# Обработка и распознавание изображений

Лабораторная работа №2. Изучение и освоение методов классификации формы изображений.

Выполнил: Алексеев Илья Алексеевич, 317 группа.

## Постановка задачи

Дано:

- фотографии графов, построенных из магнитной головоломки
- все графы делятся на четыре класса, соответствующие классам изоморфизма графов (подробнее)
- фотографии имеют размер $1024 \times 768$ и качество $72$ dpi

Найти:

- класс графа по фотографии
- сколько вершин разных степеней (подробнее)

## Решение

### Алгоритм действий

- Сегментация графа и построение скелета формы
    1. Удаление теней (источник). Для каждого RGB-канала в отдельности:
        1. Дилатация
        2. Медианное размытие
        3. Нормализация
    2. Бинаризация:
        1. Размытие Гаусса
        2. Пороговое отсечение методом Отсу
        3. Дилатация
        4. Эрозия
    3. Построение скелета изображения (средствами пакета `skimage.morphology`)

- Анализ скелета
    1. Поиск всех ветвей скелета и их пересечений (средствами пакета `skan.csr`)
    2. Найденные ветви объявляем рёбрами графа, пересечения — вершинами графа
    3. Классификация найденных рёбер графа на обычные, промежуточные и мусорные
        - подробнее в разделе “Поиск ветвей и пересечений. Классификация рёбер”
    4. Подсчёт степеней вершин на основе того, какой тип у примыкающих рёбер
        - Подробнее в разделе “Подсчет степеней вершин”

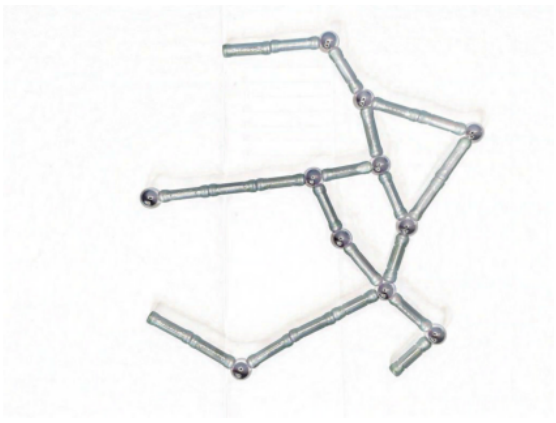
Все функции, связанные с сегментацией и анализом скелета, реализованы в файлах `segmentation.py` и `skeleton.py`. Примеры работы с этими функциями представлены в ноутбуках `segmentation.ipynb` и `skeleton.ipynb`.

## Удаление теней

Демонстрация:

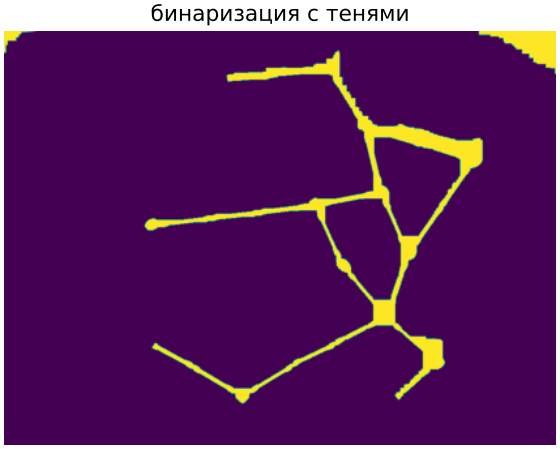
исходное изображение

без теней



Данный шаг значительно улучшает качество сегментации:

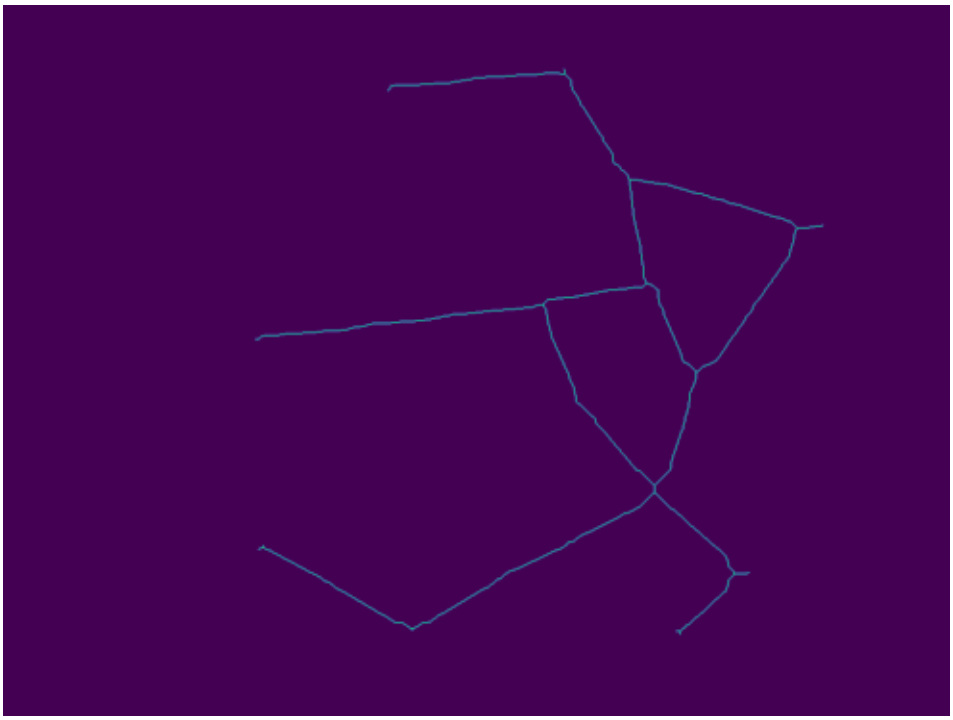
бинаризация с тенями



бинаризация без теней

## Построение скелета

Пример скелета формы

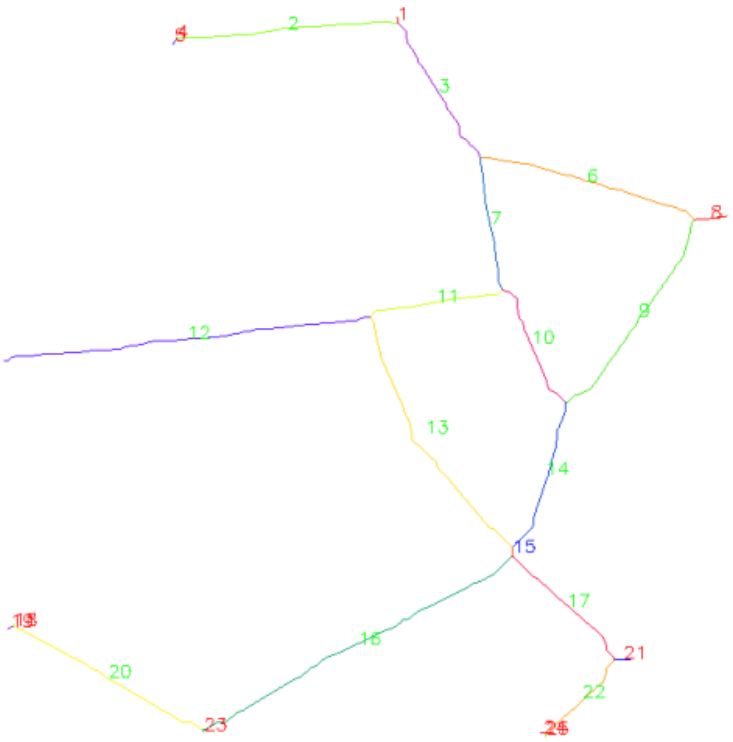


## Поиск ветвей и пересечений. Классификация ветвей

Поиск ветвей и пересечений производится полностью средствами `skan.csr.Skeleton`.

