Московский Государственный Университет имени М.В. Ломоносова

Факультет Вычислительной Математики и Кибернетики

Кафедра Математических Методов Прогнозирования

Курс «Обработка и распознавание изображений»

Отчёт о выполнении 1 лабораторной работы

Самбурский Александр
317 группа

Москва 2021

1. Введение

Одну из центральных позиций в машинном обучении занимает обработка изображений. Это направление включает в себя множество интересных и, в то же время, нетривиальных подзадач, таких как выделение контуров, сегментация, преобразования изображений, и многие другие. Для освоения навыков решения таких задач важна как теоретическая основа, предлагающая сильные инструменты для эффективной реализации алгоритмов, так и изучение свойств этих алгоритмов на практике. Данная работа направлена на изучение приёмов обработки изображений в задаче сегментации.

Сама задача заключается в распознавании фигур – фишек треугольной формы – на фотографиях. В каждом углу фишки изображены точки, соответствующие числу очков (пример изображён на рис.1).

Рис.1 Целевые объекты для детектирования

Задача ставится следующим образом: применить методы обработки изображений для:

1. Определения положения фишек – для этого достаточно предоставить приближённые координаты их центров;

2. Установления для каждой фишки соответствующее ей число очков.

Как видно, данные треугольные фишки имеют достаточно ровный контур, и при этом имеют примерно похожий «деревянный» оттенок. В связи с этим, открывается широкий спектр возможностей для решения данной задачи и отработки используемых алгоритмов. В качестве датасета для отладки выбран уровень “Intermediate”.

В данной работе было решено сделать основной упор на цветовую обработку. Данный подход позволил изучить некоторые интересные закономерности методов, о которых будет изложено ниже. Несмотря на этот выбор направления решения, используемый алгоритм не ограничивается тривиальным поиском подходящих по цвету пикселей, а на разных этапах задействует различные идеи и механики. Во время выполнения работы было отработано несколько методов обработки изображений, и некоторые хорошие результаты не были задействованы в финальном алгоритме, однако о них так же будет упомянуто. Перейдём к изложению алгоритма.

2. Метод решения задачи

Весь процесс решения естественным образом разбивается на отдельные этапы.

1. Производится поиск и выделение фишек на картинке. Целью этого этапа является формирование прямоугольников, объемлющих фишки. Естественно данное разбиение считается удачным, если каждой фишке удалось сопоставить огибающий её прямоугольник, центр которого находится близко к центру фишки.
2. В пределах каждого прямоугольника производится поиск точек на фишках. К сложности данного этапа добавляется возможность попадания в прямоугольник фрагмента другой фишки (если фишки были расположены очень близко друг к другу).
3. Подсчёт полученных значений и вывод результатов программы.

Опишем данные этапы конкретно.

2.1 Первый этап

В начале алгоритма ставится тривиальная задача: найти фишки по цветовому признаку. Для этого предварительно применяются две операции: адаптивное затемнение и снижение разрешения.

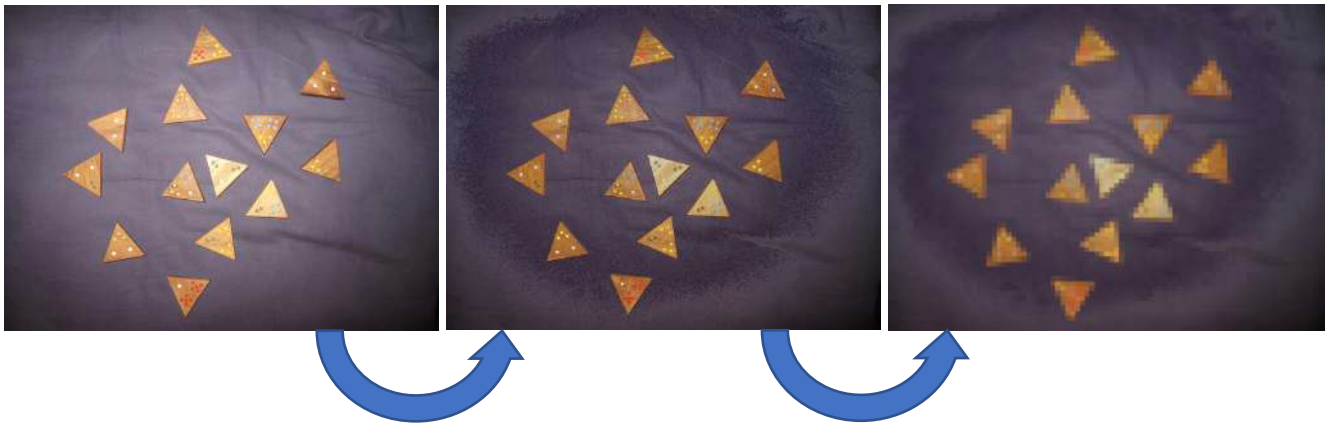


Рис.2 Работа адаптивного затемнения и сжатия

Оба преобразования значительно упрощают последующих поиск целевых пикселей. Первое снижает яркость чересчур ярких пикселей (уменьшая все RGB-значения на константную величину). Субъективный критерий яркости, применяющийся здесь – численная сумма значений трёх цветовых каналов пикселя. Чем она больше – тем пиксель визуально ближе к белому цвету и тем дальше от чёрного. Этим и объясняется выбор данного решающего правила. Затемнение позволяет сгладить различия между искомыми «деревянными» пикселями при неравномерной освещённости изображения.
Второе преобразование заключается в замене всех пикселей внутри двумерного шага заданной сетки изображения на их усреднённый цвет, а затем сжатии площади этого шага сетки в 1 пиксель. Например, в правом углу рисунка 2 изображено десятикратное сжатие (масштаб изображения увеличен). Это простое действие позволяет избавиться от маленьких группок пикселей, которые в данный момент не требуется рассматривать, и, кроме того, значительно снижает объём изображения. Таким образом, эффективность дальнейших алгоритмов повышается. В рассматриваемом примере данное удешевление стократно, и это заметно сказывается на скорости работы (и даже точности).

Потом производится поиск пикселей, чей цвет близок к цвету фишек. Данная операция может быть произведена двумя способами. Первый способ основан на простом использовании метрики в цветовом пространстве. Метрика порождается евклидовой нормой.
На рисунке изображён принцип работы поиска фишек: на изображении выделяются те пиксели, которые в своём векторном представлении находятся внутри сферы с центром в заданном цвете. Центральный цвет и радиус сферы задаются программистом. Как правило, данная инициализация допускает возможность некоторой погрешности без потери качества. Это свойство достигается благодаря прошлым преобразованиям.

