После исправления выше указанных ошибок, было проведено повторное тестирование, в котором наличие указанных ошибок не было выявлено.

Из этого следует, компьютерная обучающая игра «Winduza» полностью работоспособна и готова к использованию.

1. Компьютерные игры в обучении и технологии их разработки [Электронный ресурс]: статья. URL: <http://masters.dontu.org/2013/fknt/ilkun/library/igrrazr.pdf> (дата обращения: 20.10.2019).
2. Обзор игрового движка Unity [Электронный ресурс]: сайт, посвященный разработки игр. URL: <https://gscup.ru/load/engines/unity/3-1-0-2314> (дата обращения: 24.10.2019).
3. Торн А. Основы анимации в Unity: компьютерная литература. М.: Издательство ДМК, 2016. 176 с.
4. Васева Е.С., Шутько А.В. Проектирование веб-приложения «Живописнейшие места Урала». Научное обозрение. Международный научно-практический журнал. 2018. №4. С. 5.
5. Данилов О.Е. Разработка обучающих программ с помощью инструментов для создания компьютерных игр [Электронный ресурс]: статья. URL: <https://moluch.ru/archive/62/9423/> (Дата обращения: 16.10.2019).
6. Хокинг Д. Мультиплатформенная разработка на C#: компьютерная литература. Спб.: Издательство Питер, 2015. 336 с.

Федин И.А., Серов В.А.

Интеллектуальные методы обработки информации: алгоритмы распознавания образов

ФГБОУ ВО "МИРЭА - Российский технологический университет"
(Россия, Москва)

doi: 10.18411/lj-02-2020-27

idsp: ljjournal-02-2020-27

Аннотация

В статье рассмотрены самая популярная библиотека для решения задач «низкоуровневого» компьютерного зрения OpenCV, а также понятия эквализации гистограммы, блендинга, цветовых пространств. Дается представление о методах решения одной из наиболее сложных задач, решаемых наукой о компьютерном зрении, — детекции лиц.

Ключевые слова: компьютерное зрение, распознавание образов, детектор лиц.

Abstract

The article discusses the most popular library for solving the problems of OpenCV "low-level" computer vision, as well as the concepts of histogram equalization, blending, color spaces. An idea is given of methods for solving one of the most complex problems solved by the science of computer vision - face detection.

Key words: computer vision, pattern recognition, face detector.

В настоящее время проводится значительное число исследований, рассматривающих методы и алгоритмы машинного обучения в задачах распознавания образов [1-5]. Так, в статье А.В. Гульянова рассматриваются базовые методы и алгоритмы, применяющиеся в современных системах распознавания образов [1]. Для повышения результативности распознавания В.И. Мунерман и В.Д. Чикунов предлагают параллельную реализацию алгоритмов, имеющих принципиальные различия [2]. Хорошо зарекомендовали себя алгоритмы распознавания образов на основе анализа геометрических свойств точек контура [3]. Вместе с тем, по мнению Л.А. Лютиковой и Е.В. Шматовой, целесообразно применять логистический подход к синтезу алгоритмов распознавания образов [4]. Часто для распознавания образов применяют байесовский алгоритм классификации [5]. Однако окончательный выбор

алгоритма распознавания образов зависит от уровня сложности изображения распознаваемого объекта.

Поговорим о задачах «простого», или «низкоуровневого», компьютерного зрения. Они нередко используются как составная часть более сложных задач распознавания. Например, предобработка картинки позволяет алгоритмам машинного обучения лучше понять, что на ней изображено.

Самая популярная библиотека для решения задач «низкоуровневого» компьютерного зрения — это *OpenCV*. В ней содержится огромное количество алгоритмов, есть интерфейсы для многих языков программирования, в том числе C++ и Python. Другая известная библиотека — *skimage*, она активно используется в скриптах на Python. В примерах дальше мы будем применять OpenCV.

Изображение в памяти компьютера кодируется попиксельно, информация о пикселе в зависимости от используемой модели может быть записана по-разному, но в общем случае представление изображения в памяти компьютера — это числа (рисунок 1).

Рисунок 1. Представление изображения в памяти компьютера

Цвет каждого пикселя в зависимости от используемой модели так же может быть записан по-разному. Самая простая модель — это RGB, в соответствии с которой тремя числами кодируется яркость пикселей в красном, зеленом и синем каналах.

Рассмотрим *арифметические операции*. Итак, картинки — это матрицы чисел. В случае с черно-белыми картинками это матрицы размера высота на ширину картинки. В случае с цветной у матрицы появляется еще одна размерность, чаще всего она равна трем.

