Отчет по Заданию Лаб_1-2021 по курсу "Обработка и распознавание изображений" кафедры ММП ВМК

Охотин Андрей, 317 группа, ММП

06.04.2021

Постановка задачи

Разработать и реализовать программу для работы с изображениями фишек игрового набора Тримино. Программа должна обеспечить;

1. ввод и отображение на экране изображений в формате BMP;
2. сегментацию изображений на основе точечных и пространственных преобразований;
3. поиск фишек на картинках;
4. классификацию фишек на картинках.

Описание данных

Задача решена на уровне Intermediate: фишки расположены на синем фоне с неоднородным освещением. Основную сложность здесь представляют два пункта:

1. Из-за освещения, цвета материала, из которого сделаны тримино, и их маркировка могут значительно искажаться, что очевидно, мешает их распознаванию.
2. Фишки могут располагаться достаточно плотно, из-за чего их разделение может представлять проблему. Например, когда две фишки расположены так, что две их стороны находятся близко и параллельно друг другу.

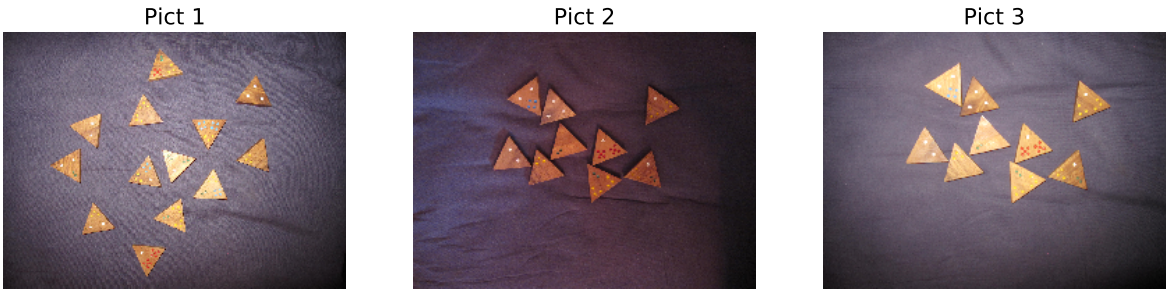
Описание метода решения. Описание программной реализации. Эксперименты.

В этой секции совмещены описанные разделы, чтобы не делать отсылки между ними, из-за чего придется часто переходить от одного к другому, и алгоритм будет непонятен. Приведенные изображения, это эксперимент, проведенный на выданных изображениях. На них поэтапно видно, какая часть алгоритма решения приводила к какому результату. Здесь же будет указано, что выполнялют те или иные функции в коде.

Решение строилось на использовании кластеризаций, сверток и непосредственной работой с определенными цветами. Надо уточнить, что под сверткой здесь понимается так называемая операция **развертки** без изменения размерности (с обрезкой краев) scipy.signal.convolve2d. Решение имеет следующие этапы:

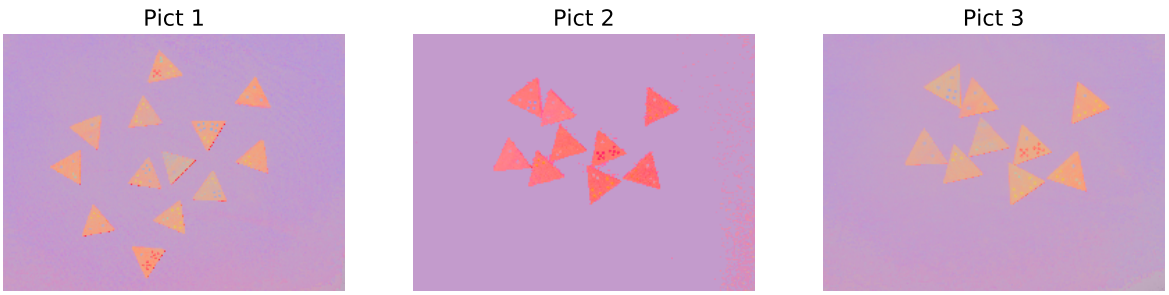
1. Предобработка изображения.

Пример необработанных изображений



В начале идет отделение цветов маркировки, чтобы в последующей обработке они не были приведены к общей цветовой гамме. Таким образом обрабатывались все цвета, кроме древесного цвета фишек, цвета фона и красной маркировки (красный из-за освещения и материала фишек детектировать сложнее) <1>. Далее, для выравнивания освещения происходила нормировка RGB каналов каждого пикселя и приведение среднего каждого канала по картинке к одному значению <2>. После шла финальная обработка, состоящая в отделении красного цвета маркировки (это происходило с учетом средних значение каналов по картинке) <3>.