Изобразим найденные ветви скелета:

Классификация ветвей




Заметим, что не все ветви соответветствуют рёбрам графа. А именно присутствуют короткие ветви, возникшие на месте

- выступавших шариков (ветви с красными номерами)
- шариков с более чем тремя примыкающими рёбрами (одна ветвь с синим номером, номер 15)

Введём несколько определений:

1. *Пиксельное расстояние между соседними пикселями* будем считать
    - равным $1$, если пиксели являются соседними по вертикали или горизонтали
    - равным $\sqrt{2}$, если пиксели являются соседними по диагонали
2. *Пиксельной длиной ветви скелета* будем называть сумму пиксельных расстояний между пикселями данной ветви.
3. *Степенью пикселя* назовём число соседних с ним пикселей (и по диагонали, и по горизонтали)

Все ветви скелета делятся на три вида:

- обычные (зеленые номера на рисунке) — пиксельная длина ветви больше некоторого порога

- промежуточные (синие номера на рисунке) — пиксельная длина ветви меньше некоторого порога и концевые пиксели имеют степени больше 1
- мусорные (красные номера на рисунке) — пиксельная длина ветви меньше некоторого порога и хотя бы один из концевых пикселей имеет степень 1

## Подсчёт степеней вершин

Заметим, что

- если к данному пикселю примыкают только обычные ветви скелета, то степень соответствующей вершины равна степени данного пикселя
- примыкающие мусорные ветви не дают вклад в степень вершины
- примыкающие промежуточные ветви вносят вклад в степень вершины, равный сумме степеней вершин связного подграфа из промежуточных ветвей

Эти замечания выливаются в алгоритм подсчета степеней вершин:

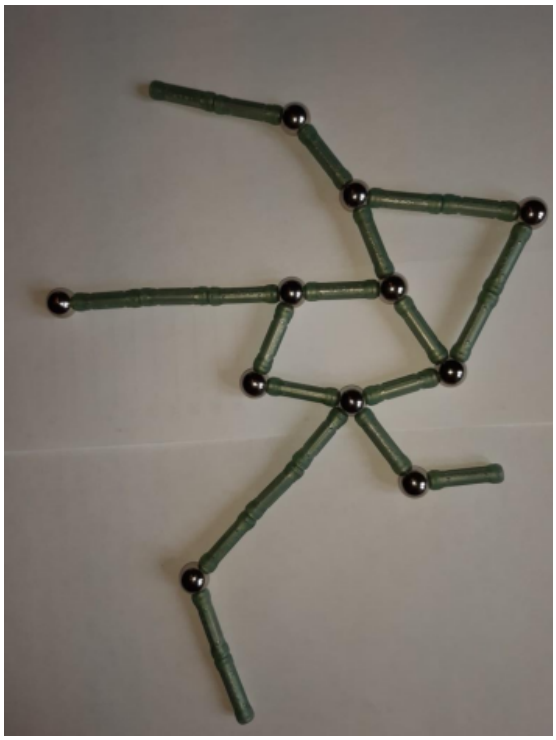
- сначала для каждой вершины
  - запоминаем число обычных рёбер, примыкающих к данной вершине
- затем для каждой вершины
  - если среди примыкающих ребёр есть промежуточные, то
    - посчитать сумму ранее посчитанных степеней всех вершин связного подграфа из промежуточных путей
  - слить все вершины данного подграфа в одну

Простота программной реализации во многом была достигнута благодаря объекто-ориентированному программированию: использованы сущности `Node` и `Edge` с интерфейсами для обновления примыкающих путей. Сумма степеней в подграфах вычисляется с помощью обхода в глубину.

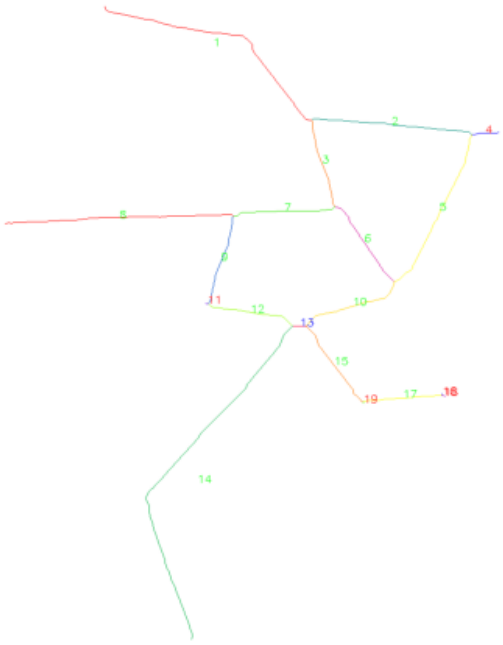
## Улучшение детектирования ветвей

Рассмотрим пример:

исходное изображений



детектированные ветви



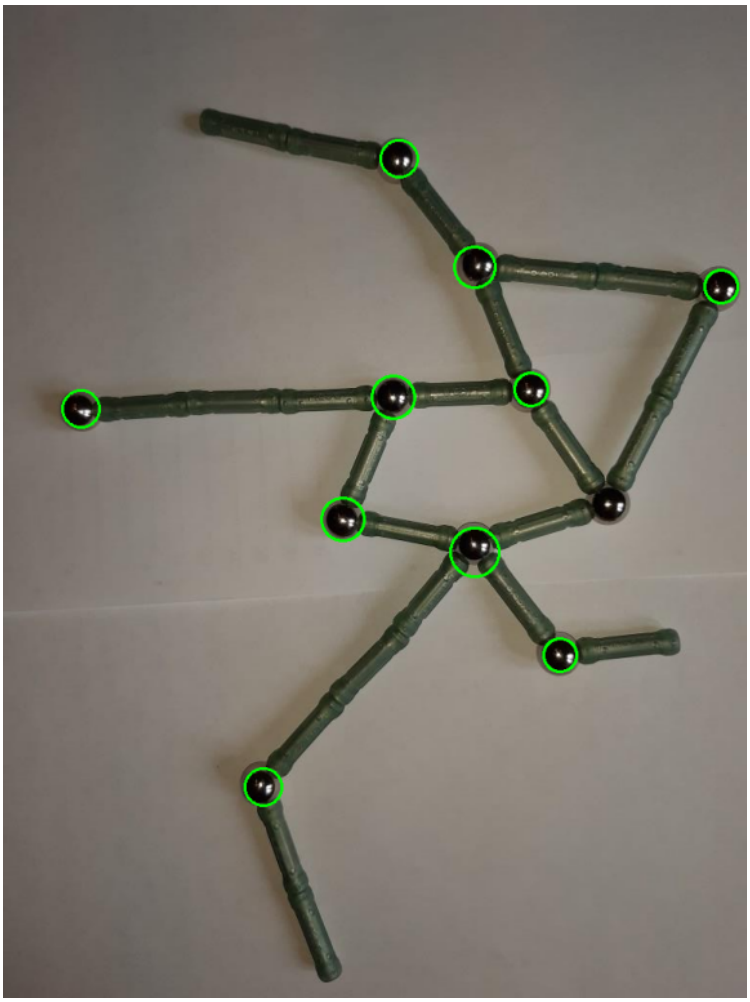

В данном случае ветвь 1 на самом деле является двумя рёбрами, соединёнными вершиной степени 2 (поскольку на исходном изображении они были разделены шариком). Как и ветвь 14.

Чтобы исправить это, была предложена следующая модификация:

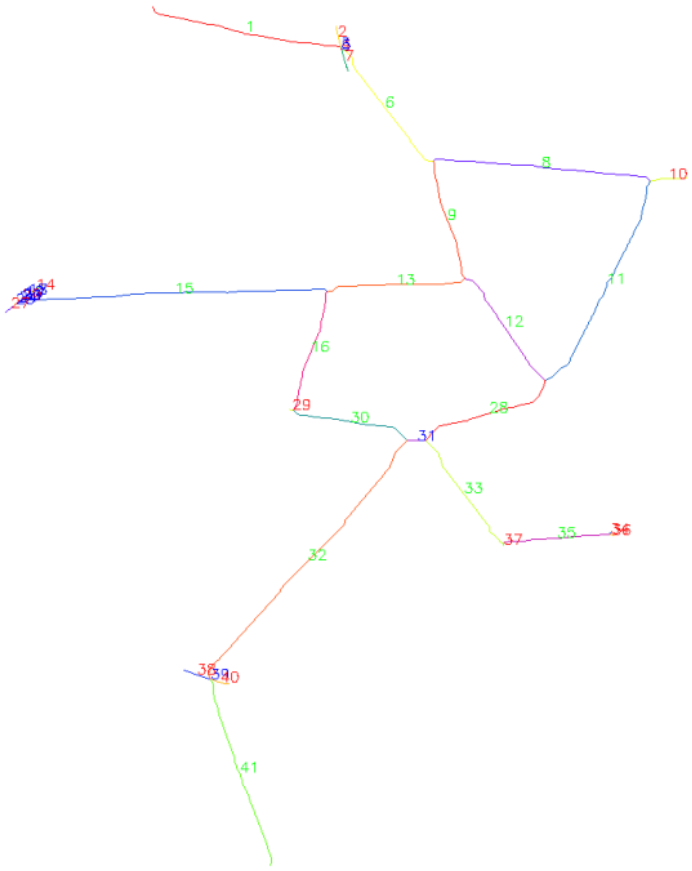
- на скелет формы добавить “засечки” в те места, где два ребра соединены вершиной степени 2
- такие места можно найти с помощью детектирования окружностей на исходном изображении
    1. Перевод цветного изображения в полутоновое
    2. Размытие Гаусса
    3. Детекция окружностей с помощью алгоритма Хаффа
- “засечка” — это отрезок, пересекающий ветвь скелета
    - угол, под которым отрезок пересекает ветвь выбирается случайно из равномерного распределения на отрезке $[0^\circ, 180^\circ]$
    - такой выбор угла не гарантирует, что отрезок не будет сливаться с ветвью

