# Отчет по распознаванию структуры графов

## Постановка задачи

Задача заключается в разработке и реализации алгоритма для распознавания на фотографии графа определенного класса с помощью некоторого признакового описания этого графа. В качестве входа программа получает фотографии с графом, далее сегментирует изображения на основе точечных и пространственных преобразований, генерирует признаковое описание структуры  графов на фотографиях и выводит классы, к которым относятся графы.

## Описание данных

В качестве входных данных (данных для обучения) дается набор из цветных изображений с графами, построенными из магнитных деталей игры-головоломки в формате $1024 \times 768$ с разрешением 72 dpi. Всего есть четыре класса графов, примеры которых представлены на изображениях: 2.jpg (класс 1, рис. 1) , 3.jpg (класс 2, рис. 2), 22.jpg (класс 3, рис. 3), 9.jpg (класс 4, рис. 4). На этих рисунках представлены фотографии для двух классов задач: **Intermediate** (на белом фоне), **Expert** (на пестром фоне). Эти задачи все были решены и по постановке отличаются только фоном: белый и пестрый. Остальные фотографии представляют собой изображения с такими же фонами и графами, изоморфными четырём эталонным образцам (рис. 1- рис. 4)

Рис. 1

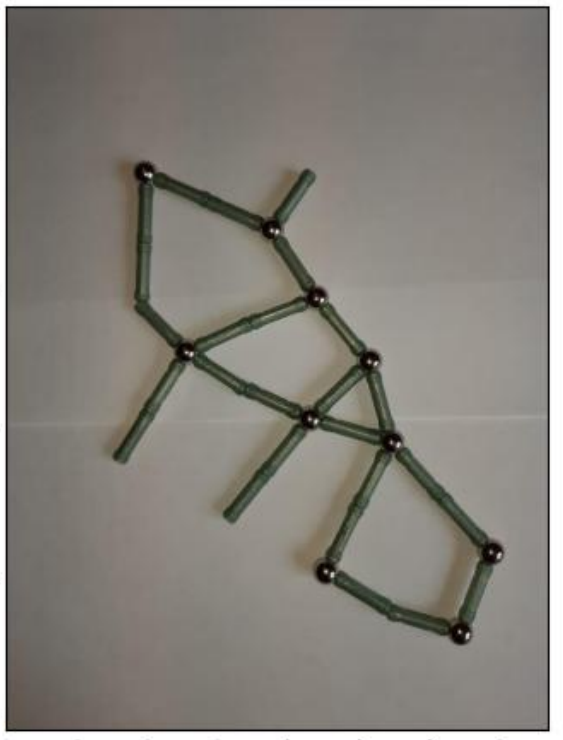


Рис. 2

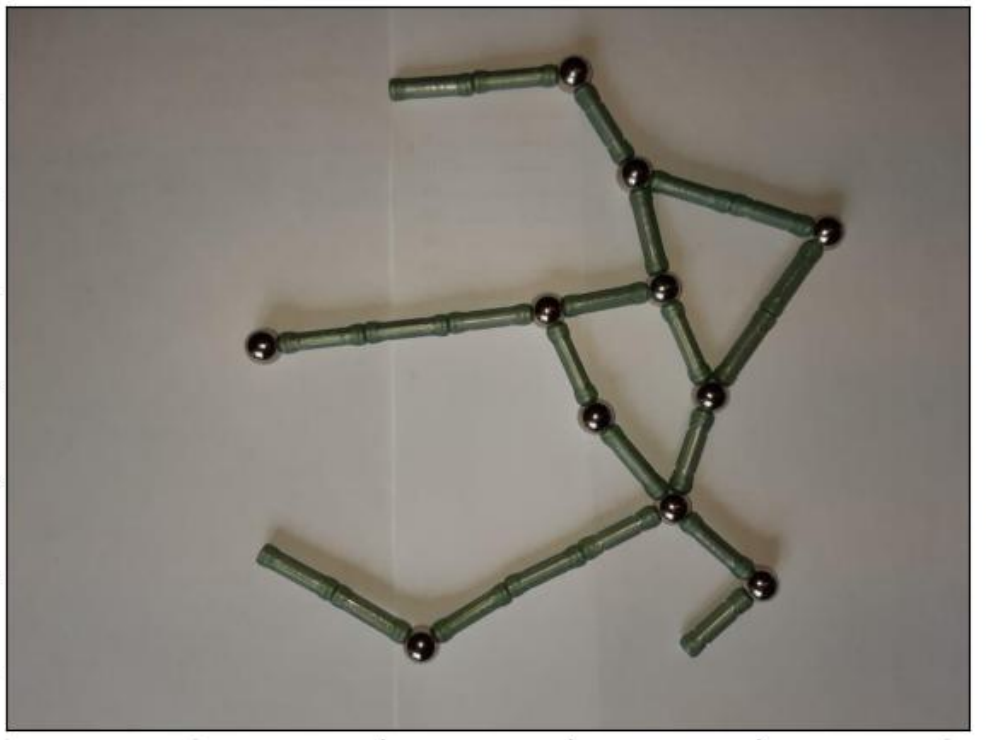


Рис. 3

Рис. 4

## Описание метода решения

Для решения задачи необходимо было определиться с тем, какое признаковое описание графа наиболее подходящее, чтобы можно было однозначно детектировать класс графа. Можно заметить, что, чтобы однозначно опередилить класс, необходимо знать только количество вершин степени 3 и 4: класс 1 - 3 вершины степени 3, 3 вершины степени 4, класс 2 - 4 вершины степени 3, 1 вершина степени 4, класс 3 - 5 вершин степени 3, 2 вершины степени 4, класс 4 - 4 вершины степени 3, 2 вершины степени 4. Алгоритм состоит из нескольких этапов, а