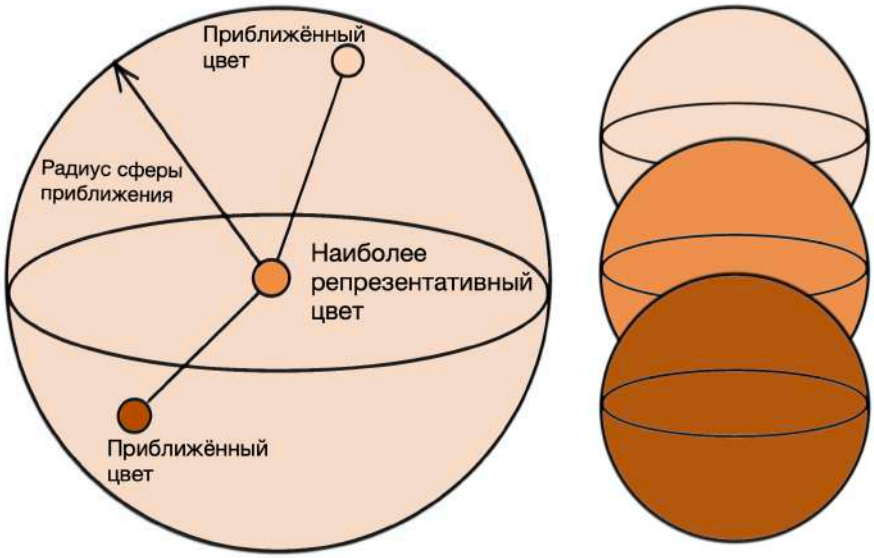


Рис.3 Элементарные решающие поверхности в цветовом пространстве

Данный подход с лёгкостью можно обобщить. Первое усложнение заключается в том, что можно использовать не одну сферу. Данный подход позволяет очень точно описать целевой цветовой оттенок, и не требует существенного увеличения времени работы алгоритма в условиях векторных вычислений.

Второй метод иначе эксплуатирует метрику в цветовом пространстве. Для определения сходства пикселя с искомым цветом всё пространство делится гиперплоскостями. Каждая гиперплоскость задаётся уравнением, связывающем цвета одного пикселя в одну линейную комбинацию и задающем её значение. Этот подход обеспечивает возможность задания взаимных условий на значения цветовых каналов. Например, если необходимо найти деревянный цвет, подойдёт условие превалирования красного канала над некоторой выпуклой линейной комбинацией зелёного и синего. При этом данный метод остаётся независимым относительно яркости пикселей, но требует немного больше ограничений, чем первый подход.

На рисунке справа изображено разбиения цветового пространства указанным способом. Три заданные гиперплоскости отделяют оранжевый фрагмент, который может использоваться как подходящий сегмент для сходства с нужным пикселем. В эти разбиения войдут пиксели с разной яркостью, но одним оттенком.

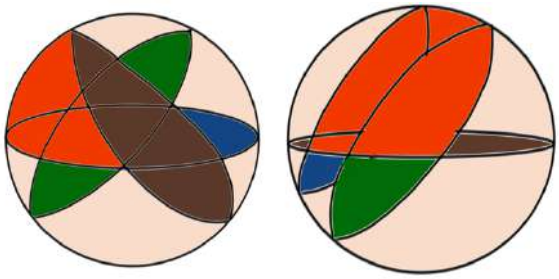


Рис.4 Разбиение цветового пространства гиперплоскостями

Данные методы хорошо справляются с задачей детектирования «деревянных» пикселей:

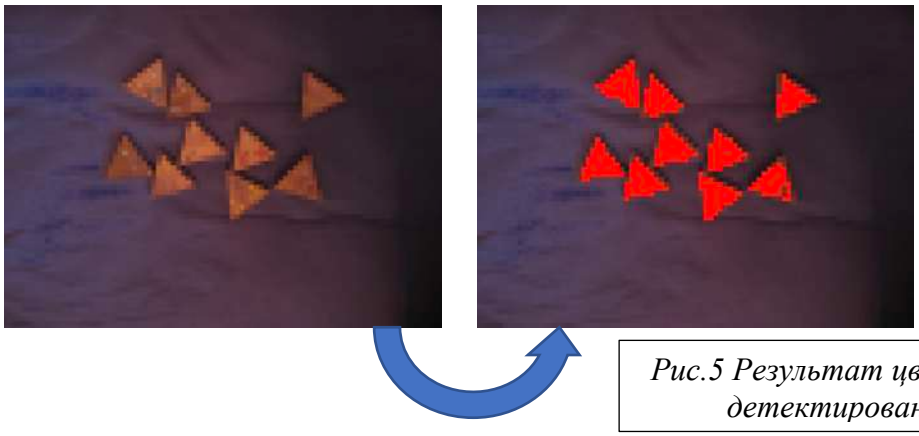


Рис.5 Результат цветового детектирования

После удачного детектирования остаётся выделить прямоугольники, содержащие группы помеченных пикселей. Считается, что пиксели находятся в одной группе, если между ними можно провести криволинейный ряд пикселей, так же содержащийся в этой группе. Координаты прямоугольников высчитываются из экстремальных координат каждой группы (минимум и максимум осевых номеров их пикселей) и множителя сжатия на прошлом этапе (прямоугольники выделяются на оригинальном изображении):

