Лабораторная работа №2 по урсу "Обработка и распознавание изображений"

Голдобина Ольга, 317 гр.

уроветь Advanced

Постановка задачи

Данная лабораторная работа направлена на изучение методов классификации формы изображений.

Необходимо было реализовать программу для классификации изображений моделей графов.

В задании уровня Advanced нужно было научиться классифицировать графы на пёстром фоне.

Описание данных

Данные представляют собой набор изображений моделей графов из деталей магнитной головоломки на белом или пёстром фоне. Формат изображений - JPG. Имеется 4 класса графов. В наборе присутствует по одному эталлонному изображению каждого класса: 2.jpg (класс I), 3.jpg (класс II), 4.jpg (класс III), 5.jpg (класс IV). Также прилагается файл с разметкой классов всех входных изображений.

Описание метода решения

Реализованная программа позволяет классифицировать графы как на белом, так и на пёстром фоне.

Для построения признакового описания графов предлагается определить, вершины каких степеней присутствуют в графе, и для каждой степени подсчитать количество таких вершин. Вершины степени 2 не учитываются (это связано с тем, что функции из используемой библиотеки не всегда коректно распознают такие вершины). Подсчета количества степеней 1,3,4,5 хватает для того, чтобы точно определить класс графа.

Выделение признаков осуществляется в несколько этапов:

1. выделение графа на изображении
2. построение скелета
3. построение графа по скелету с помощью библиотеки sknw.
4. подсчет числа вершин различных степеней.

Выделение фигуры графа на изображении отличается для белого и цветного фона.

Чтобы выделить граф на белом фоне достаточно бинаризовать изображение, применить дилатацию и удалить лишние мелкие объекты.

Для выделения графа на пёстром фоне бинаризация не производится. Вместо этого опытным путём для каждого из трёх каналов изображения были подобраны пороги, которым наиболее вероятно удовлетворяет цвет графа. Таким образом строится маска, выделяющая пиксели, принадлежащие графу. Чтобы объединить отдельные пиксели в цельную фигуру применияются опреации дилатации и закрытия. Далее удаляются лишние мелкие объекты. Иногда из-за недостаточного количества выделенных точек фигура получалась разрывной. Для устранения мелких разрывов несколько раз в цикле применялись операции дилатации и скелетенизации. Это позволяет удленнить отростки графа и соединить их, при этом не испортив изначальную структуру графа. Стоит отметить, что данная процедура замедляет работу программы, но повышает качество построения фигуры.

Построение скелета осуществляется с помощью skeletonize из skimage.

Построение графа осуществляется с помощью библиотеки sknw. Построенный граф нельзя сразу использовать для подсчета вершин. Это связано с тем, что у скелета присутствуют мелкие отростки, которые распознаются как рёбра. Такие рёбра необходимо удалить. Кроме того, в скелете имеются короткие звенья, которые должны распознаваться как точка, но в графе отражены как две близко расположенные вершины. Такие вершины нужно склеить.

После одготовки графа производится **подсчет вершин всех степеней** средствами той же библиотеки.

Для классификации изображений заранее составлены признаковые описания эталонных графов. Они выглядят так:

класс 1: {1: 3, 3: 3, 4: 3},

класс 2: {1: 4, 3: 4, 4: 1},

класс 3: {1: 4, 3: 5, 4: 2, 5: 1},

класс 4: {1: 6, 3: 4, 4: 2}.

Для сравнения входного графа с эталонными используется L1-норма, то есть сумма модулей отклонений числа вершин определенной степени от их числа в эталонном графе.

В качестве класса выдается тот, расстояние до эталона которого минимально.

Описание программной реализации

Реализованная программа написана на Python 3. Использовались такие библиотеки, как matplotlib, skimage, collections, networkx и sknw.

Программа разделена на функции, которые содержатся в модуле "graph_classification.py".

Для работы с изображениями применялась только библиотека skimage. Далее будут указываться только названия используемых функций.

Для выделения фигуры графа реализованы функции get_figure и get_figure_colored. Функция get_figure принимает на вход чёрно-белое изображение. В ней производится бинаризация изображения с помощью threshold_local. Далее удаляются мелкие объекты с помощью remove_small_objects, применяется функция dilation и очщается граница с помощью clear_border.

Функции get_figure_colored на вход подается цветное изображение. По заданным порогам выделяются пиксели фигуры графа. С помощью remove_small_objects удаляются лишние пиксели, попавшие в маску. Применяются функции dilation и closing. Далее с помощью label выделяются все связные компоненты маски, и снова применяя remove_small_objects, удаляются все, кроме компоненты наибольшей площади (это предоложительно фигура графа). Далее в цикле трижды применяются функции skeletonize и dilation для устранения разрывов фигуры.

Функция draw_graph рисует построенный граф поверх входного изображения. Она нужна для визуализации работы алгоритма.

