness и n_segments) метода SLIC в представленном коде. Как это влияет на качество сегментации?
2. Рассмотрите примеры практического применения сегментации на основе графов на изображениях из различных областей. Сделайте выводы.

Глава 6. Сегментация методом активных контуров

Сегментация изображений на основе активного контура (Active Contour Segmentation, также известна как Snake Segmentation) – это один из методов компьютерного зрения для выделения объекта на изображении путем создания замкнутой кривой (контура), которая охватывает границы объекта.

Основная идея метода – определить контур границы объекта, который нужно выделить на изображении. Контур представляют в виде замкнутой кривой или поверхности и перемещают по изображению до тех пор, пока он не достигнет определенного состояния. Это состояние находят путем минимизации энергии контура с использованием различных функционалов.

Активный контур состоит из двух основных компонентов: модели формы и модели энергии. Модель формы представляет собой математическое описание формы объекта, который нужно выделить. Модель энергии определяет, какие точки на изображении должны быть притянуты или отдалены от контура, чтобы определить его конечную форму. Модель энергии также может включать различные функционалы, такие как градиент яркости, кривизну контура и силу натяжения.

Алгоритм активного контура состоит из нескольких этапов.

1. *Инициализация*. Сначала задаём начальное положение контура объекта либо исходя из его расположения на изображении, либо автоматически.
2. *Вычисление энергии*. Контур представим в виде математической кривой, например, сплайн-кривой, которую установим на изображении вокруг объекта. Далее вычислим энергию контура, которая состоит из двух частей: внутренней энергии и внешней энергии. Значение внутренней энергии определим на основе свойств контура, таких как его длина, кривизна и так далее. Цель внутренней энергии – сделать контур гладким и замкнутым. Внешнюю энергию вычислим на основе свойств изображения, таких как яркость, текстура, градиент и так далее.

3. *Обновление контура*. Изменяем форму и положение контура так, чтобы контур повторял границу объекта.
4. *Завершение алгоритма*. Останавливаем алгоритм, когда контур перестает изменять свою форму и расположение, что означает соответствие контура границам объекта.

Преимущества метода активного контура включают:

- Способность выделять объекты со сложными формами и текстурами.
- Гибкость и адаптивность алгоритма к изменениям контура объекта.
- Возможность автоматической сегментации без необходимости ручной настройки параметров алгоритма.

Однако метод активного контура также имеет некоторые недостатки:

- Сложность подбора параметров алгоритма, которые могут сильно влиять на результат сегментации.
- Чувствительность к начальному положению контура, что может привести к ошибочной сегментации.
- Недостаточная точность для выделения мелких деталей объекта.

Ниже приведём реализацию данного метода сегментации с помощью библиотеки skimage.

```
import numpy as np
import cv2
import skimage.segmentation as seg

# Загрузите изображение
image = cv2.imread('image.jpg', cv2.IMREAD_GRAYSCALE)

def circle_points(resolution, center, radius):
    """
    Генерация точек, находящихся на границе круга.
    """
    radians = np.linspace(0, 2*np.pi, resolution)
```

```
        # полярные координаты
        c = center[1] + radius*np.cos(radians)
        r = center[0] + radius*np.sin(radians)
        return np.array([c, r]).T

    # Исключение последней точки, поскольку замкнутый круг не
должен иметь повторяющихся точек
    points = circle_points(resolution=300, center=[170, 170],
radius=110)[:-1]

    # Сегментация активного контура
snake = seg.active_contour(image, points, alpha=0.2, beta=0.5)

    # Визуализация
    fig, ax = plt.subplots(nrows=1, ncols=1, figsize=(7, 7))
    ax.imshow(image, cmap='gray')
    ax.axis('off')
    ax.plot(points[:, 0], points[:, 1], '--r', lw=3)
    ax.plot(snake[:, 0], snake[:, 1], '-g', lw=3);
```

В данной программе используем функцию circle_points для вычисления координат *x* и *y* точек на границе круга. Так как в нашем случае resolution = 300, то функция вернет 300 таких точек. Затем осуществляем сегментацию объекта на изображении с помощью функции seg.active_contour() с параметрами под названием alpha и beta. Разная настройка параметров функции позволяет управлять построением замкнутой кривой вблизи границы объекта. Например, более высокие значения alpha заставляют кривую сокращаться быстрее, тогда как большие значения beta делают контур более гладким. На рисунке 19 приведен пример выделения контура объекта.