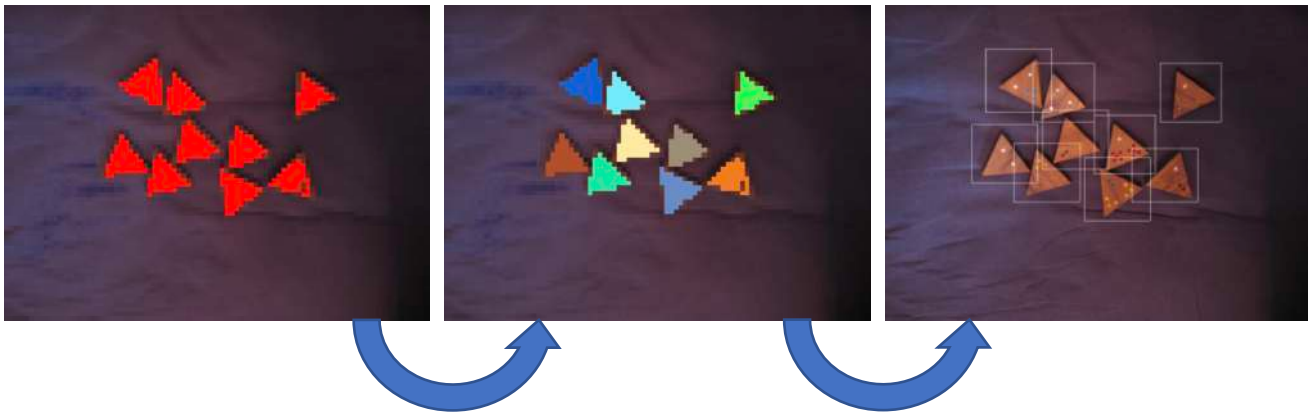


Рис.6 Результат приближённой сегментации

Теперь перейдём к изложению проблем данного метода разделения на группы.
Продемонстрированные выше результаты достаточно оптимистичные: все группы разбились корректно, и полученные прямоугольники взаимно однозначно соответствуют своим фишкам и весьма хорошо описывают их положение на картинке. Есть две типичные ситуации, которые данный наивный метод будет отрабатывать неправильно:

- Некоторые пиксели были размечены отдельно от основных групп – получаются индивидуальные группы маленьких размеров. Эти пиксели могут по смыслу относиться к ближайшей полноценной группе, либо по случайности оказаться и вовсе вдалеке от остальных групп. В обоих случаях наивный подход разбиения на группы выделит отдельный прямоугольник для этого множества, что недопустимо.

- Две и более фишек оказались в одной группе из-за их близости на оригинальном изображении. Здесь частично проявляется минус использования свёртки (сжатия) изображения. Если несколько фишек неконтролируемо попадут в один прямоугольник, то дальнейшее распознавание бессмысленно: как положение, так и число очков на фишках правильно вычислить не удастся. Ниже продемонстрирована эта ошибка.

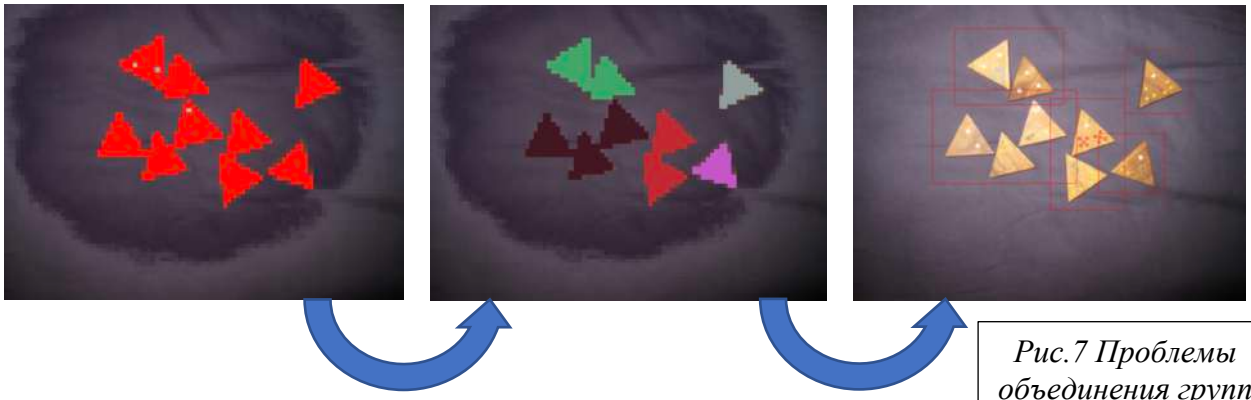


Рис.7 Проблемы объединения групп

Для преодоления данных проблем используется следующая стратегия.

- Любые группы, включающие малое число пикселей (ниже некоторого порога) удаляются. Это действие не влияет на будущее распределение корректных групп.
- Что касается объединённых групп, то здесь интереснее. Опять же, факт некорректного объединения сильно заметен: пикселей в таких группах примерно в 2 раза больше, чем подразумевается (в случае двух объединённых фишек). Следовательно, такие некорректные группы так же можно найти, сравнивая число пикселей в них с порогом. Далее итеративно эти (и только эти) отобранные группы срезаются по периметру группы на ширину одного пикселя. Как правило, для рассматриваемых изображений можно зафиксировать число итераций, равное 5 для корректной работы алгоритма. После этого из группы удаляются отдельные пиксели, «отколовшиеся» во время обрезания. Все оставшиеся пиксели составляют новые группы, которые так же итеративно наращиваются по периметру для возвращения примерно исходного размера. В итоге все огромные группы делятся корректно:

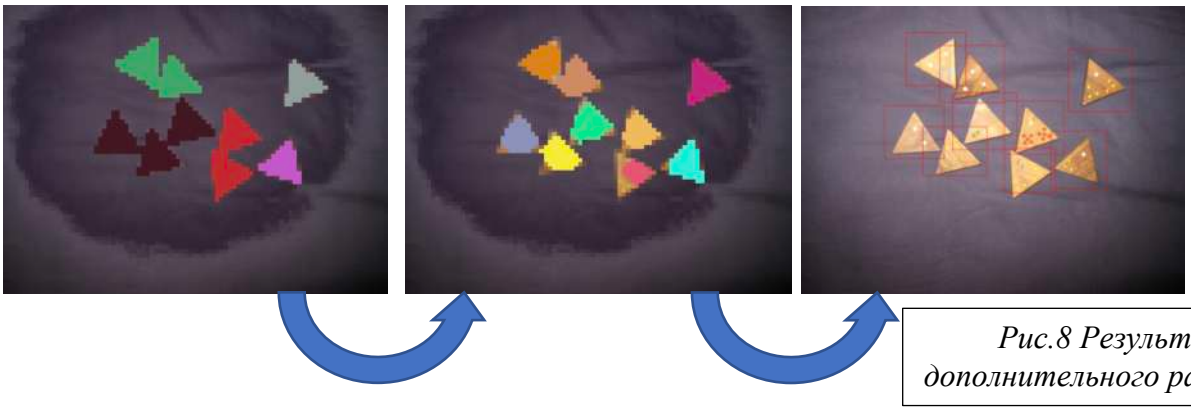


Рис.8 Результат дополнительного разбиения

Минус данного исправляющего под-алгоритма состоит в том, что получаемые новые силуэты фишек могут быть искривлены и не идеально сочетаться с исходным изображением. Однако, во-первых, это не влияет на точность используемого далее алгоритма, так как от этого этапа требуется только выделить прямоугольники, а не сами силуэты, и положение этих прямоугольников зависит только от экстремальных

крайних точек выделяемых групп, из-за чего точность не упадёт. Во-вторых, можно усилить этот под-алгоритм на этапе наращивания по периметру: производить наращивание в пределах тех пикселей, которые изначально относились к огромной группе. Однако в данной работе это не обязательно и не было применено.