Функция get_features осуществляет выделение признаков из изображения.По номеру изображения она считывает нужный файл. Далее осуществляются следующие действия:

1. в зависимости от того, какой фон на изображении, применяется функция get_figure или get_figure_colored.
2. строится скелет с помощью функции skeletonize.
3. производится построение графа по скелету с помощью функции build_sknw из библиотеки snkw. Она возвращает объект класса networkx.MultiGraph.
4. Производится удаление коротких рёбер с помощью метода remove_edge и склеивание соответствующих вершин с помощью метода contracted_nodes. Ребро считается коротким, если его длина меньше 47 пикселей.
5. Выполняется подсчёт количества вершин различных степеней. Степени вершин можно получить с помощью метода edges. Подсчет осуществлятся с помощью Counter из collections.

В функции distance реаизовано вычисление L1-нормы для признаковых описаний графов.

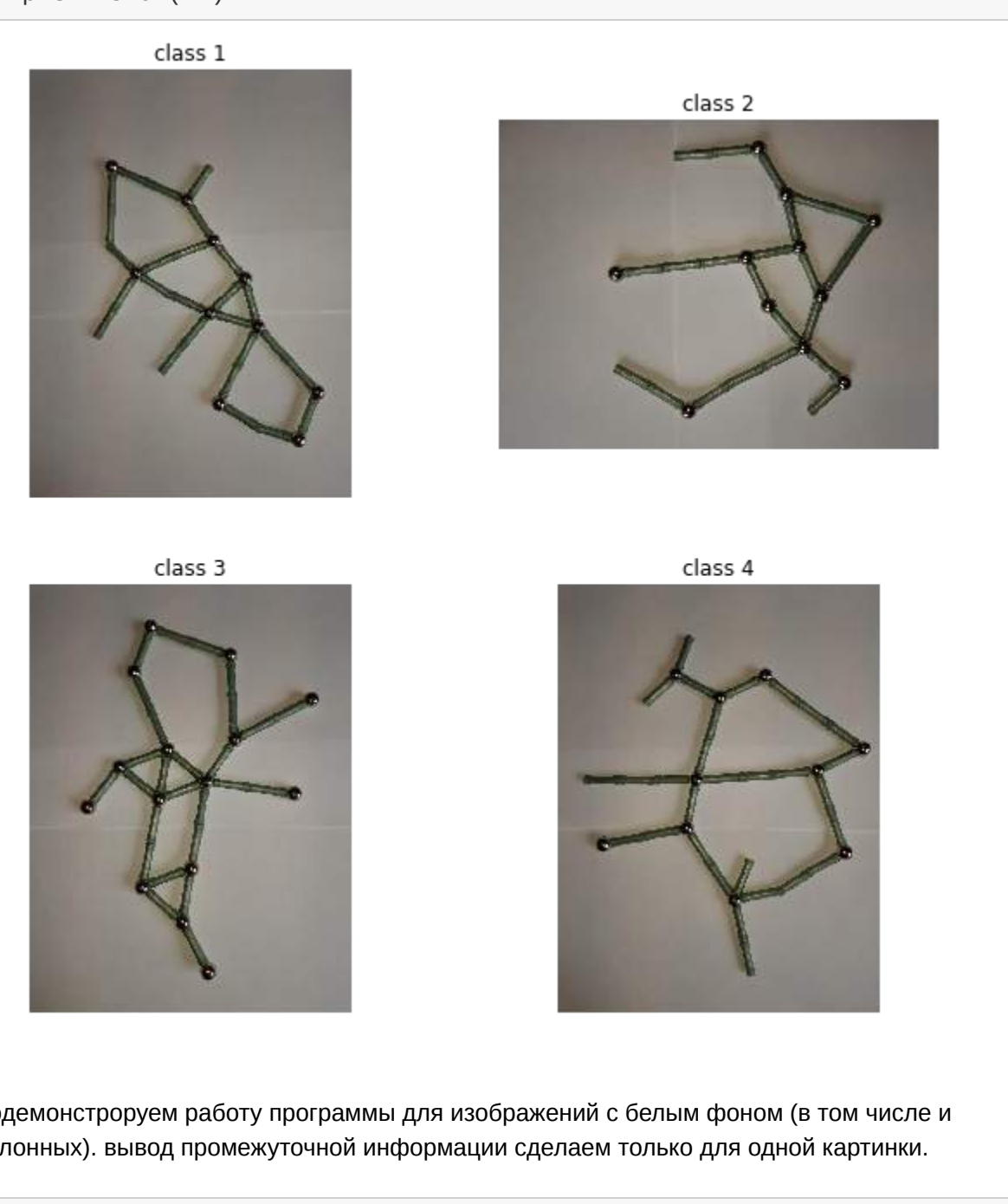
В функции classify производится сравнение полученного на вход признакового описания графа с эталонными с помощью функции distanse. графу присваивается класс наиболее близкого по заданной метрике эталона.