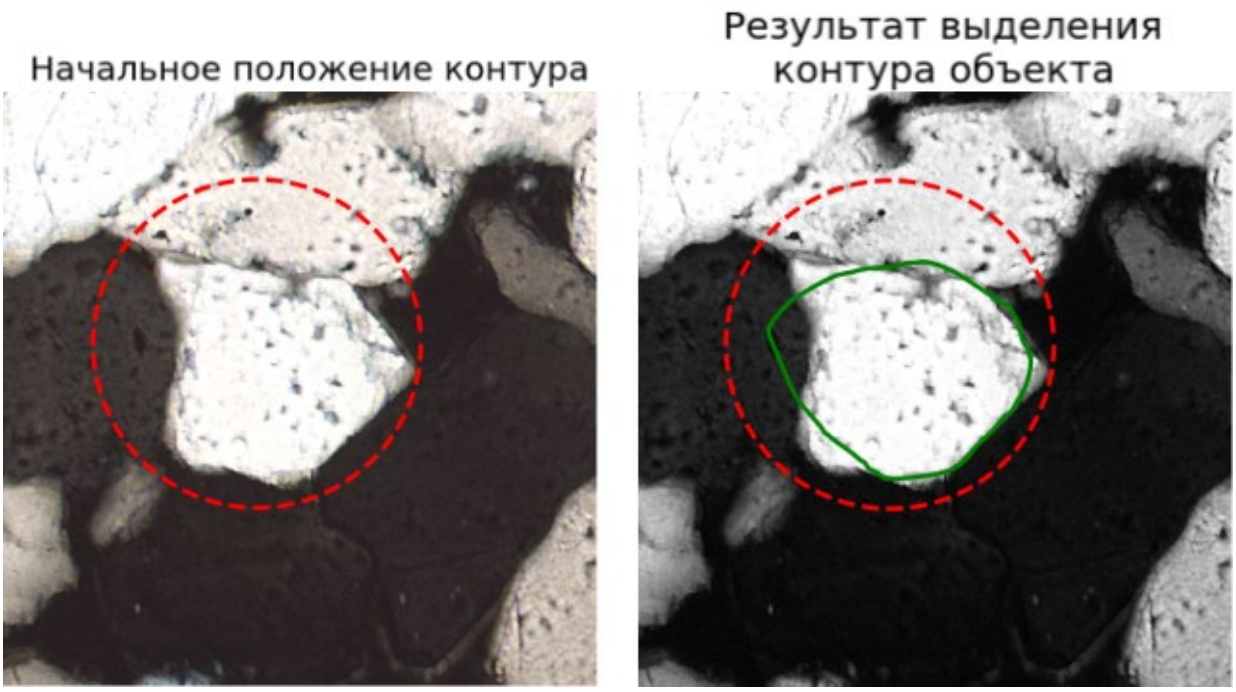


Рисунок 19 – Выделение объекта на основе активного контура. Красный – начальное положение контура, зеленый – результат

Контрольные вопросы

1. Что такое метод активного контура (Snake Segmentation)?
2. Какова основная цель метода активного контура?
3. Какие этапы включает алгоритм активного контура?
4. Какие преимущества и недостатки имеет метод активного контура?

Лабораторная работа

1. Примените метод активного контура к объектам на изображении с различными начальными формами контура (прямоугольник, треугольник, эллипс и т.д.).
2. Проведите исследование по оптимизации параметров alpha и beta для конкретного изображения.