Пример предобработанных изображений



Код этого пункта находится в файле data_preprocess. Алгоритм этого пункта выполняет функция normalized_img. Она содержит в себе функции transform_hard_colors <1>, norm_img <2> и color_transform <3>.

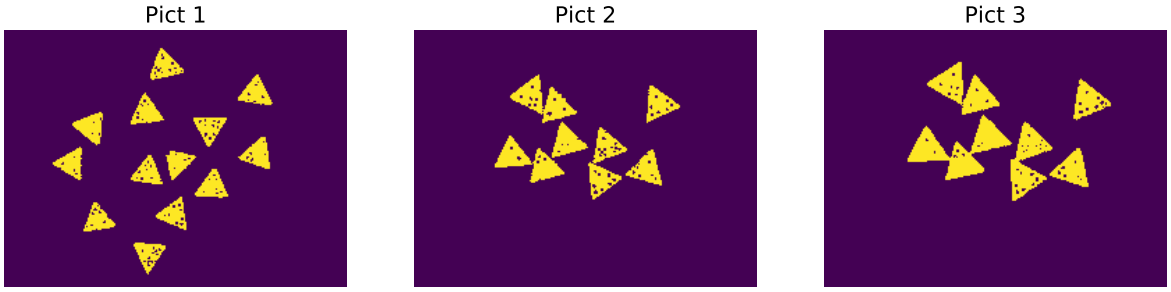
2. Детекция фишек тримино.

   (a) Определение цвета фишек <5>. Использовалась смесь из двух Гауссиан и алгоритм k-means. Помимо значений каналов для каждого пикселя были добавлены новые признаки из комбинаций значений каналов <4>. Цвет определялся кластером, в котором

находилось второе по величине количество объектов. Так как эти алгоритмы не обеспечивают целостности участков из пикселей, принадлежащих одному кластеру, далее было применено две свертки. Таким образом удалось избавится от большей части шума. Далее предсказания алгоритмов были усреднены, чтобы избавится от остатков шума. В итоге была получена **маска** (под маской понимается матрица размером как исходное изображение, состоящая из 0 и 1), в которой были 1 на позициях пикселей, которые относятся к фишкам тримино.
Код этого подпункта находится в файле image_analyzer в классе Data в методе average_alg <5>. Добавление фичей происходит в методе add_magical_features <4>.

(b) Составление маски фишек для изображения <6>. Следующим этапом идет уточнение границ фишек. Для этого были взяты по маске, полученной в прошлом пункте пиксели, относящиеся к фишкам. Для них была применена смесь из двух Гауссиан, как в прошом пункте, но на данном этапе, мы получим очень плотный кластер из пикселей, цвета материала, из которого сделаны фишки тримино. Остальные цвета на этапе предобработки были приведены к одному и тому же цвету, а потому отнесены в другой кластер. Эта процедура так же необходима для очистки от шума в виде других цветов, которые мы не смогли отделить на этапе предобработки. Для этого хорошо подходит модель смеси Гауссиан. Мы получаем одну Гауссиану, которая имеет очень маленькое стандартное отклонение и сосредочена на цвете тримино. Вторая же Гауссиана, имеет гораздо большее стандартное отклонение и вбирает в себя все объекты остальных цветов. При этом для проведения такой процедуры необходимо отделение фона, как в предыдущем пункте, так как он является слишком сильным источником шума. В итоге мы получаем маску, точно очерчивающую границы тримино, а так же выделяющую маркировку. Далее будем работать только с этой маской.

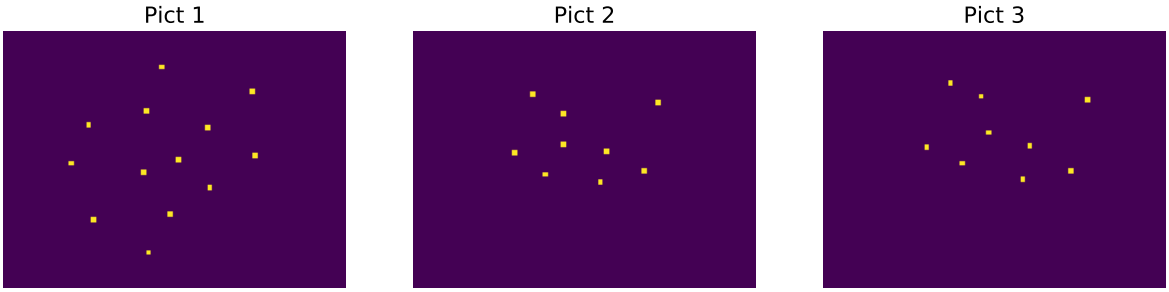
Пример итоговых масок для изображений



Код этого подпункта находится в файле image_analyzer в классе Data в методе trimino_material <6>.