На этом первый этап программы заканчивается. В качестве дополнения кратко приведём иные идеи выделения фишек на картинке (без использования цветовой составляющей). Для выделения приближённых контуров можно использовать свёрточные ядра на сжатом изображении. В качестве таких ядер подходят:

- «Контрастное» ядро (в середине – с использованием адаптивного затемнения, справа – без):

```
[-1. -1. -1.]
[-1.  8. -1.]
[-1. -1. -1.]
```

Рис.9 Результаты применения свёртки

Как видно, в данной стратегии не нужно использовать затемнение, что логично, так как с данном подходе упор делается на контуры объектов, а не цветовую составляющую. Ниже все результаты получены без затемнения.

- «Вертикальное» ядро:

```
[-1.  2. -1.]
[-1.  2. -1.]
[-1.  2. -1.]
```

Рис.9.1

- «Горизонтальное» ядро:

```
[-1. -1. -1.]
[ 2.  2.  2.]
[-1. -1. -1.]
```

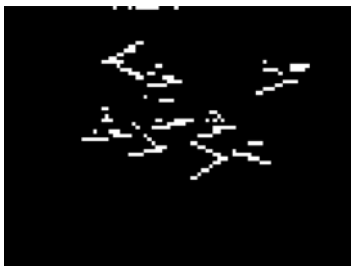


Рис.9.2

- Линейная комбинация этих ядер (очень хорошие результаты):

```
[-2.  1. -2.]
[ 1.  4.  1.]
[-2.  1. -2.]
```

Рис.9.3

Таким, образом, к данным маскам можно применить похожие алгоритмы поиска огибающих прямоугольников, что описаны выше, но с некоторыми дополнениями.

2.2 Второй этап

Перейдём ко второму этапу алгоритма. На вход подаются выделенные на прошлом этапе прямоугольники: они вырезаются из оригинального изображения – и, таким образом имеют примерно тридцатикратное уменьшение в размерах, чем оригинал. В данный момент мы можем считать, что они содержат фишки достаточно адекватно: треугольники расположены около центра этих изображений. Например, от прошлого этапа могут поступать такие изображения (это реальные результаты работы первого этапа программы):

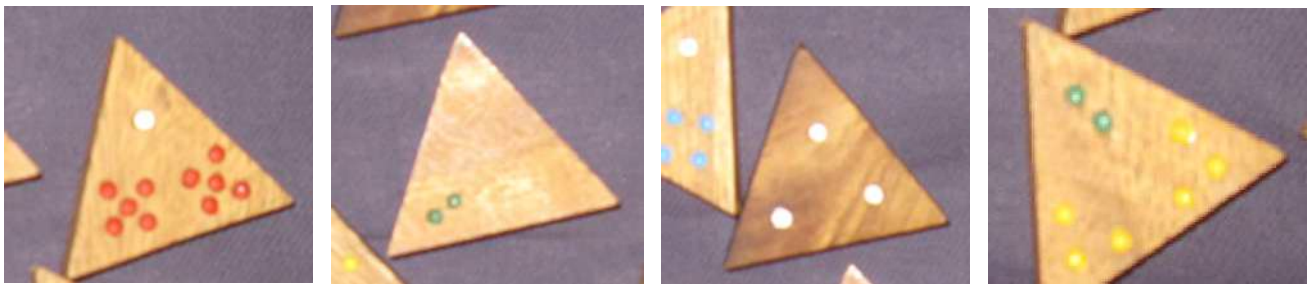


Рис.10 Результат работы 1 этапа

Текущая задача состоит в детектировании точек на данных фишках. Стоит учитывать, что на картинки могут попасть фрагменты других фишек, и их точки не нужно учитывать. Значительным упрощением при решении данной задачи служит следующее свойство: цвет точек на фишках взаимно однозначно соответствует их количеству: то есть красных точек в углу всегда пять, голубых – четыре, жёлтых – три, зелёных – две и белая – одна. Вместе с тем в задаче есть и усложнение – в углу фишки может не находиться ни одной точки. Эти аспекты будут учитываться при решении.

Для начала можно (скорее в качестве дополнительной отработки навыков, а не для повышения точности алгоритма) попробовать выделить треугольники на картинках и удалить оставшиеся фрагменты (сделать их чёрными). Для этого можно воспользоваться свёрткой с «Контрастным» ядром. При этом предварительное сжатие изображения напрямую влияет на точность алгоритма. В этой части алгоритма сжатие играет очень важную роль. Как показала практика, лучше всего сжимать изображение в три раза. Это позволяет сгладить «шероховатости» поверхности и самих фишек, что облегчает детектирование контуров посредством свёртки. После того, как свёртка применена, и контуры изображения выделены, производится заполнение внутренностей подходящего контура (как правило, нужный контур детектируется, начиная от середины маски) для получения маски. После этого данная маска накладывается на изображение, и получается обрезанная фишка.

Рис.11 Маскирование фона

Данная иллюстрация демонстрирует оптимистичный результат работы этого под-алгоритма.

На самом деле, такое хорошее качество зачастую не удаётся достичь из-за того, что в прямоугольнике оказываются другие фишки:

Рис.12 Проблемы маскирования фона

Зачастую алгоритм не может хорошо задетектировать нужные фишки (это скорее связано с программной реализацией, чем с концепцией: необходимо подобрать хорошие параметры работы алгоритма заполнения фишек, чтобы он работал качественнее, и есть некоторые баги при детектировании несуществующих контуров вдоль границ изображения). В целом, результат работы под-алгоритма не играет существенную роль в дальнейших вычислениях, и приведён здесь как потенциально рабочая концепция, которую при необходимости можно доработать.