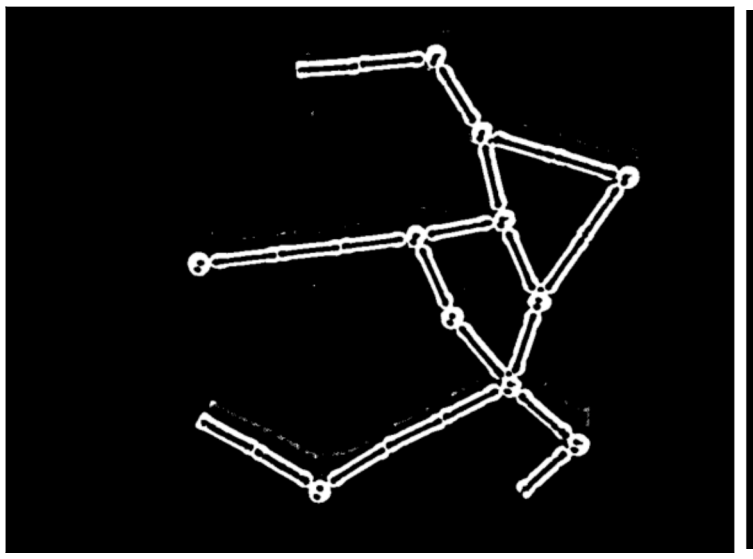
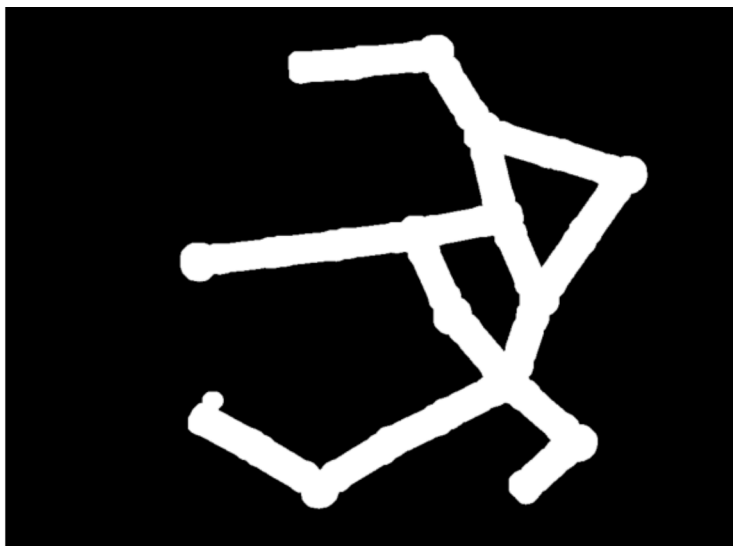
именно:

1. Выделение графа
2. Построение признакового описания графа
3. Классификация графа

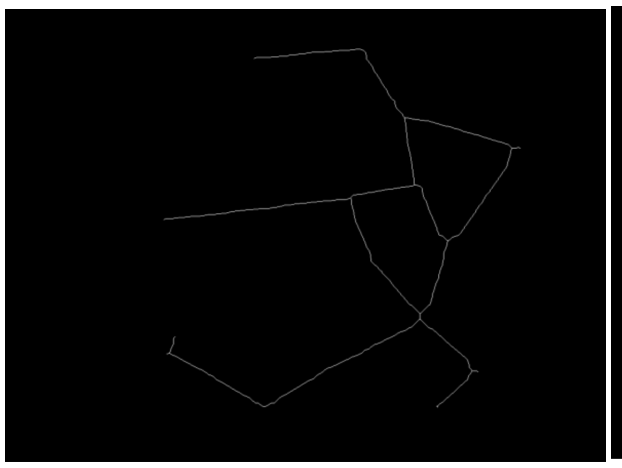
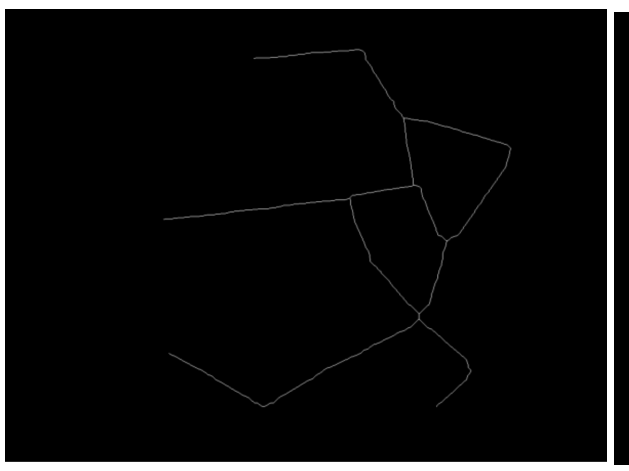
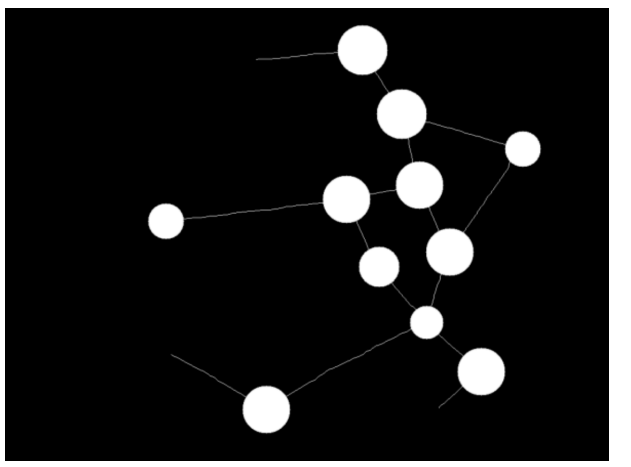
Задача была решена как для уровня **Expert** так и для **Intermediate**, однако для реализации понадобились две программы каждая, из которых отличается лишь начальными этапами, поэтому, сначала будет разделение по типу задачи, а далее описаны общие принципы работы для обеих программ. В качестве примеров, иллюстрирующих работу алгоритмов, возьмем для **Intermediate** 3.jpg (рис. 1), а для **Expert** 22.jpg (рис. 3).