В функциях get_figure, get_figure_colored, get features имеется параметр verbose, который может принимать значения 0, 1, 2. Он отвечает за количество выводимой информации о промежуточных преобразованиях изображений.

Эксперименты

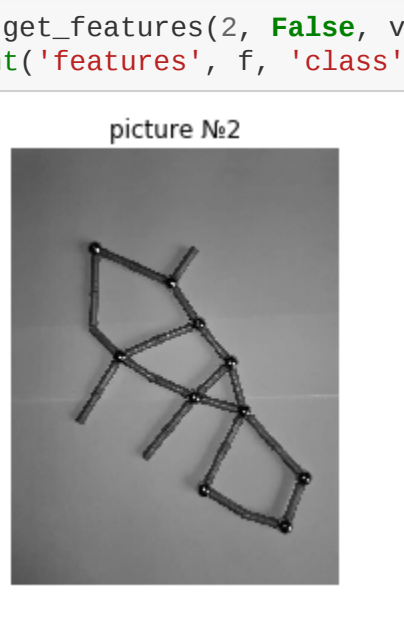
выведем эталонные графы.

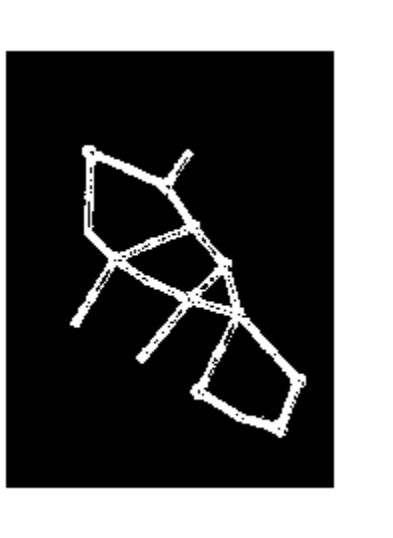
In [244]:
```
plt.figure(figsize=(10,10))
for i in [2,3,4,5]:
    im = read(i)
    plt.subplot(2,2,i-1)
    plt.axis('off')
    plt.title('class {}'.format(i-1))
    plt.imshow(im)
```

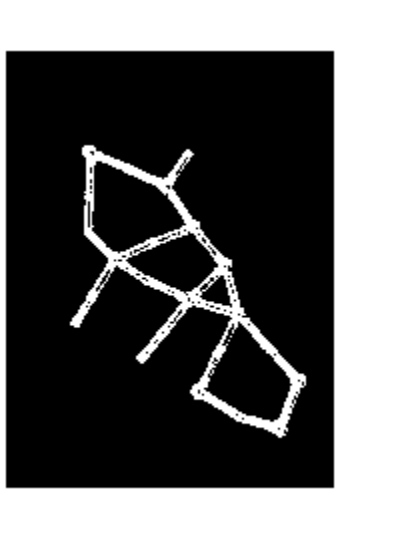


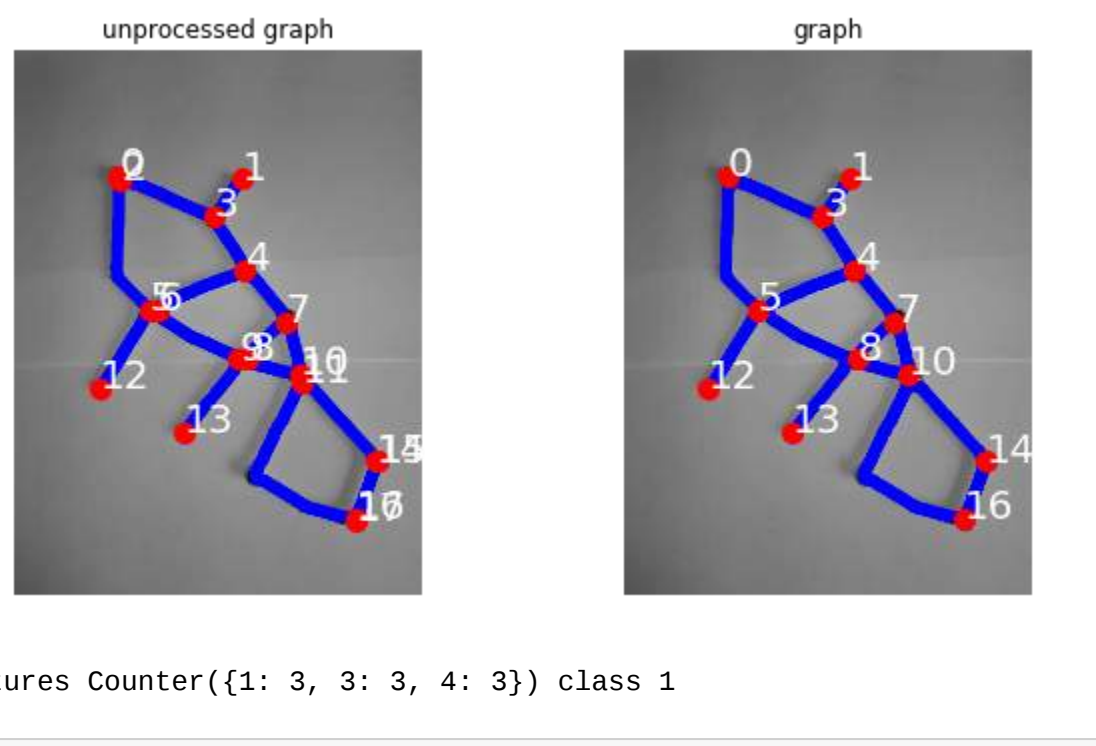
продемонстрируем работу программы для изображений с белым фоном (в том числе и эталонных). вывод промежуточной информации сделаем только для одной картинки.