(c) Количество фишек <7>. Теперь по этой маске можно определить количество фишек. Это делалось с помощью скользящего по маске окна, в котором центральному элементу присваивалось минимальное из всех элементов попавших в это окно. Далее было применено несколько сверток, чтобы сгладить границы областей. Таким образом наложение друг на друга масок фишек, расположенных рядом было нивелировано. В итоге от фишек тримино на маске остаются лишь небольшие непересекающиеся округлые фигуры, которые нетрудно посчитать.

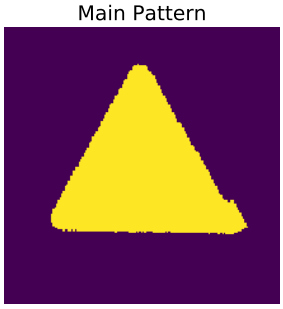
Пример найденных центров



Код этого подпункта находится в файле image_analyzer в функции num_triangles <7>.

(d) Выделение блоков <8>. Теперь необходимо получить блоки, к которых будет ровна одна фишка тримино по центру. Чтобы найти каждую отдельную фишку тримино использовалась метрика следующего вида. Метрика <9> определяет похожесть фигуры, находящейся внутри определенной области на треугольник, с центром в центре этой области. Для подсчета использовалась маска полученная в подпункте (b). Брался образец фишки тримино (маска с 1 в виде треугольника) под всеми возможными углами поворота.

Образец для метрики



Значение метрики для каждого блока (области маски из 0 и 1) считалось, как количество попавших внутрь образца единиц, поделенное на количество единиц в образце. Таким образом получилось избежать помех из-за находящихся рядом других фишек. Значение метрики бралось как максимальное по всевозможным вращениям образца. В итоге мы имеем значение такой метрики, посчитанное для всех блоков этой маски такого размера. Из предыдущего пункта мы знаем N количество фишек. Остается взять N наиболее похожих блоков. Так же, теперь мы можем посчитать центр фишки. Домножив поэлементно образец под оптимальным углом (мы его знаем из этапа подсчета метрики), мы получим отмаскированную фишку. Посчитав среднее для номеров строк и столбов единиц в этой отмаскированной области, мы получим центр фишки.

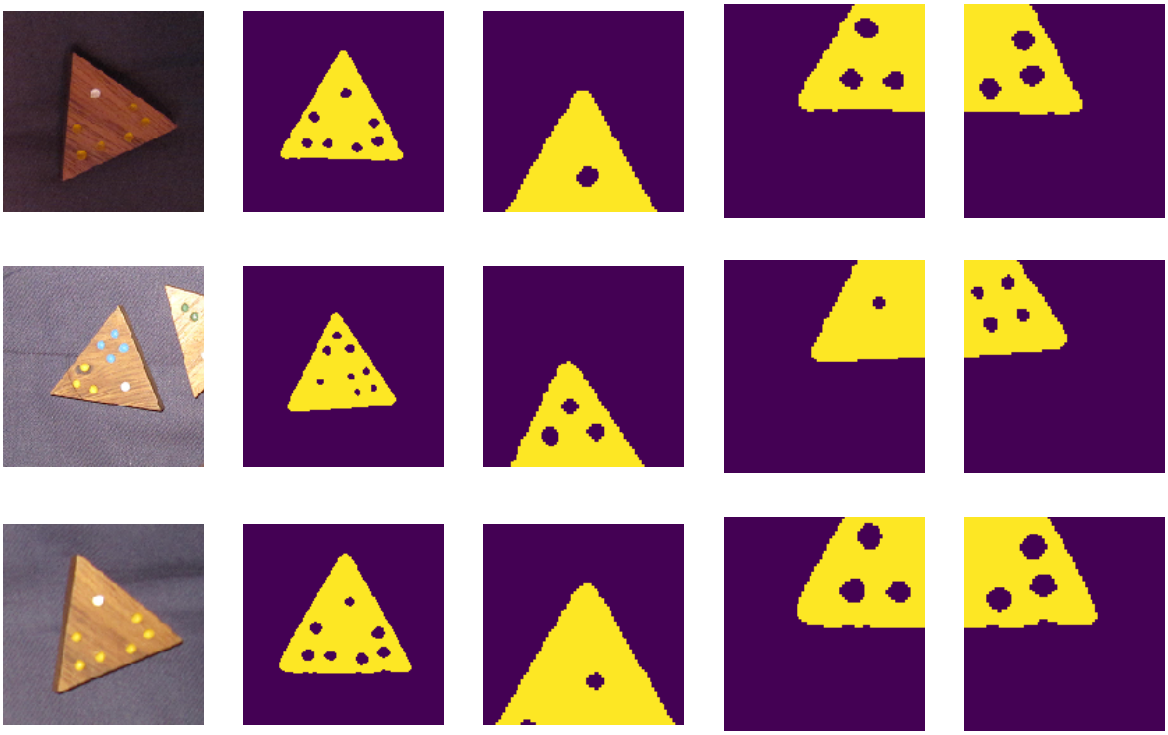
Код метрики находится в файле image_analyzer в классе TrianglesMetric <9>. Подсчет метрики и выделение N блоков в соответствии с результатом функции num_triangles происходит в классе Triangles в конструкторе __init__ <8>.