Результат работы алгоритма Хаффа:

Детекция окружностей



Демонстрация засечек:




Такая модификация не всегда решает проблему и, более того, может “испортить” другие вершины. Решить такой нюанс можно, например, следующим образом: запустить алгоритм $k$ раз и провести голосование между всеми результатами. Этот алгоритм используется далее для проверки.

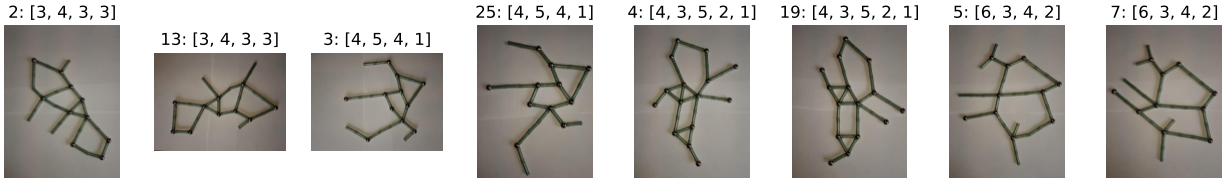
# Тест

Тест проводился на выборке из восьми изображений (папка `images`).

Пример результатов работы:

```
2:  [3, 4, 3, 3]    [3, 4, 3, 3]    [3, 4, 3, 3]    [3, 4, 3, 3]    [3,
4, 3, 3]
13: [3, 4, 2, 4]    [3, 4, 3, 3]    [3, 4, 3, 3]    [3, 4, 3, 3]    [3,
4, 3, 3]
3:  [4, 5, 4, 1]    [4, 5, 4, 1]    [4, 5, 4, 1]    [4, 5, 4, 1]    [4,
5, 4, 1]
25: [4, 5, 4, 1]    [4, 5, 4, 1]    [5, 5, 4, 1]    [6, 4, 5, 1]    [4,
5, 4, 1]
4:  [3, 4, 5, 2, 1] [4, 3, 5, 2, 1] [4, 3, 5, 2, 1] [4, 3, 5, 2, 1] [4,
3, 5, 2, 1]
19: [4, 3, 5, 2, 1] [6, 3, 5, 2, 1] [3, 4, 5, 2, 1] [4, 3, 5, 2, 1] [5,
4, 5, 2, 1]
5:  [6, 3, 4, 2]    [6, 3, 4, 2]    [6, 3, 4, 2]    [6, 3, 4, 2]    [6,
3, 4, 2]
7:  [6, 3, 4, 2]    [6, 3, 4, 2]    [6, 3, 4, 2]    [6, 3, 4, 2]    [6,
3, 4, 2]
```

Ответы:




Видим, что все фотографии снабдены правильными описаниями. Отображение таких описаний на множество меток классов — задача тривиальная.

# Итоги

Самые сложные части решения — это построение скелета и поиск в нем ветвей — реализованы в пакетах `skimage.morphology` и `skan.csr`. Самая трудоёмкая часть решения — обход всего скелета для подсчёта степеней вершин. Также значительная доля результата не была бы получена без модификации с засечками и голосованием. По итогу получилось реализовать генерацию корректного признакового описания фотографии графа.