Список литературы

1. Otsu N., A threshold selection method from gray-level histograms / N. Otsu // IEEE transactions on systems, man, and cybernetics. – 1979. – Vol. 9. – №. 1. – P. 62-66. – doi: 10.1109/TSMC.1979.4310076.
2. Wellner P. D., Adaptive thresholding for the DigitalDesk / P. D. Wellner // Xerox, EPC1993-110. – 1993. – Vol. 404. – 17 p.
3. Bradley D., Adaptive thresholding using the integral image / D. Bradley, G. Roth //Journal of graphics tools. – 2007. – Vol. 12. – №. 2. – P. 13-21. – doi: 10.1080/2151237X.2007.10129236.
4. Muerle, J. L., Experimental evaluation of techniques for automatic segmentation of objects in a complex scene / J. L. Muerle, D. C. Allen // Pictorial pattern recognition. – 1968. – P. 3-13.
5. Adams R., Seeded region growing / R. Adams, L. Bischof // IEEE Transactions on pattern analysis and machine intelligence. – 1994. – Vol. 16. – No. 6. – P. 641-647.
6. Horowitz S. L., Pavlidis T. Picture segmentation by a tree traversal algorithm / S. L. Horowitz, T. Pavlidis // Journal of the ACM (JACM). – 1976. – Vol. 23. – No. 2. – P. 368-388.
7. Beucher S., Use of watersheds in contour detection / S. Beucher, C. Lantuejoul // Proc. Int. Workshop on Image Processing, Sept. 1979. – 1979. – 12 p.
8. Sobel I., History and Definition of the Sobel Operator. [Электронный ресурс] // Researchgate.net: научно-информационная соц. сеть, 2014. URL: https://www.researchgate.net/publication/239398674_An_Isotropic_3_3_Image_Gradient_Operator (дата обращения: 18.11.2023).
9. Canny J., A computational approach to edge detection / J. Canny // IEEE Transactions on pattern analysis and machine intelligence. – 1986. – Vol. 6. – P. 679-698. – doi: 10.1109/TPAMI.1986.4767851.
10. Обнаружение объектов методом Оцу: [Электронный ресурс]. URL: https://habr.com/en/articles/112079/

11. Метод активных контуров для сегментации изображений: [Электронный ресурс].°URL:°https://imaging.cs.msu.ru/files/courses/varmethods2022/varmethods2022-book-2.pdf
12. Предварительная обработка изображений: [Электронный ресурс]. URL: https://api-2d3d-cad.com/processingimages/
13. Python OpenCV – морфологические операции: [Электронный ресурс].°URL:°https://bestprogrammer.ru/programmirovanie-i-razrabotka/python-opencv-morfologicheskie-operatsii
14. Введение в сегментацию изображений с помощью кластеризации K-средних: [Электронный ресурс]. URL: https://machinelearningmastery.ru/introduction-to-image-segmentation-with-k-means-clustering-83fd0a9e2fc3/
15. Обзор алгоритмов сегментации: [Электронный ресурс]. URL: https://habr.com/ru/companies/intel/articles/266347/
16. Осваиваем компьютерное зрение — 8 основных шагов: [Электронный ресурс]. URL: https://habr.com/ru/articles/461365/
17. Семантическая сегментация: краткое руководство: [Электронный ресурс]. URL: https://neurohive.io/ru/osnovy-data-science/semantic-segmention/
18. Обзор основных алгоритмов выделения контуров изображения: [Электронный ресурс]. URL: https://newtechaudit.ru/obzor-osnovnyh-algoritmov-vydeleniya-konturov-izobrazheniya/

Электронное учебное издание
сетевого распространения

Тумаков Дмитрий Николаевич
Каюмов Зуфар Дамирович
Маркина Ангелина Геннадьевна
Хайруллина Дина Ибрагимовна
Егорчев Антон Александрович

ОСНОВНЫЕ АЛГОРИТМЫ СЕГМЕНТАЦИИ ИЗОБРАЖЕНИЙ

Учебное пособие

Подписано к использованию 23.08.2024.
Гарнитура «Times New Roman».
Заказ 58/8

Издательство Казанского университета

420008, г. Казань, ул. Профессора Нужина, 1/37
тел. (843) 206-52-14 (1704), 206-52-14 (1705)