В OpenCV используется такое же представление матриц, как в библиотеке NumPy. Это значит, что для них можно использовать стандартные арифметические операции, например сложение.

Однако не все так просто: сложение матриц в NumPy не учитывает переполнение. Для изображений переполнение — это нелогичная операция. Если при сложении двух картинок яркость где-то превысила 255, то, как правило, она должна оставаться равной 255, а не превратиться в 4.

Пример ниже показывает, как отличается сложение в NumPy и OpenCV.

```
import numpy as np
import cv2
x = np.uint8([250])
y = np.uint8([10])
print cv2.add(x,y)
```

```
[[255]]
```

```
x+y
```

```
4
```

Для примера возьмем картинку (рисунок 2).

Рисунок 2. Образец изображения

Первым делом изображение нужно сделать серым (даже если оно на вид уже в оттенках серого, в файле, из которого мы загружаем, оно обычно сохранено как цветное).

```
img1 = cv2.imread('imgs/lenna.jpg')
gray = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
```

Команда `cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)` будет неоднократно встречаться в дальнейшем. Она используется для преобразования цветовых пространств, в том числе из RGB в серые картинки. После превращения картинки в серую можно прибавить к ней какое-то число.

```
gray_add = cv2.add(gray, 50)
```

Такое преобразование эквивалентно увеличению яркости картинки (рисунок 3).

Рисунок 3. Преобразование картинки

Можно не прибавлять, а умножать на некий коэффициент.

```
gray_mul = cv2.multiply(gray, 1.3)
```

Умножение картинки эквивалентно увеличению ее контрастности. Можно попробовать использовать больший коэффициент (например, 1.8) – рисунок 4.

Рисунок 4. Увеличение контрастности

Именно так и работают алгоритмы изменения яркости и контраста во многих популярных графических редакторах. Однако для этой цели можно использовать и более сложные функции.

Рассмотрим *эквализацию гистограммы*. В данном случае гистограмма — это представление картинки, по которому видно, сколько в ней пикселей той или иной яркости. Ниже показана гистограмма какого-то изображения (рисунок 5). Черная линия — это кумулятивная гистограмма, которая отвечает на вопрос, у какого количества пикселей яркость меньше значения x .

Рисунок 1.5. Гистограмма до эквализации и после

В результате эквализации гистограмма картинки растягивается таким образом, чтобы кумулятивная гистограмма была близка к линейной функции. Выполнить эквализацию можно следующей функцией:

```
equ = cv2.equalizeHist(gray)
```

Если взять нашу предыдущую картинку, то результат будет выглядеть вот так (рисунок 6).

Рисунок 6. Результат эквализации

Рассмотрим *блендинг*. Это еще один пример применения простых арифметических операций к картинкам. Если наша задача — скомбинировать два изображения, то можно попробовать их сложить. Но в таком случае, если объекты наложатся друг на друга, получится «каша» (рисунок 7).

Рисунок 7. Результат Блендинга

Пусть для одной картинки известно, где расположен объект, а все остальное пространство занимает фон. Тогда можно помещать второй объект туда, где находится фон. В месте, где первый объект накрывается вторым, будет также использоваться второй объект.

Такое объединение требовательно к качеству вырезания картинки. Если по краям неаккуратно обрезан фон, то будет видна некрасивая белая полоса.

Кажется, что научиться аккуратно вырезать объект из фона — сложная задача. Это так, потому что фон неоднородный и недостаточно просто выбросить белые пиксели. Можно воспользоваться хитрым алгоритмом смешивания двух картинок и построить маску таким образом, что ее значение будет тем больше, чем дальше пиксель от белого.

Там, где на исходном изображении располагаются белые пятна, будут браться пиксели со второго изображения, и неаккуратное вырезание объекта будет не так заметно. Выше (рисунок 7) можно увидеть, как такое простое преобразование помогает избавиться от проблемы.

Существуют более сложные алгоритмы блендинга. В случаях, когда требуется скопировать объект с неоднородным фоном и вставить его в другое изображение, простые методы, смешивающие цвета, не помогают. Методы похитрее используют оптимизацию, чтобы определить, где находится объект, а где — фон. Затем свойства объекта переносятся без изменений, а свойства фона берутся с картинки, на которую вставляется объект (рисунок 8).

Рисунок 8. Сложный алгоритм блендинга

Рассмотрим метод *цветовых пространств*. Ранее мы говорили только о представлении в RGB, но существуют и другие варианты. Например — HSV.