Куда более лёгкий и эффективный способ выделить площадь фишки – опять прибегнуть к цветовому аспекту. На определённом этапе становится понятно, что «цветовое» решение способно сверхкачественно детектировать фишки. Соответственно при необходимости точного определения положения треугольника можно воспользоваться этим методом. В качестве поиска похожих пикселей используется второй метод, описанный в первом этапе (с разбиением цветового пространства через гиперплоскости). Вот некоторые результаты данного преобразования:

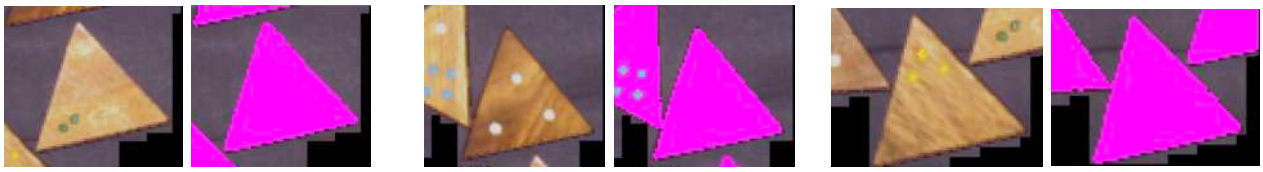


Рис.13 Результаты «агрессивного» детектирования

Алгоритм настолько хорош, что не требует тщательной настройки разделяющих гиперплоскостей (настройки подходящих цветов), и показывает такое же качество на других картинках.
Как видно, мощность поиска цветов даже захватывает некоторые типы точек на фишках. Это не является проблемой, так как решение в данном моменте подразумевает «агрессивный» поиск пикселей для фишек (то есть искусственно завышается порог похожести пикселей), а во время детектирования самих точек будет производится более точный поиск. Треугольники важно отметить (по качеству) именно так, как изображено выше, потому что большой акцент дальнейших операций будет делаться на определении принадлежности найденных точек к площади фишки.

Для того, чтобы найти точки на изображении вновь используется второй метод определения цветового сходства – через разделяющие гиперплоскости.

Требуется задать решающие правила сходства для 5 цветов: красного, голубого, жёлтого, зелёного и белого.
После этой инициализации изображение с фишкой вновь подвергается адаптивной смене яркости: если оригинал в среднем тёмный, то изображение осветляется. Если наоборот – светлый – то изображение затемняется. Это помогает достичь хороших результатов для разного типа освещённости исходных изображений. После этого производится распределение пикселей по их цветовым группам. Ниже представлены результаты работы данного алгоритма (фиолетовый цвет соответствует площади фишки, красный – всем найденным потенциальным точкам):

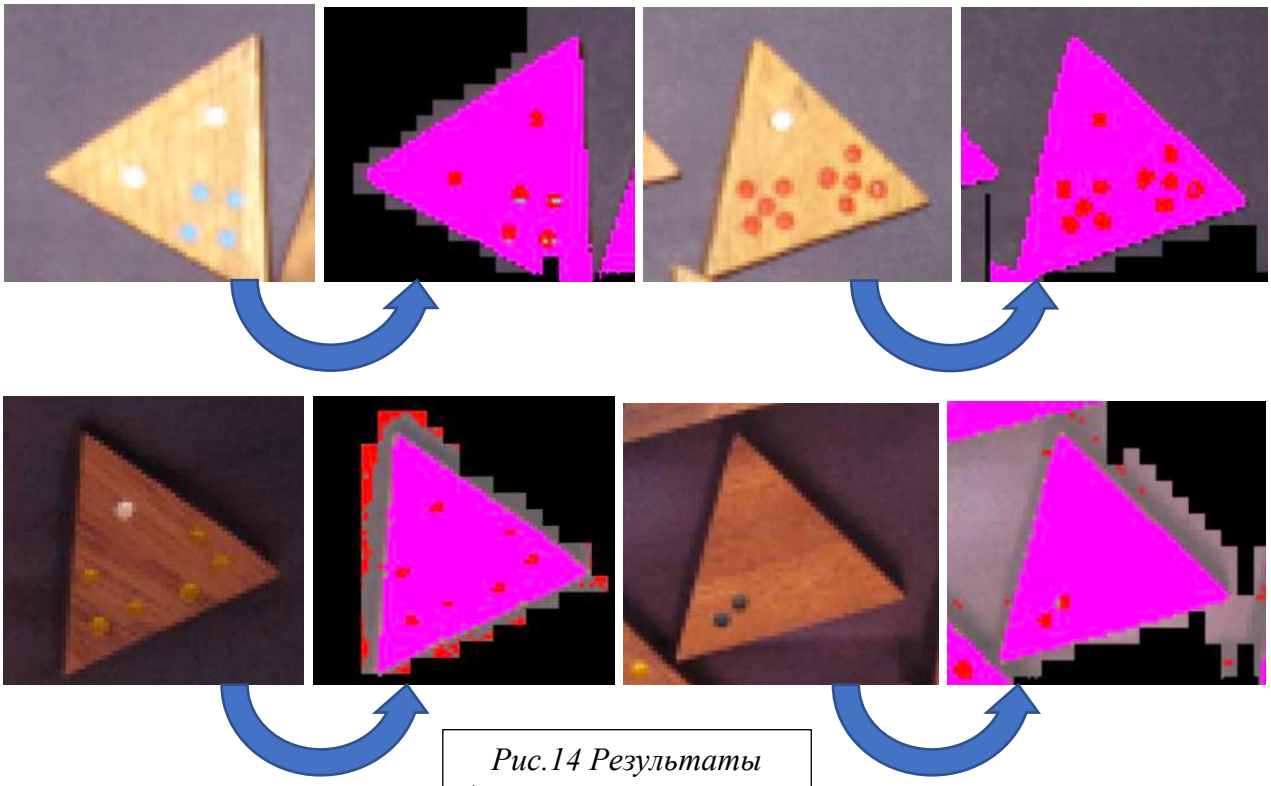


Рис.14 Результаты детектирования точек

По данным иллюстрациям видно, что распознавание фишек происходит на достойном уровне. «Агрессивное» детектирование фишек (фиолетовый цвет на рисунках) теперь открывает возможность не терять качество ответа из-за найденных подходящих цветов вне фишки, так как теперь можно точно определить её положение. Укажем, что, как правило вне фишек ошибочно детектируется только оттенок зелёного, другие цвета определяются практически без ошибок. Соответственно это приводит к выводу, что дополнительная настройка распознавания зелёных цветов может повысить качество работы программы, не использующей уточнение положения фишки. Однако в данном случае это не требуется, фишка уже замечательно отыскивается.
Отдельно отметим, что с детектирование фишек указанным способом возможно на оригинальном изображении с адаптивным изменением яркости:

Рис.15 Демонстрация возможностей «агрессивного» детектирования

Как видно, качество распознавания высокое.