## 1.1 Выделение графа (Intermediate).

Сначала изображение изменили на черно-белое, затем, оно размывалось с помощью медианного фильтра, что позволило корректно выделить центры шаров (вершины), используя преобразование Хафа. После изображение бинаризовалось для того, чтобы найти скелет графа, однако при бинаризации возникали шумы внутри графа (черный шум) и снаружи (белый шум). Для того, чтобы убрать белый шум использовалась эрозия, после чего дилатация для заполнения всех черных дыр внутри графа.

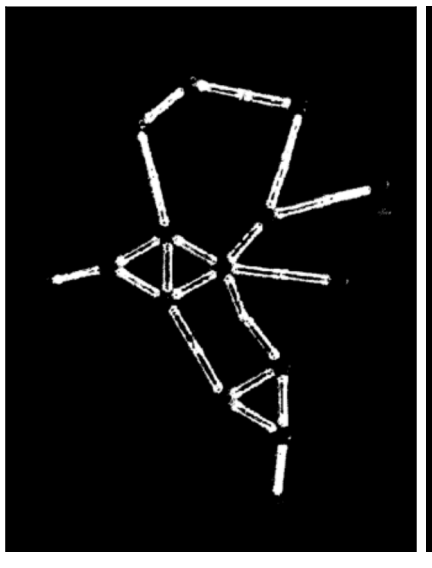
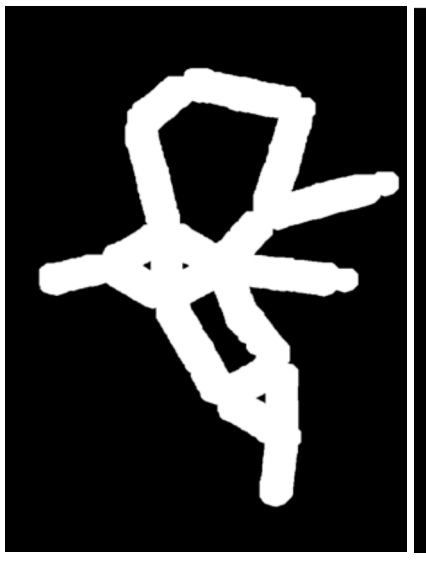
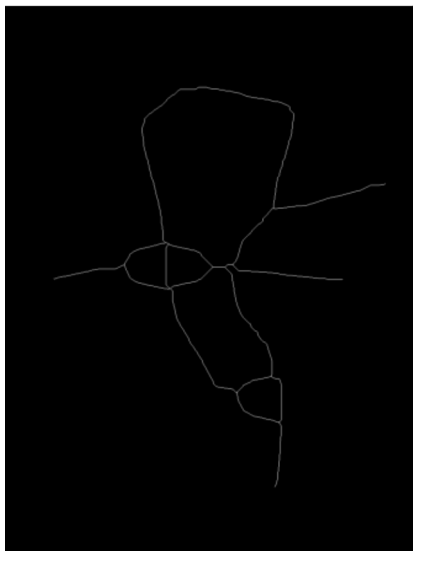



Затем в графе выделялся скелет и его "стригли", чтобы побочные ветви не мешали при определении степеней вершин. На псоледней фотографии выделен граф с найденными вершинами.