Компоненты этого пространства — тон (hue), насыщенность (saturation) и значение (value) – рисунок 9.

Рисунок 9. Метод «Цветовые пространства»

Это пространство позволяет манипулировать цветом и его насыщенностью по отдельности. Тон обозначает цвет пикселя, он закодирован числом от 0 до 360, как угол на цилиндре. Насыщенность принимает значение 0, если картинка серая.

Когда мы работаем с картинкой в HSV, то можем легко сделать ее более насыщенной, умножив соответствующий канал на какой-то коэффициент. Попробуем увеличить насыщенность на 50% (коэффициент 1,5) – рисунок 10.

```
hsv = cv2.cvtColor(img1, cv2.COLOR_BGR2HSV)
hsv[:, :, 1] = cv2.multiply(hsv[:, :, 1], 1.5)
result = cv2.cvtColor(hsv, cv2.COLOR_HSV2BGR)
```

Рисунок 10. Увеличение насыщенности на 50%

Рассмотрим метод *детектор лиц* — Каскады Хаара. Одна из задач, решаемых наукой о компьютерном зрении, — это детекция лиц. Среди первых методов решения наиболее удачный — это каскады Хаара. Применяя этот метод, из картинки можно вычленять достаточно простые признаки. Для этого необходимо использовать несколько прямоугольников (рисунок 11).

Рисунок 11. Каскады Хаара

Пиксели, попадающие в белый прямоугольник, берутся со знаком «плюс», в черный — со знаком «минус». Все значения суммируются, и получается одно число. Прямоугольники и коэффициенты для них выбираются с помощью алгоритма AdaBoost. У лица имеются некоторые паттерны, и в итоге каскад фильтров такого рода показывает, есть ли внутри него лицо или нет.

Сейчас есть методы детекции лиц, превосходящие по качеству каскады Хаара. Тем не менее, это простой подход, и он часто встречается в готовом к использованию виде. Если не требуется решать задачу с высоким качеством, а получить детектор нужно быстро и просто, каскады Хаара из библиотеки OpenCV — это отличный вариант.

Следующим рассмотрим метод сегментации. Задачу сегментации в общем случае решить тоже несложно. Один из способов — отрезать пиксели выше какого-то порога и назначить их объекту, а пиксели ниже порога — фону (рисунок 12).

Рисунок 12. Сегментация изображения

На этом примере видно, что монеты намного темнее, поэтому достаточно подобрать такую границу, чтобы они все оказались ниже. Вот код, который позволяет это сделать при помощи OpenCV:

```
img = cv2.imread('coins.png')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
ret, thresh = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)
```

Рассмотрим способ - *Линейная фильтрация изображений*. Важный класс преобразования изображений — это линейные фильтры. С их помощью решаются задачи поиска границ, углов, удаления шумов.

Способ *скользящее среднее — свертка*. Проще всего объяснить, что такое линейная фильтрация, на примере. Пусть требуется подсчитать среднее в окне 3×3 для каждого пикселя. Вычисление среднего можно записать так:

$$g[n, m] = \frac{1}{9} \sum_{k=-1}^{n+1} \sum_{l=-1}^{m+1} f[k, l] = \frac{1}{9} \sum_{k=-1}^1 \sum_{l=-1}^1 f[n-k, m-l] \quad (1)$$

Переписав формулу в следующем виде, можно получить выражение для свертки:

$$(f \cdot h)[m, n] = \frac{1}{9} \sum_{k,l} h[m-k, n-l], \quad (2)$$

здесь f — это изображение (двухмерная функция, характеризующая картинку), k, l — координаты пикселя, f — яркость пикселя, h — ядро свертки (матрица 3×3 , состоящая из единиц).

$$\frac{1}{9} \begin{matrix} & h \\ \begin{matrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{matrix} \end{matrix}$$

Если ядро свертки — матрица, то свертка — это скользящее среднее. В OpenCV произвести такую свертку можно следующим образом:

```
kernel = np.array([[1,1,1],[1,1,1],[1,1,1]], np.float32) / 9
dst = cv2.filter2D(img1, -1, kernel)
```

Картинка при этом становится более размытой. Также размытие изображения можно получать при помощи свертки с гауссовской функцией (рисунок 13).

```
cv2.GaussianBlur(img1, (5,5), 0)
```

Рисунок 13. Фото до применения скользящего среднего, после и после применения гауссовского размытия.

Далее рассмотрим способ - детекция границ. С помощью сверток, как на картинке ниже, можно получить вертикальные и горизонтальные границы изображения. Если объединить результаты этих двух сверток, можно получить все границы.