2.3 Слабые стороны цветового решения и их обработка

Перед описанием заключительного этапа уточним поведение рассматриваемого алгоритма на уровне сложности «Beginner». Дело в том, что справедлив вопрос об успешности детектирования фишек на фоне, напоминающем их собственный цвет. Ответ: алгоритму, основанному на цветовом решении, действительно труднее выделить фишки из общего фона, и в данном случае лучше использовать стратегию, основанную на применении свёрточных ядер. В данных примерах они работают превосходно:

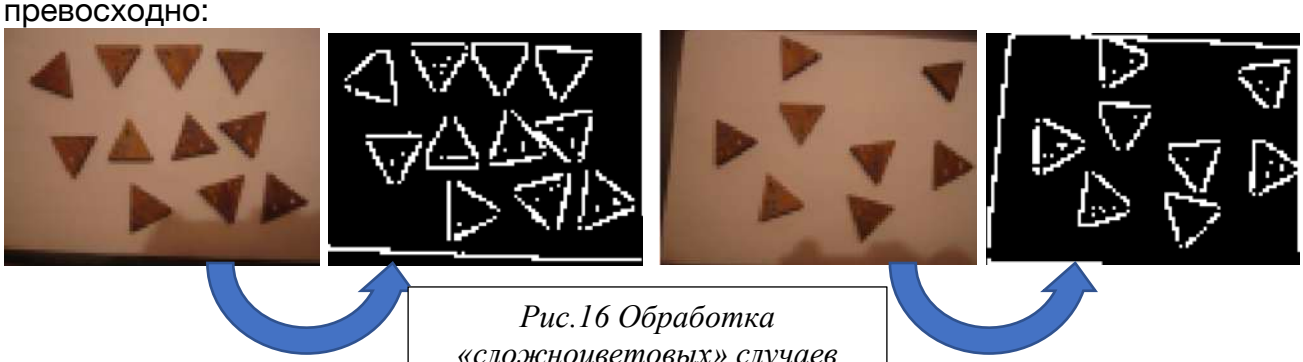
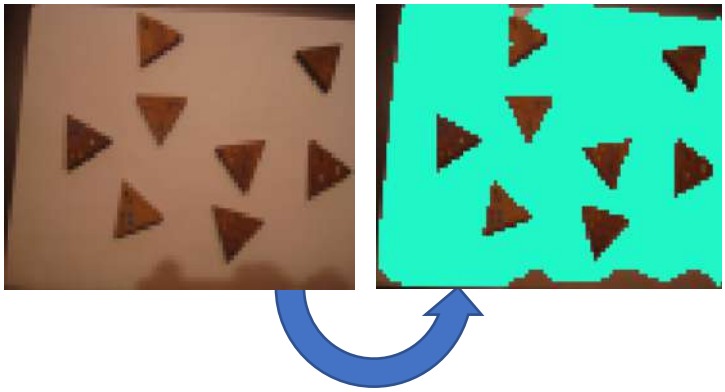


Рис.16 Обработка «сложноцветовых» случаев

Забавным фактом является то, что в данном случае алгоритм способен детектировать практически всё, кроме фишек (рис. справа). То есть в данном случае используемые инструменты не теряют свою полезность, и их можно использовать даже в таких цветово-сложных ситуациях.



Чтобы отделить случаи, удобные для цветового решения от описанных выше, достаточно определить усреднённый цвет фона, что не составляет труда, и после этого следовать вдоль одной из двух генеральных стратегий: с и без акцента на цветовом решении.

2.4 Третий этап

По сути, третий этап является прямым аддоном первого и второго этапов.
Здесь остаётся определить, какие очки наиболее вероятно изображены на фишке, а также элементарно посчитать координаты центров прямоугольников, полученных с первого этапа работы алгоритма.

Очки на фишках так же считаются довольно просто: достаточно задать решающее правило соотнесения выделенных пикселей на втором этапе к реальным точкам фишек.

Среди этих правил естественным образом возникают следующие:

- Точки должны содержать пиксели, близкие друг к другу по расположению.
- Точки должны содержать близкое к заданному число пикселей. Это задаётся двумя порогами, между которыми должно располагаться количество фишек одной точки.
- Точки, как правило, располагаются недалеко от центра изображения.
- Точки, конечно, располагаются внутри какой-нибудь фишки.

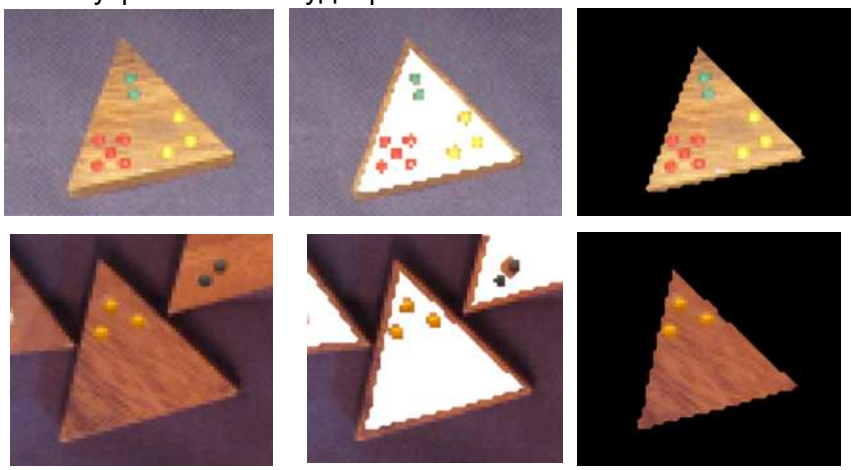


Рис.17 Работа уточнённой сегментации

Важной задачей является избежание спутывания очков на разных углах фишки. Этого можно достичь, если задать дополнительное правило: точки в разных углах фишек должны быть разделены друг от друга определённым расстоянием. Эта задача решается просто: можно сравнивать расстояние между самыми отдалёнными точками одного цвета при условии, что гарантирована их принадлежность целевой фишке. Второй способ – посчитать число выделенных точек одного цвета – опять же при гарантии того, что они принадлежат целевой фишке. Остаётся отделить целевую фишку изображения от остальных.

Здесь в полной мере пригождается «Агрессивное» детектирование площади фишки со второго этапа.

Достаточно, как и в первом этапе программы, итеративно обрезать полученные площадки по периметру, а затем оставить ту, которая располагается около середины изображения.