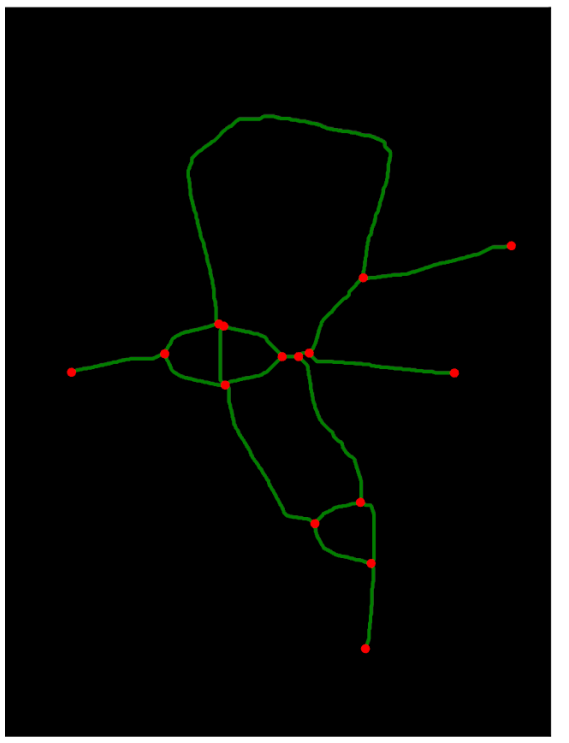




## 1.2 Выделение графа (Expert).

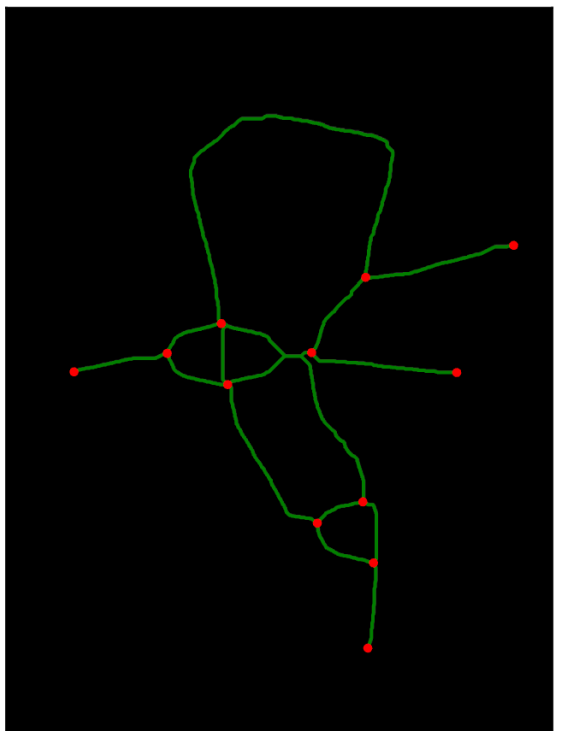
Поскольку в **Expert** изображения на пестром фоне, то обычной бинаризацией выделить граф не получится, в этом случае надо выделять цвета. Был выделен бирюзовый цвет (цвет ребер) с помощью перевода в hsv пространство и определения диапозона данного цвета, далее убирались шумы с помощью эрозии и применялась дилатация для заполнения пропусков, возникающих как из-за шума так и из-за вершин, поскольку они серебряные. После, как и в **Intermediate**, строился скелет, который впоследствии "стригся" (ниже приведен уже подстриженный скелет).

Если в **Intermediate** получилось выделить все вершины с помощью преобразования Хафа, то здесь с этим было много проблем, поэтому брался немного другой подход, а именно граф из скелета преобразовывался в представление графа из библиотеки *networkx*, тем самым находились не все вершины, а наиболее походящие на вершины графа: должно быть не слишком гладкое соединение (вершины степени два, например, не получилось найти из-за сильной дилатации, которая привела к гладкости, что видно на изображениях выше и ниже)

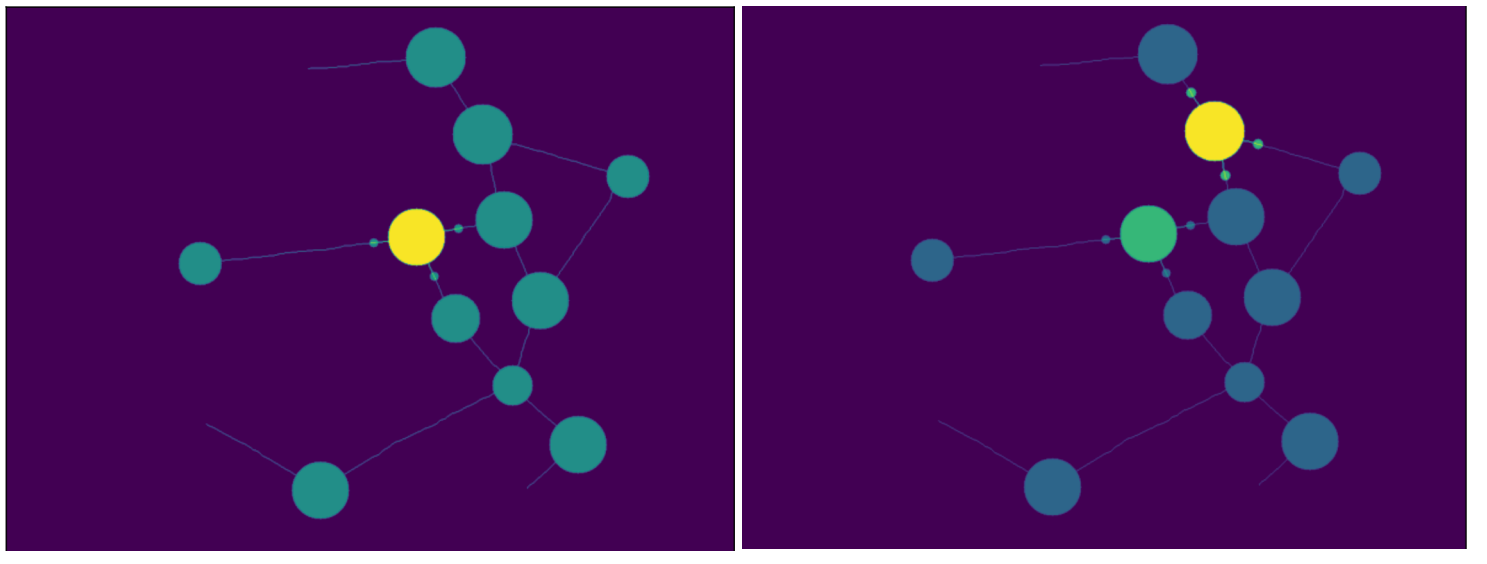


После такого преобразования, оказалось, что выделяются лишние вершины, которые также стали следствием сильной дилатации. Эта проблема решалась довольно просто: находилось растояния между всеми найденными вершинами и те вершины, которые стоят рядом друг с другом сливаются в какую-то одну. В результате получилось следующее:

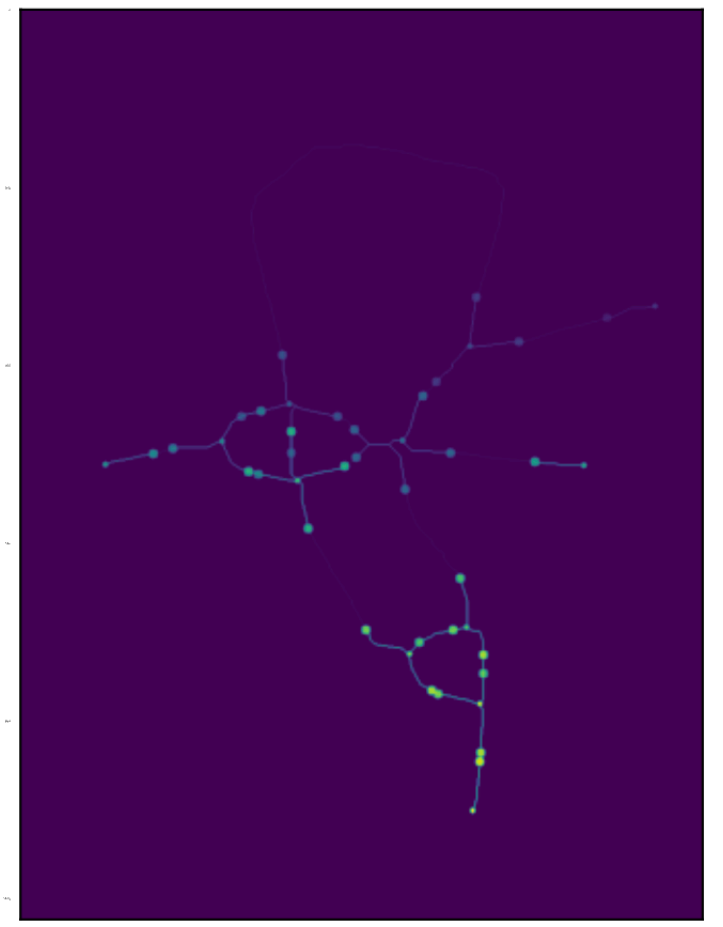


## 2. Построение признакового описания графа

Далее **Intermediate** и **Expert** работают одинаково. Итак, с помощью найденных вершин необходимо найти количество вершин степени три и степени четыре. Для решения этой проблемы делалось следующее: представим, что найденные вершины это источники распространения волны во все стороны, тогда будем распространять пиксели от каждой вершины и, если они удалились на достаточно далекое растояние от вершины и при этом по-прежнему находятся на графе, то закрашиваем их маленькими кругами и увеличиваем степень вершины. На изображениях приведены два шага работы алгоритма. Большие круги - это магнитные шарики.



И приведем результат работы для **Expert**:



## 3. Классификация изображений

По найденным степеням вершин, с помощью признакового описания, которое определялось в самом начале данного параграфа, с легкостью определяются классы графов

# Программная реализация

Задание сдается в виде архива ***solution.zip***. Прежде всего его надо разархивировать. Программы написаны на Python с помощью Jupyter Notebook.  Для корректной работы необходимо в images\Expert  и images\Intermediate (внутри solution) поместить фотографии для соответствующих задач. Программы называются task2_expert.ipynb и task2_intermediate.ipynb и написаны соответственно для **Expert** и **Intermediate**. Для того, чтобы получить классифицированные графы необходимо запустить все ячейки ноутбука ***task2_expert.ipynb*** или ***task2_intermediate.ipynb***, ответ будет в последней ячейке.

**Результат работы программы:**

```
# находим массив со всеми степенями вершин
ans = get_description_graph(src)

# для использования массива как ключа в словаре, переводим его в сторку
ans_str = str(ans)
if ans_str not in graph_class:

    # создаем новый список под новый класс
    graph_class[ans_str] = list()

# добавляем граф в нужный класс
graph_class[ans_str].append(el)

# вывод ответа
print('\nANSWER:')
num_class = 0
for el in graph_class.values():
    num_class += 1
    print("CLASS", str(num_class) + ":", el)
```

[29] ✓ 1.4s Python

```
processing of image images/Expert/9.jpg
processing of image images/Expert/28.jpg
processing of image images/Expert/29.jpg
processing of image images/Expert/16.jpg
processing of image images/Expert/10.jpg
processing of image images/Expert/22.jpg
processing of image images/Expert/23.jpg
processing of image images/Expert/18.jpg

ANSWER:
CLASS 1: ['images/Expert/16.jpg', 'images/Expert/18.jpg']
CLASS 2: ['images/Expert/28.jpg', 'images/Expert/29.jpg']
CLASS 3: ['images/Expert/22.jpg', 'images/Expert/23.jpg']
CLASS 4: ['images/Expert/9.jpg', 'images/Expert/10.jpg']
```

PROBLEMS 58 OUTPUT DEBUG CONSOLE TERMINAL JUPYTER zsh

```
(base) nickpan@MacBook-Pro-Nick PROJECT %
```