In [258]:
```
f = get_features(2, False, verbose=2)
print('features', f, 'class', classify(f))
```





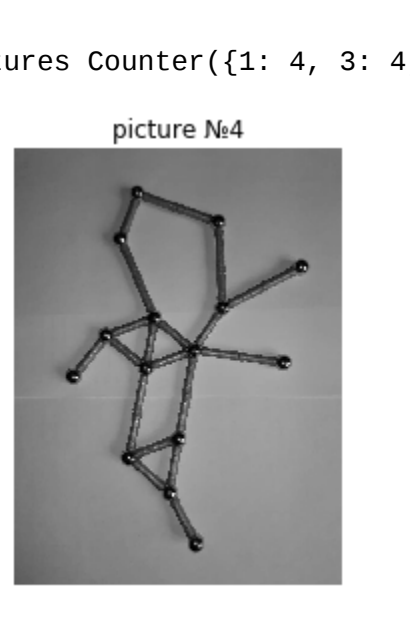




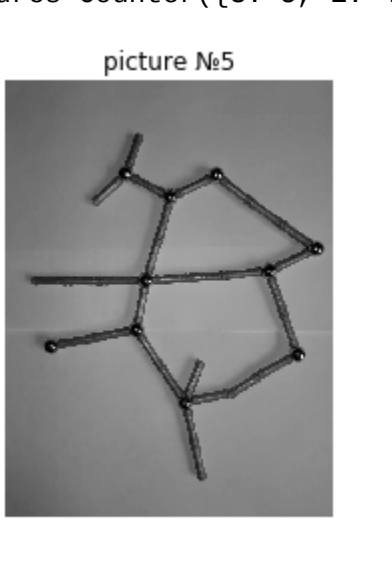

```
features Counter({1: 3, 3: 3, 4: 3}) class 1
```

In [249]:
```
for i in [3, 4, 5, 7, 13, 19, 25]:
    f = get_features(i, False, verbose=0)
    print('features',f, 'class', classify(f))
```


```
features Counter({1: 4, 3: 4, 4: 1}) class 2
```

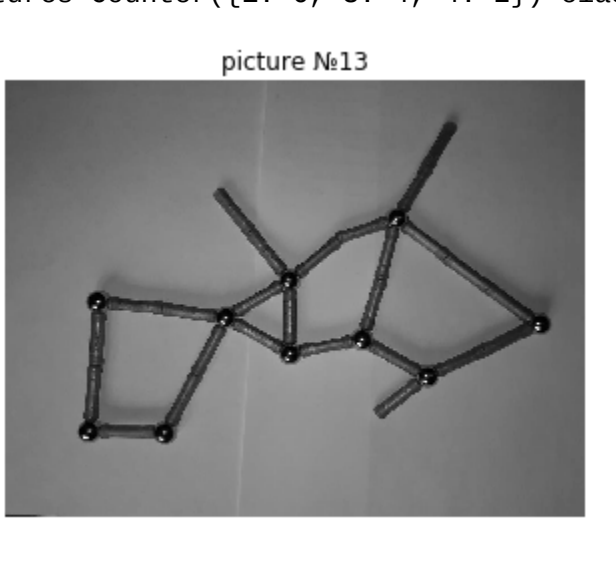



```
features Counter({3: 5, 1: 4, 4: 2, 5: 1}) class 3
```