Реализация этого пунтка инкапсулирована в класс ImageReader и находится в файле image_analyzer. В него передается нормализованное изображение из предыдущего пунк-

та. В нем, в конструкторе последовательно вызываются описанные функции и методы. А далее в методе find_triangles этого класса происходит поиск блоков с фишками тримино.

3. Считывание блоков. Нам надо привести блоки, к одной и той же ориентации. В нашем случае чтобы одна из сторон фишки была параллельна горизонтальной гране блока. Из последнего подпункта предудыщего пункта, мы знаем положение блоков и то на какой угол повернуты, находящиеся там фишки тримино. Теперь поворачиваем блоки в соответствии с этими углами и получаем одинаково ориентированные фишки. Далее для каждого блока использовалось увеличение областей из 0, чтобы точнее определить маркировку, а так же две свертки, для избавления от шума. Теперь, так как имеем одинаково центрированные и ориентированные блоки, углы фишек располагаются одинаково. Мы разделяем блок на три части (верхний, левый и правый угол, как в изображениях) и считаем маркироку в каждом углу <10> (количество областей из 0, внутри области из 1. напомню, что блоки - это маски из 0 и 1) <11>.

Образцы найденных блоков



Реализация считывания блоков инкапсулирована в класс BlocksReader и находится в файле blocks_reader. Конструктор __init__ в этом классе выполняет алгоритм <11>. Код определения количества точек (маркировки того или иного угла фишки тримино) находится в этом же файле в функции read_points.

Программная реализация

Код программы представлен в файле detector, который может быть запущен при выполнении условий указанных в README. В нем высокоуровневыми командами выполняется описанный

алгоритм. Время работы алгоритма приблизительно равно 3-4 минутам для одного изображения. Все операции с изображениями в коде выполнены достаточно низкоуровнево, без использования таких мощных библиотек как OpenCV. Описанные три пункта находятся в функции normalized_img и классах ImageReader и BlocksReader. Форматированный вывод происходит в файл predictions.txt в папке output.

Эксперименты

Проводить эксперименты удобно в Jupyter-Notebook в файле experiments.ipynb, предварительно поместив в папку **данные** изображения. В нем уже есть пример кода, для тестирования алгоритма. По итогам экспериментов на предоставленных изображениях, описанный алгоритм решает задачу детектирования положения центров фишек тримино на заявленном уровне с высокой точностью: отклонение по каждой координате не превышает 30 пикселей. Точность маркировки определялась не по целой фишке, а по тому, как алгоритм распознавал маркировку на каждом углу фишки. Таким образом можно точнее определить, где алгоритм правильно распознал. На предоставленных изображениях была получена точность определения маркировки: Pict_2_1.bmp - 91.5%; Pict_2_2.bmp - 81.4%; Pict_2_3.bmp - 85.1%.

Выводы

Алгоритм инвариантен к смещению, поэтому при выделении той или иной части картинки, точность не падает. Основную проблему может представлять освещение. Изображения Pict_2_2.bmp и Pict_2_3.bmp отличаются только освещением, и точность алгоритма на них не сильно отличается. При такой задаче точного распознавания очень сильно влияет первоначальная предобработка, цель которой, сделать алгоритм инвариантным к освещению. Далее эффективно могут работать методы кластерации, для определения цветов тех или иных объектов, а так же операция свертки для понижения шума.