Jupyter Server: Local Ln 1, Col 24 Spaces: 4 LF Cell 9 of 9

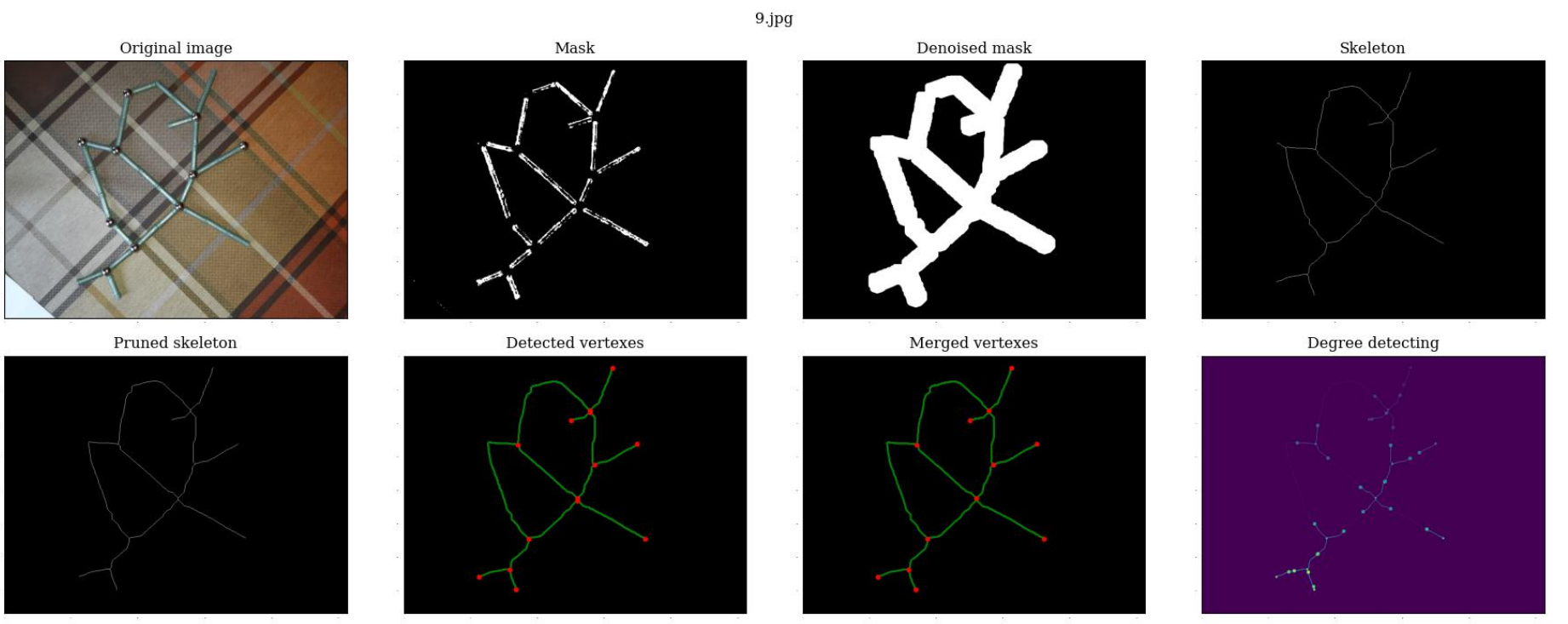
**Используемые библиотеки:**

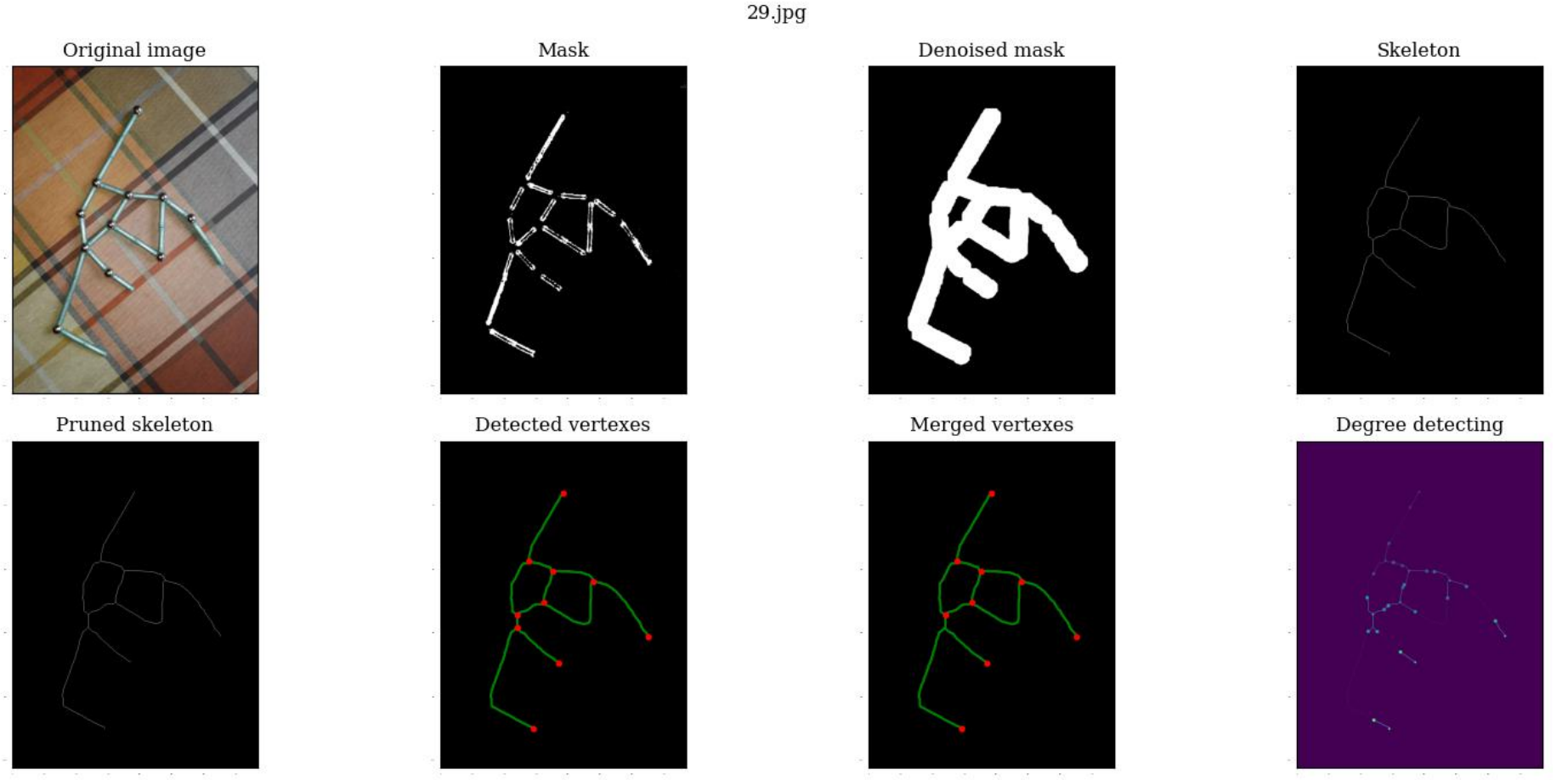
- glob (для работы с файлами)
- numpy (для работы с изображениями)
- skimage (для построения скелета)
- plantcv (для "стрижки" скелета)
- networkx (для работы с графом)
- sknw (для преобразования из скелета в граф networkx)
- cv2 (для обработки изображения и поиска вершин)
- sklearn (для поиска попарных растояний)