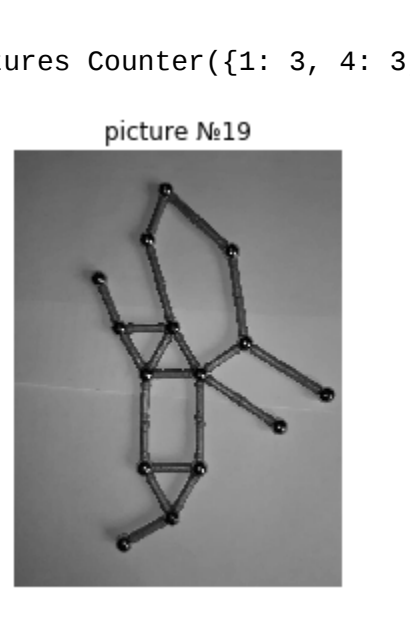



```
features Counter({1: 6, 3: 4, 4: 2}) class 4
```


```
features Counter({1: 6, 3: 4, 4: 2}) class 4
```




```
features Counter({1: 3, 4: 3, 3: 3}) class 1
```

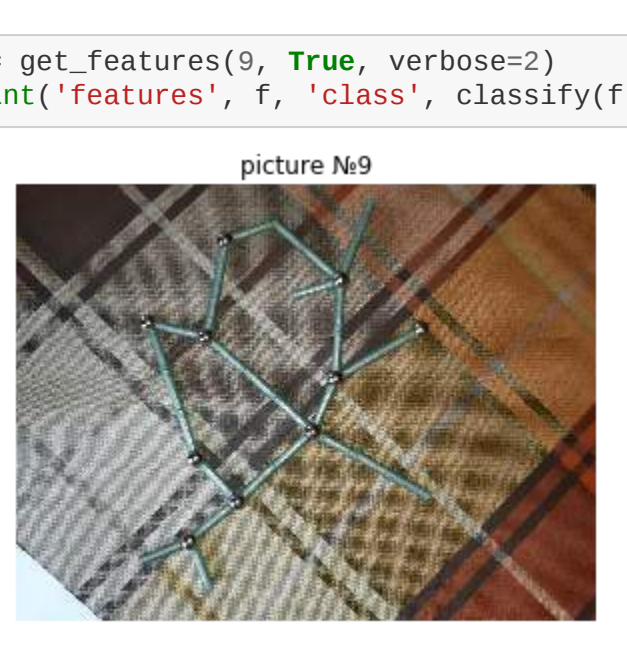



```
features Counter({3: 5, 1: 4, 4: 2, 5: 1}) class 3
```

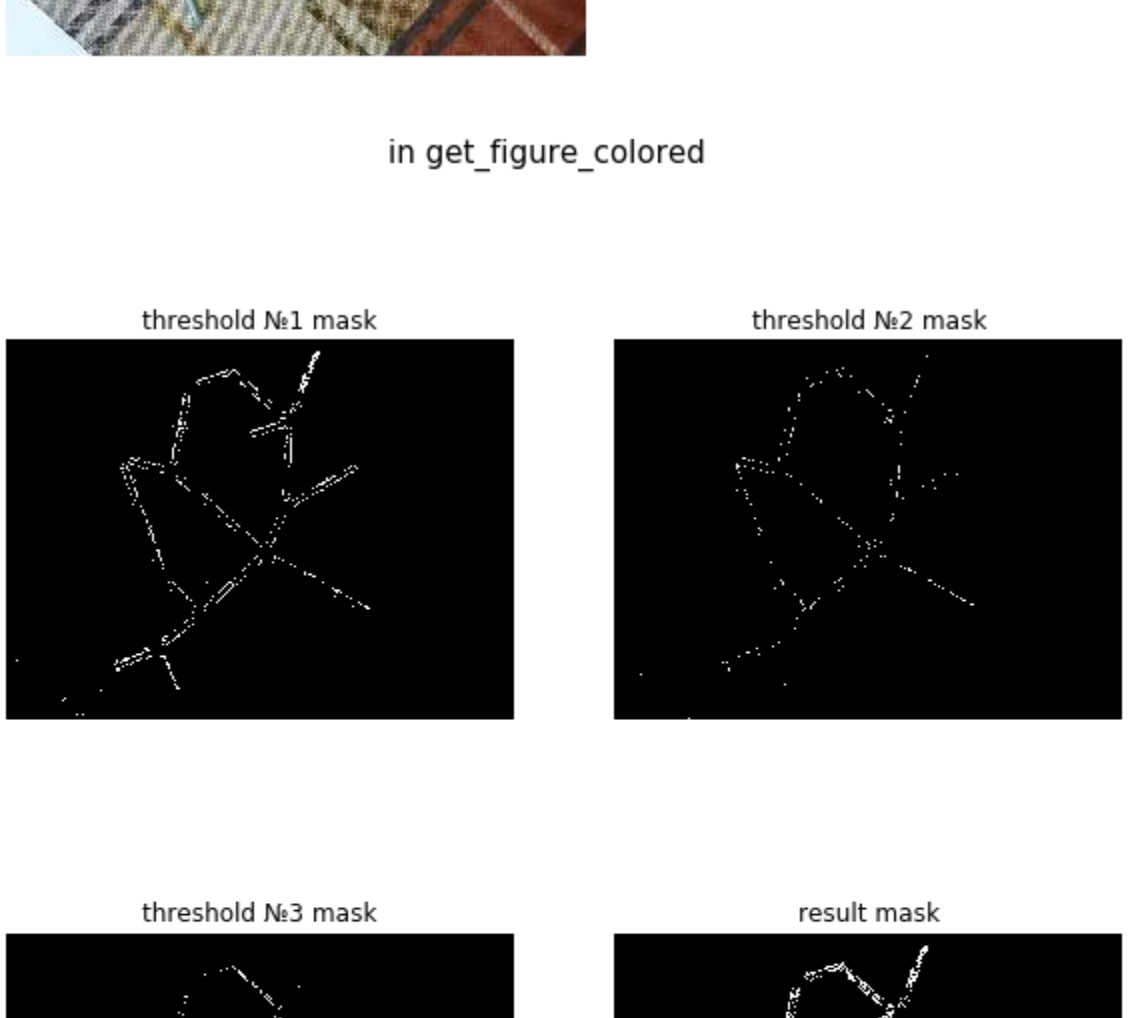

```
features Counter({1: 4, 3: 4, 4: 1}) class 2
```