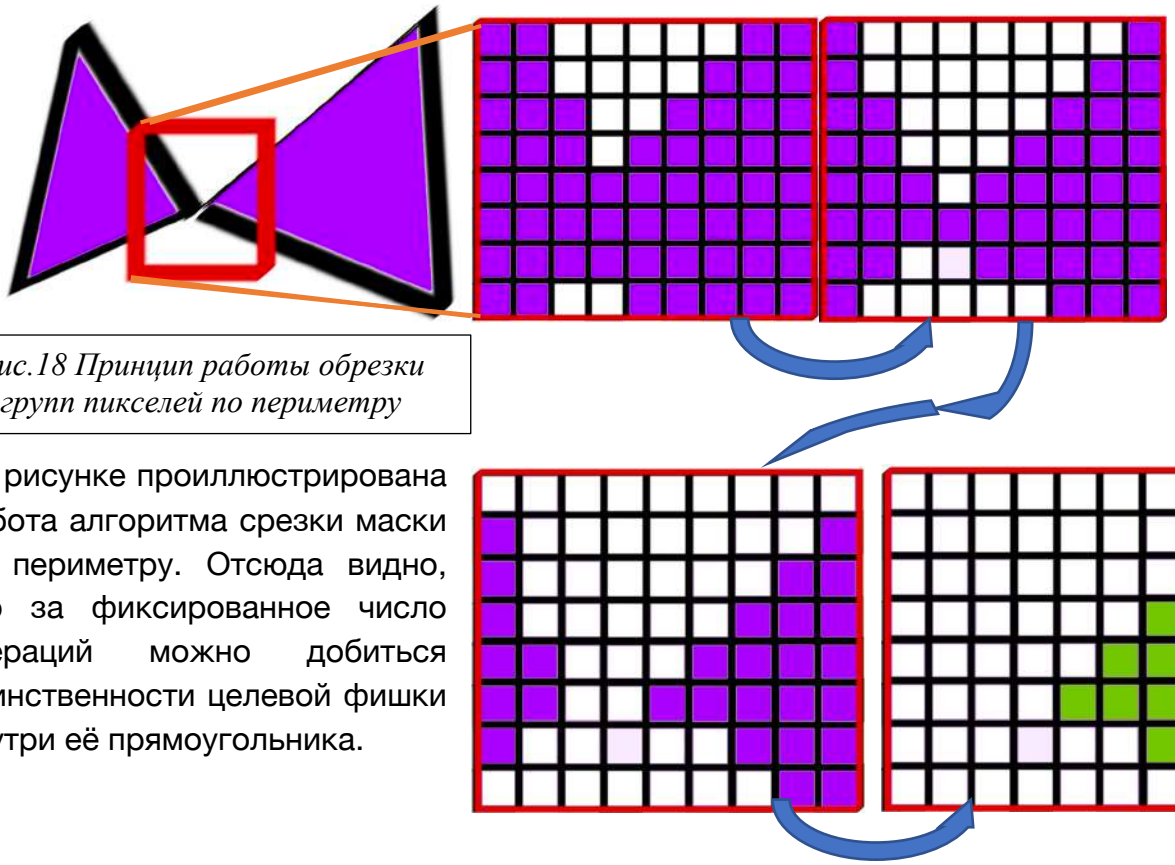


Рис.18 Принцип работы обрезки групп пикселей по периметру

На рисунке проиллюстрирована работа алгоритма срезки маски по периметру. Отсюда видно, что за фиксированное число итераций можно добиться единственности целевой фишки внутри её прямоугольника.

Эти условия и были реализованы для работы программы по подсчёту очков фишки. Для нахождения приблизительных координат её центра, как указано выше, достаточно усреднить координаты вершин огибающего её прямоугольника.

2.5 Основные аспекты программной реализации

Программа реализована на языке Python с использованием библиотеки numpy, что позволяет очень эффективно работать с большими массивами данных. Для дополнительного повышения эффективности обработки изображений используется сжатие массивов без критической потери информации.

Интерфейс для загрузки изображений в программу и получения результатов содержится в прилагаемом коде. Было решено не использовать текстовые файлы для вывода в них результатов, так как через Jupiter гораздо проще увидеть результаты работы программы.

Программа допускает ручную настройку параметров. Некоторые из них влияют на качество распознавания, некоторые – на эффективность программы. Многие параметры скрыты внутри использующих их функций. Перечислим основные регуляторы алгоритма:

- При сжатии изображений доступен для регулировки коэффициент сжатия.
- При изменении яркости возможно задание порога яркости пикселей и константа, на которую яркость будет изменена.
- При поиске похожих по цвету пикселей задаётся радиус сферы приближения в цветовом пространстве.
- Программа допускает использование свёрточных ядер. Реализованы следующие ядра: для выявления горизонтальных, вертикальных, наклонных штрихов; ядро, выявляющее контрастность; оператор Собеля.
- Для удобства визуализации с библиотекой MatPlotLib можно задать размер выводимых на экран изображений.

Описывать подробнее программную реализацию не имеет смысла, ибо все эксплуатируемые алгоритмы описаны в секциях выше, а исполнение тех или иных приёмов можно найти в прилагаемом к решению коде.

Теперь укажем файлы, прилагаемые к данному отчёту:

- Chernovik.ipynb – черновой вариант кода. Содержит все написанные процедуры (в том числе не вошедшие в финальную реализацию), и все возможные типы промежуточных результатов работы программы. Предупреждение: этот файл при использовании может замедлить работу компьютера, так как в нём содержится вывод промежуточных процедур программы, что занимает много памяти.
- Code.ipynb – чистовой вариант кода. По умолчанию выводит только конечный результат работы программы, однако допускает запуск в режиме полного вывода всех промежуточных результатов.

3. Результаты работы и выводы

В ходе выполнения данной лабораторной работы были рассмотрены алгоритмы распознавания изображений, а также изучены основные закономерности, позволяющие повысить эффективность методов, как в вычислительном, так и в качественном соотношениях. При решении основной упор был сделан на использовании цветовой метрики для сравнения и обработки пикселей. Помимо этого, в работе уточняется область применения данных алгоритмов, способы его развития и обобщения. В качестве дополнения выделяются методы, основанные на свёрточных преобразованиях, производится анализ допустимости применения этих подходов к рассматриваемой задаче и их использовании в комбинации с цветовым решением.