-1	-1	-1
0	0	0
1	1	1

-1	0	1
-1	0	1
-1	0	1

Свертки для получения границ: слева — горизонтальных, справа — вертикальных

Рисунок 14. Границы изображения, полученные с помощью применения сверток. Слева — горизонтальные, в середине — вертикальные, справа — все границы, полученные объединением результатов двух сверток

Такие ядра являются частью преобразования Превитта, их использование — это самый простой способ найти границы изображения.

На самом деле существует много способов определения границ. Каждый из них применяется в разных условиях, и, в зависимости от задачи, необходимо использовать тот или иной способ.

Рассмотрим способ определения – Корреляция. Данный пример линейного преобразования очень похож на свертку, но записывается немного в другом виде:

$$r_{f,g}[k,l] = \sum_{n=-\infty}^{\infty} \sum_{m=-\infty}^{\infty} f[n,m]g*[n-k,m-l] = \sum_{n=-\infty}^{\infty} \sum_{m=-\infty}^{\infty} f[n+k,m+l]g*[n,m], \quad k,l \in \mathbb{Z}. \quad (3)$$

В отличие от свертки корреляция используется, чтобы показать меру похожести двух изображений. Это может быть использовано для поиска объектов. Например, требуется найти лицо футболиста (рисунок 15).

Рисунок 15. Корреляция

На рисунке слева показан результат применения корреляции для поиска лица. Белое пятно — это место, где оно найдено. Корреляцию можно использовать с различными параметрами: нормировать, применять ее различные вариации.

Получается, что корреляция — это очень простой способ поиска объектов на изображении, если имеется их точные копии.

Как видим, для улучшения работы систем анализа и обработки медицинских изображений, очевидно, необходим метод, обеспечивающий автоматизированный выбор преобразования изображения.

1. Гурьянов А.В. Методы и алгоритмы машинного обучения в задачах распознавания образов // Молодежный научно-технический вестник. 2015. № 3. С. 22-28.
2. Мунерман В.И., Чикунов В.Д. Параллельная реализация алгоритмов распознавания образов // Системы компьютерной математики и их приложения. 2015. № 16. С. 91-93.
3. Дмитриев В.Л. Об алгоритме распознавания образов на основе анализа геометрических свойств точек контура // Информатизация образования и науки. 2016. № 1 (29). С. 138-146.
4. Лютикова Л.А., Шматова Е.В. Логический подход к синтезу алгоритмов распознавания образов // Известия Кабардино-Балкарского научного центра РАН. 2015. № 6-2 (68). С. 112-116.
5. Абдикеримова Г.Б. реализация байесовского алгоритма классификации для распознавания образов средствами MATLAB // Научная дискуссия: вопросы технических наук. 2016. № 7-8 (37). С. 6-13.

Чернецова Е.А., Шишкин А.Д.

Сравнительная характеристика способности датчиков системы дистанционного мониторинга обнаруживать нефтяные загрязнения водной поверхности

*Российский Государственный Гидрометеорологический Университет
(Россия, Санкт-Петербург)*

doi: 10.18411/lj-02-2020-28

idsp: ljjournal-02-2020-28

Аннотация

Рассматриваются различные типы датчиков, которые могут применяться в системе дистанционного мониторинга водной поверхности и оценивается их способность обнаруживать и определять параметры нефтяных загрязнений водной поверхности. Обобщены требования к датчикам дистанционного мониторинга нефтяных загрязнений. Проанализированы преимущества и недостатки мониторинга нефтяных загрязнений с помощью спутника и с помощью самолета. Показано, что повысить достоверность информации, поступающей в центр вторичной обработки данных мониторинга можно с помощью слияния информации, поступающей от различных сенсоров. Приводится сравнение уровней слияния данных.

Ключевые слова: водная поверхность, дистанционный мониторинг, нефтяное загрязнение, сигнал, датчик, слияние данных

Abstract

Various types of sensors that can be used in a system for remote monitoring of the water surface are considered and their ability to detect and determine the parameters of oil pollution of the water surface is evaluated. The requirements for sensors remote monitoring of oil pollution are summarized. The advantages and disadvantages of monitoring oil pollution by satellite and by plane are analyzed. It has been shown that it is possible to increase the reliability of the information received at the secondary processing center of the monitoring data by merging the information received from various sensors. A comparison of the levels of data fusion is given.

Keywords: water surface, remote monitoring, oil pollution, signal, sensor, data fusion