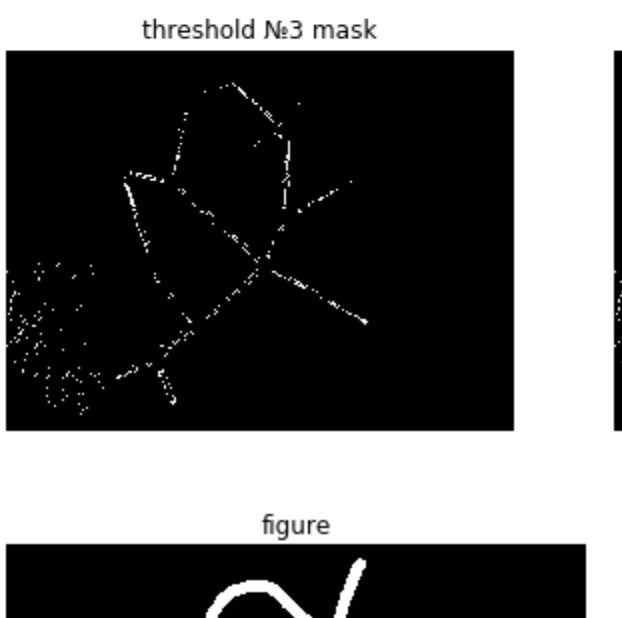
продемонструем работу программы для изображений с пёстрым фоном. вывод промежуточной информации сделаем только для одной картинки.

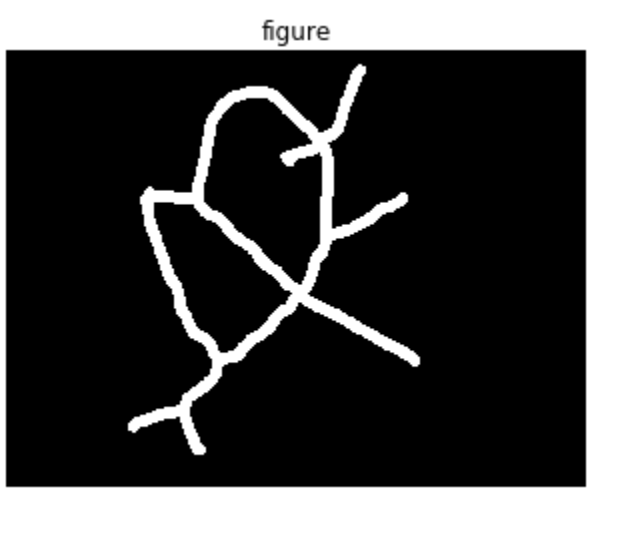
In [255]:
```
f = get_features(9, True, verbose=2)
print('features', f, 'class', classify(f))
```