Краткое содержание алгоритма следующее:

1.1 Выделить по цветовому признаку маску для фишек;

1.2 Распределить по группам данные маски;

1.3 Выделить прямоугольники для каждой фишки;

2.1 Для каждой фишки детектировать целевые цвета;

2.2 Добиться единственности фишки в прямоугольниках;

2.3 Разгруппировать точки на фишках;

3.1 Произвести финальные подсчёты;

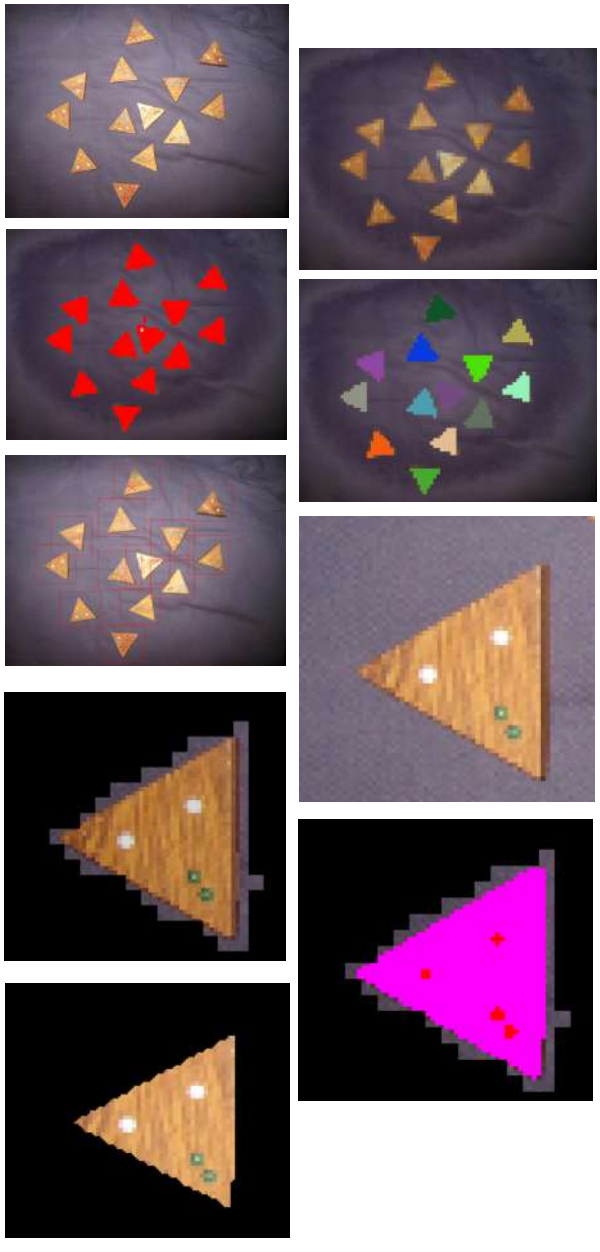


Рис.19 Визуализация работы алгоритма

В результате данный алгоритм производит обработку изображения, позволяя получить данные о расположении фишек и соответствующее им количество очков с приемлемой точностью.

Алгоритм реализован на языке Python в среде разработки Jupiter Notebook. В программе используются инструменты для векторных вычислений, что позволяет эффективно обрабатывать результаты программы. Для удобства проверки работоспособности все результаты программы выводятся там.

Ниже продемонстрирован конечный результат работы программы:

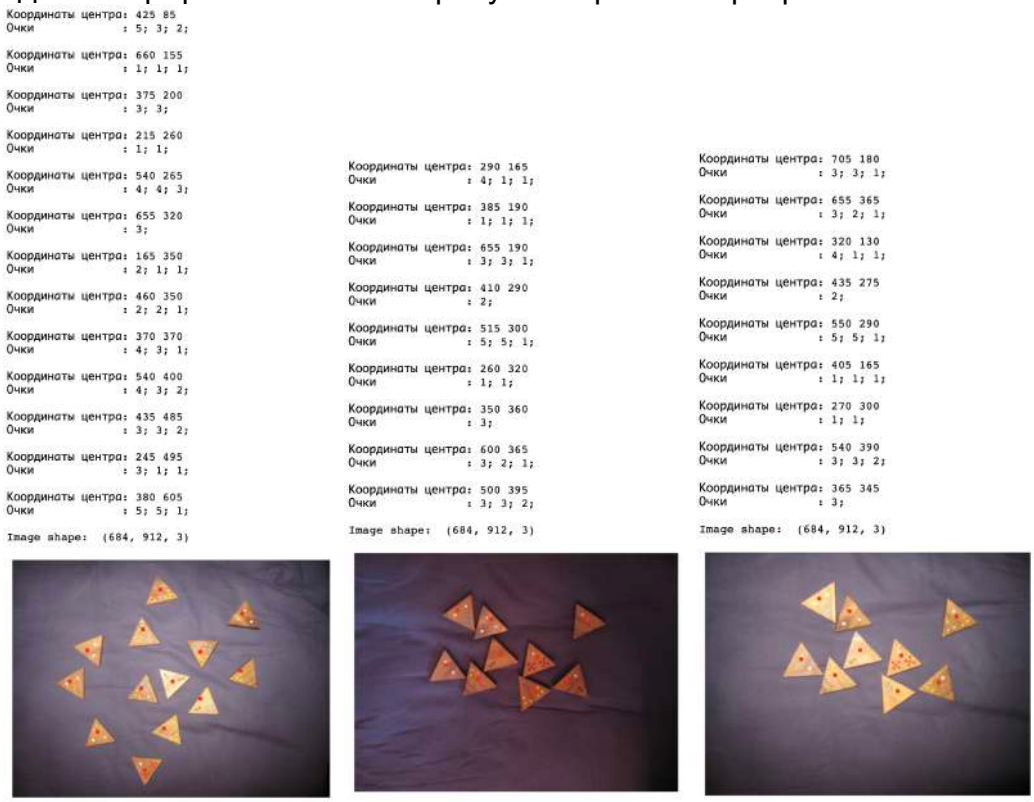


Рис.20 Результаты работы алгоритма

Стоит отметить, что на обучающих изображениях удалось получить полное совпадение числа очков для всех фишек. Как видно на изображениях, координаты фишек так же определяются достаточно хорошо. Естественно, возможен риск переобучения, но отсюда видно, что предложенный метод способен решить поставленную задачу и допускает дальнейшее улучшение. На данный момент алгоритм обрабатывает изображения за 5-10 секунд.

В качестве развития данного направления работы можно дополнительно обработать случаи с ухудшением видимых цветовых различий фона и объектов, улучшить механизм адаптивного изменения яркости, увеличить точность определения цветов, разработав более эффективную реализацию разбиения цветового пространства, а также применить к полученной модели результаты инвариантных относительно цвета программ. Плюс к этому хорошим укреплением алгоритма является его исследование на других более объёмных датасетах с установкой подходящих параметров, что повысит его качество.

4. Используемые источники:

- Гонзалес Р., Вудс Р. Цифровая обработка изображений. М., Техносфера, 2006