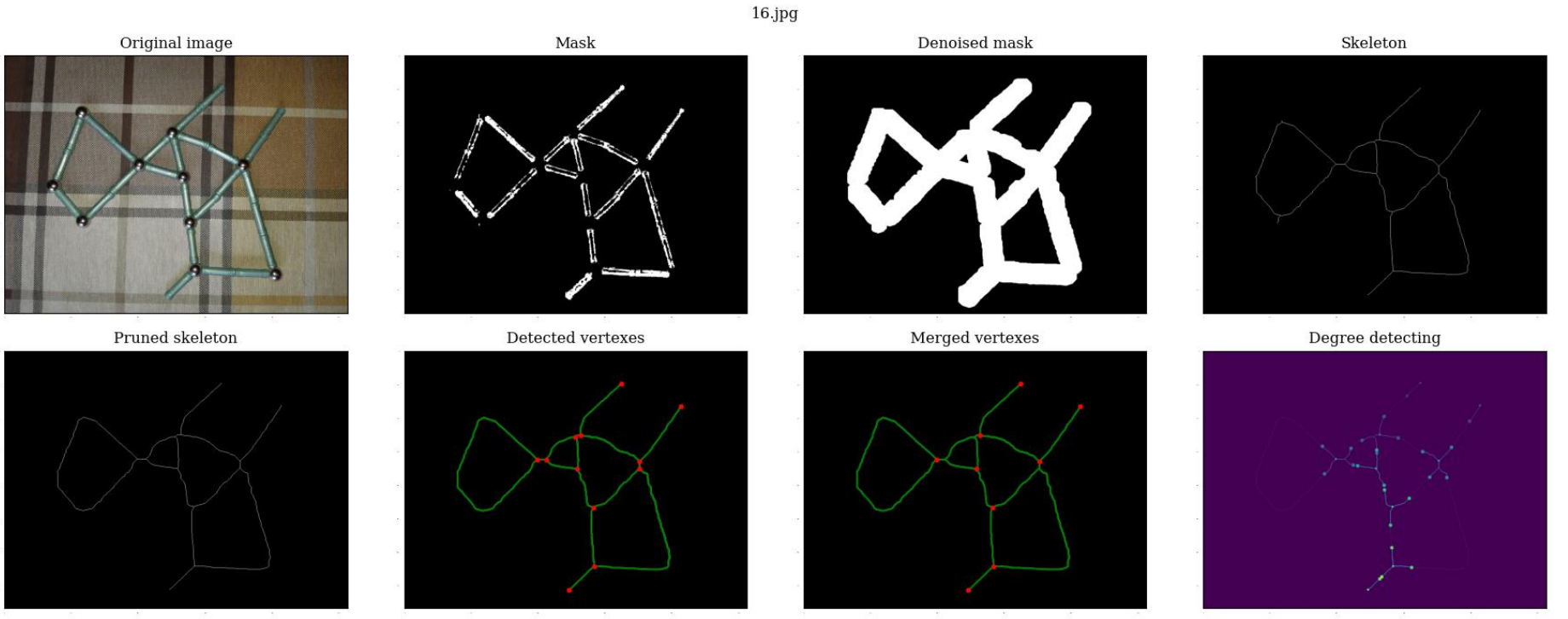
# Эксперименты

Продемонстрируем, как работает программа:

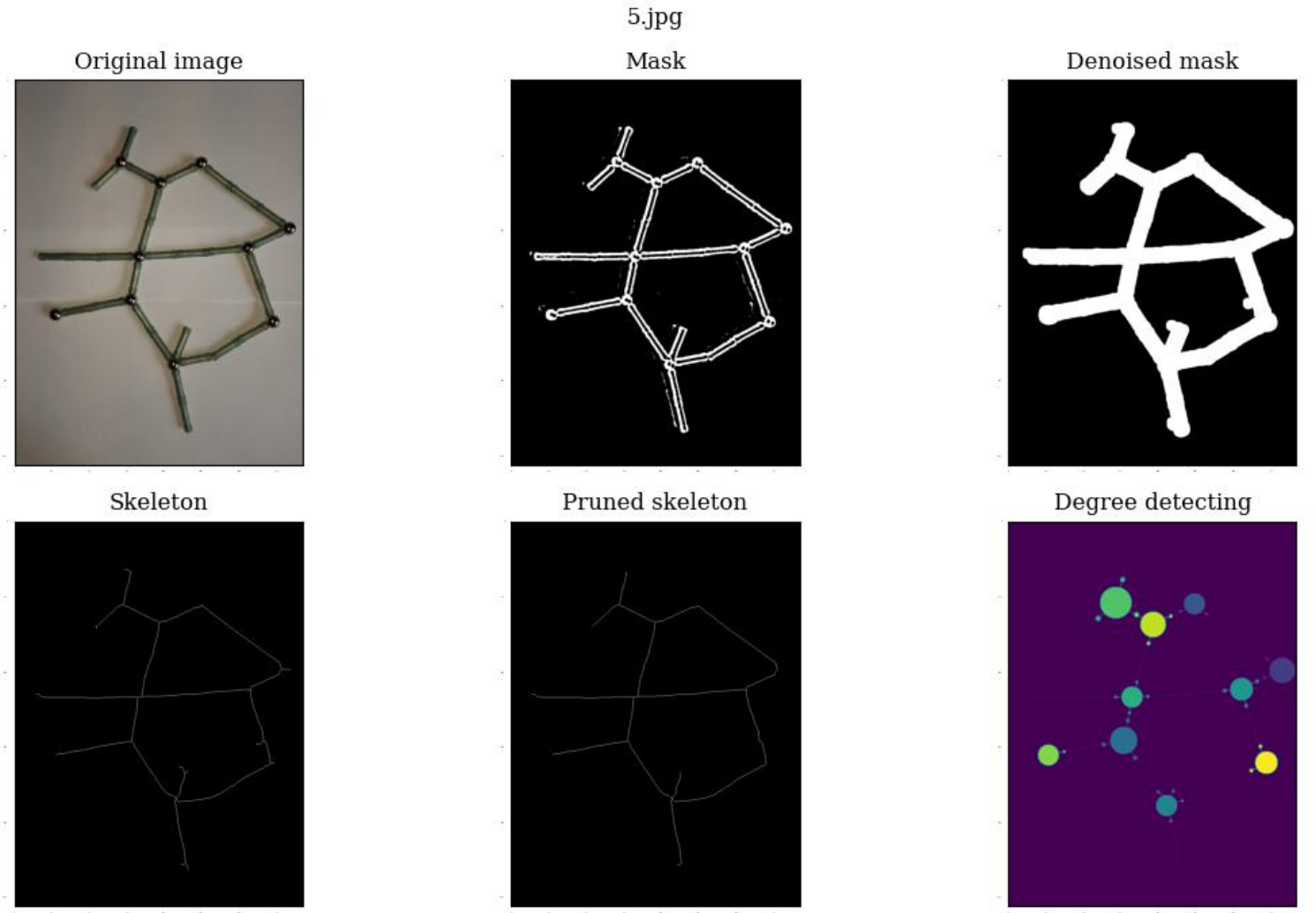
**Expert:**



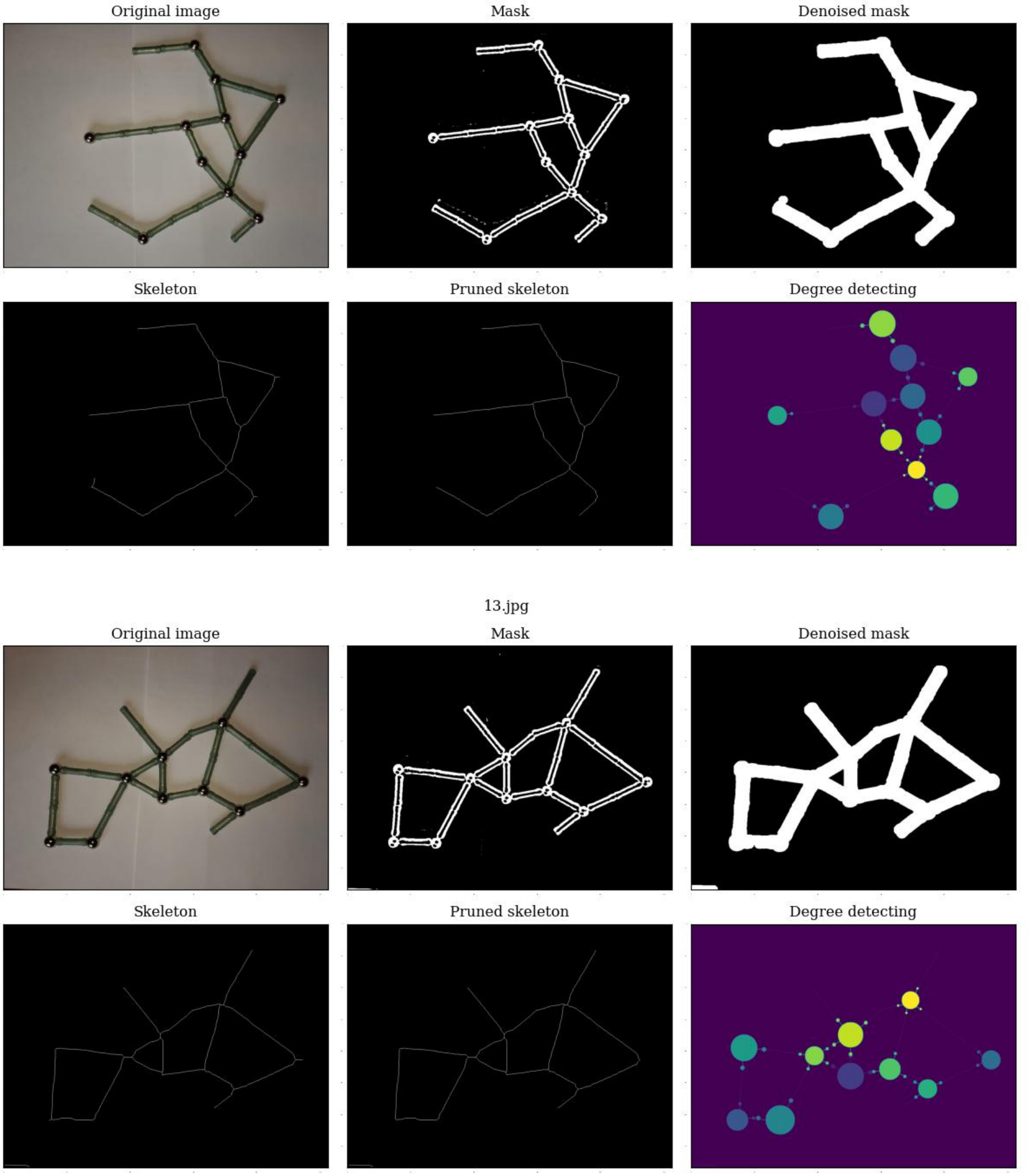






**Intermediate:**



3.jpg




13.jpg




## Выводы

В результате данной работы была решена задача распознавания графов, построенных из магнитных палочек и шариков. Из плюсов можно выделить необходимость в относительно небольшой выборке для реализации алгоритма, а также экономия времени, в случае если бы мы хотели обучить, например,  нейронную сеть или алгоритм из машинного обучения. При этом данный подход довольно ограничен тем, что все-таки, если на изображениях будут дефекты, лишние объекты, или же палочки будут слишком близки друг к другу, то это может привести к неправильной классификации и новому подбору гиперпараметров (например, размер ядра для эрозии) для решения задачи.