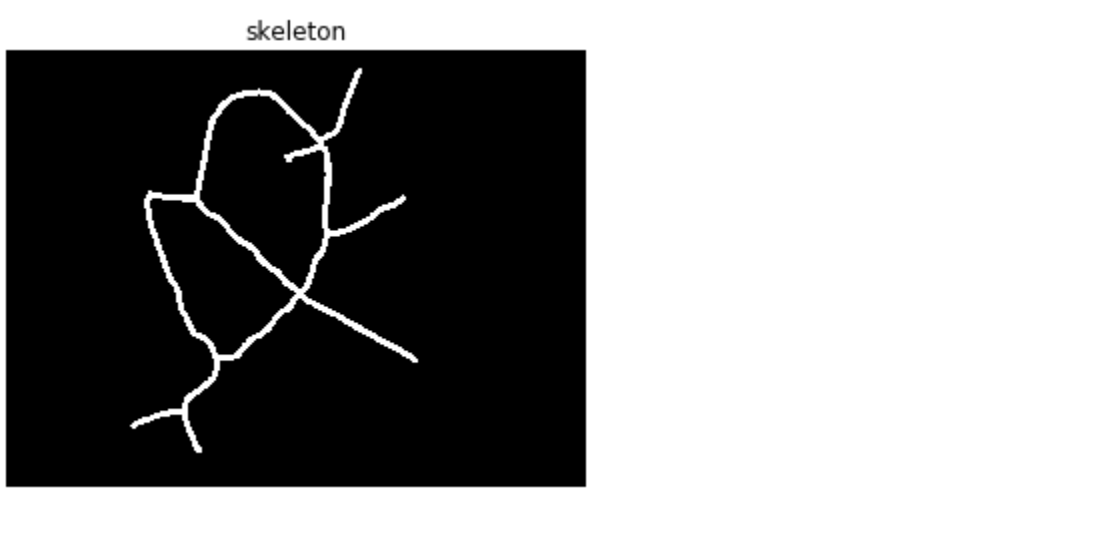


in get_figure_colored



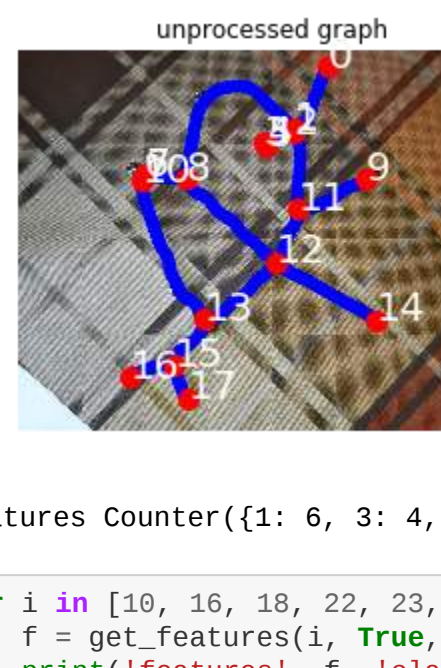




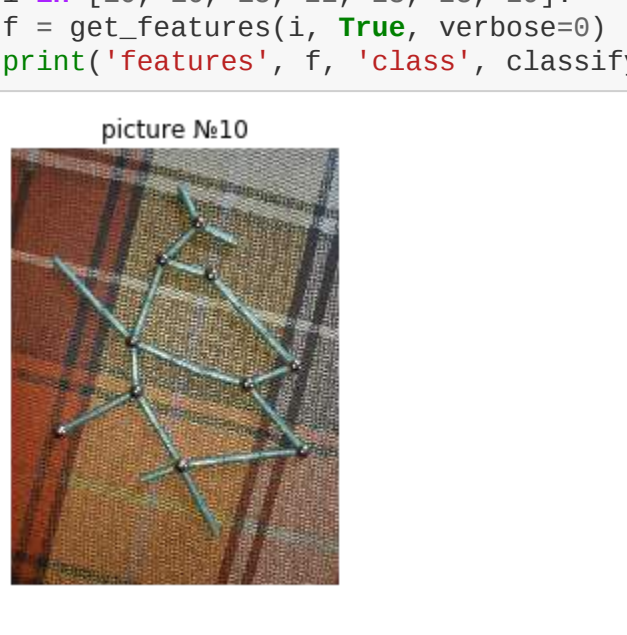



```
features Counter({1: 6, 3: 4, 4: 2}) class 4
```

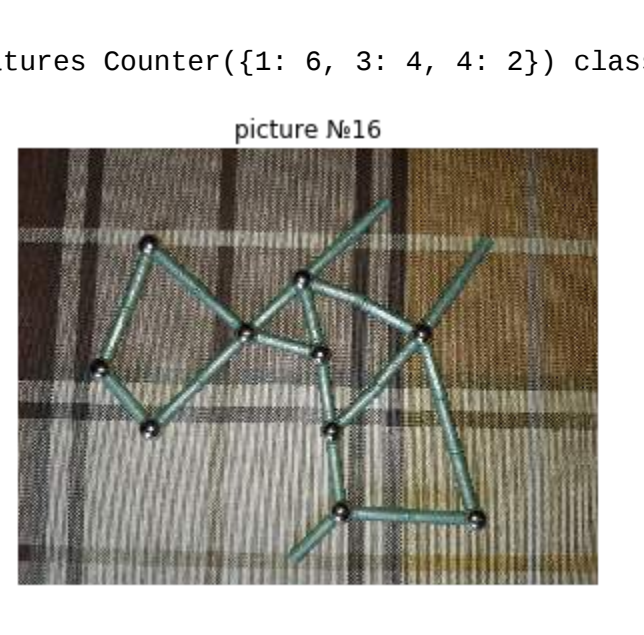
In [257]:
```
for i in [10, 16, 18, 22, 23, 28, 29]:
    f = get_features(i, True, verbose=0)
    print('features', f, 'class', classify(f))
```



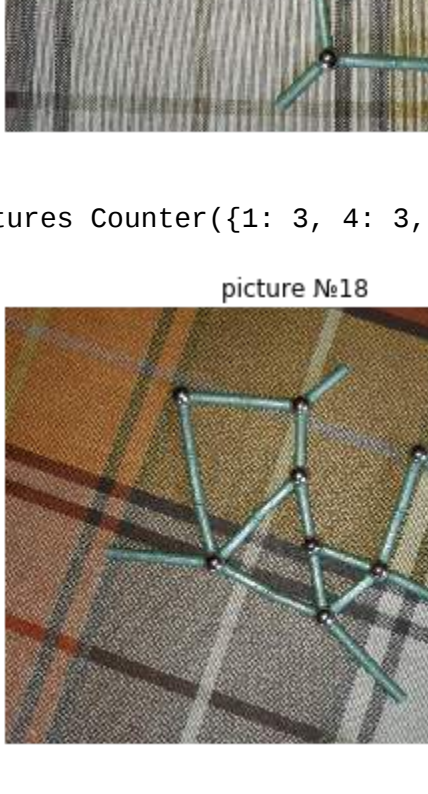

```
features Counter({1: 6, 3: 4, 4: 2}) class 4
```



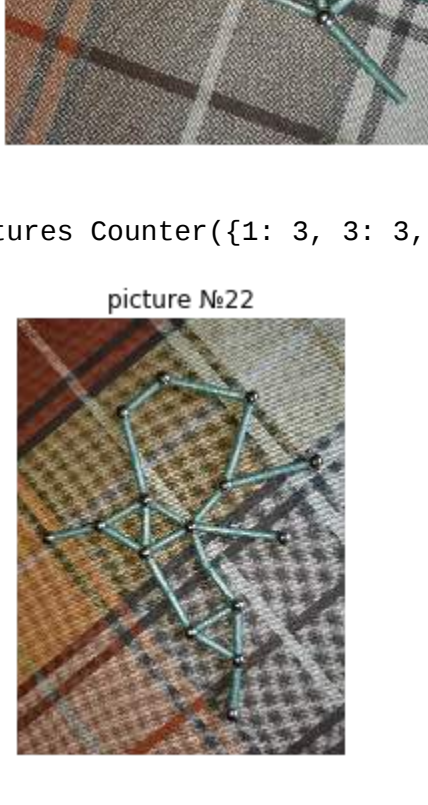

```
features Counter({1: 3, 4: 3, 3: 3}) class 1
```



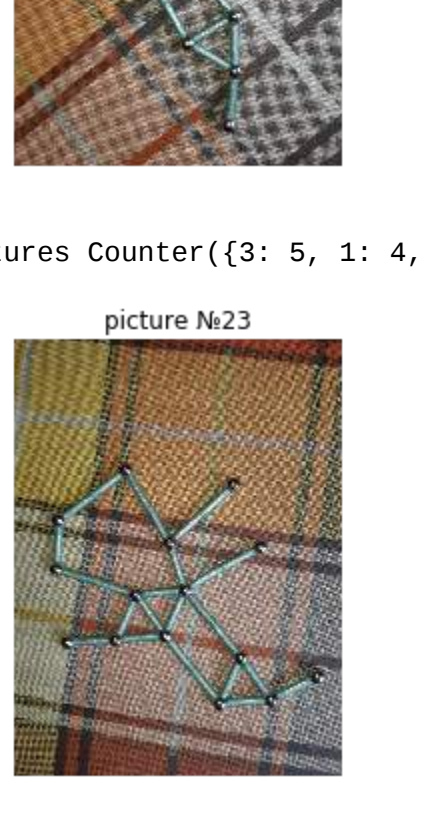

```
features Counter({1: 3, 3: 3, 4: 3}) class 1
```




```
features Counter({3: 5, 1: 4, 4: 2, 5: 1}) class 3
```




```
features Counter({3: 5, 1: 4, 4: 2, 5: 1}) class 3
```




```
features Counter({1: 4, 3: 4, 4: 1}) class 2
```


```
features Counter({1: 4, 3: 4, 4: 1}) class 2
```

видим, что графы классифицируются правильно.

Выводы

Таким образом, были изучены методы классификации формы изображений и реализована программа для классификации графов. Как показали эксперименты, удалось достичь высокого качества распознавания.