

Урок 3

Компьютерное зрение

3.1. Компьютерное зрение

Компьютерное зрение — это область науки, которая занимается разнообразными задачами, связанными с анализом изображений и видео. Можно считать, что во всех них требуется ответить на вопрос, что изображено на картинке. Несмотря на кажущуюся тривиальность вопроса, ответить на него не так просто.

3.1.1. Классификация сцены



Рис. 3.1

На рисунке 3.1 показана картинка. Чтобы ответить на вопрос, что на ней изображено, можно описывать сцену в целом. Понятно, что картинка сделана на улице (вне помещения), где-то в азиатской стране (кто-то может узнать площадь Тяньаньмэнь в Пекине).

Другой подход — выделять отдельные объекты на изображении (рис. 3.2). На картинке видно автобус, портрет, крышу, небо и т.д.

Можно пойти дальше и говорить про физические свойства отдельных объектов. Например, крыша — наклонная, автобус едет, и он твёрдый, на стене висит изображение Мао Цзедуна, ветер дует справа налево (это можно определить по движению флага).

Из примера выше можно заключить, что для ответа на вопрос, что изображено на рисунке, используется весь жизненный опыт. Например, знание о том, что существует ветер (на картинке его нельзя увидеть в явном виде), что такое транспорт. Для того чтобы ответить на определённые вопросы, необходимо знать историю Китая. Соответственно задача заключается не в том, чтобы смотреть на пиксели, а в использовании знаний.



Рис. 3.2

3.1.2. Внутриклассовая изменчивость



Рис. 3.3

Другой пример. На вопрос, что такое стул, можно ответить первое, что придёт в голову. Например, стул — это нечто с четырьмя ножками и спинкой. Фотография на рисунке 3.3 показывает, что не все стулья подходят под это определение.

Стул достаточно сложно описать в терминах форм. Стул — это некое концептуальное понятие, то, на чём сидят. Можно представить, как сложно объяснить это понятие инопланетному существу, которое не знает даже, что такое сидеть и не умеет это делать. Прежде чем научить существо находить на картинках стул, было бы неплохо, чтобы оно поняло концепцию "сидеть". Абсолютно то же самое происходит, когда компьютер учат распознавать изображения. В идеале, чтобы он отвечал на вопросы про стулья так же хорошо, как человек, ему нужно понимать концепцию "сидеть".

3.1.3. Искусственный интеллект

В науке об искусственном интеллекте существует понятие "ИИ-сложные задачи". Это класс задач, решение которых эквивалентно созданию искусственного интеллекта. Считается, что задача компьютерного зрения в общей постановке (ответ на вопрос, что изображено на картинке, и на все вопросы про это изображение) является ИИ-сложной. Выше было продемонстрировано, что действительно, для ответа на вопрос об изображении не просто смотреть, а использовать весь свой жизненный опыт, образование, а иногда и интуицию.

К сожалению, искусственный интеллект до сих пор не создан. Поэтому наука о компьютерном зрении решает только определённые подзадачи, речь о которых пойдёт далее.

3.2. Задачи компьютерного зрения

Далее будут приведены примеры нескольких задач, которые решаются с применением технологии компьютерного зрения.

3.2.1. Поиск по изображению

Первый пример — поиск изображений в интернете. Сейчас существует несколько сервисов, которые позволяют искать картинки. Изначально для поиска использовались текстовые запросы. Некоторое время назад в части из таких сервисов появилась возможность поиска по загруженному изображению. От пользователя требуется загрузить картинку, а сервис будет искать похожие на неё изображения в интернете.

На самом деле, это работает следующим образом. Сначала индексируются изображения из интернета. Для них строятся некоторые цифровые представления, из которых формируется структура данных, по которой можно быстро производить поиск. Для пользовательской картинки также используется цифровое представление, по которому в сформированной структуре данных ищутся дубликаты или похожие картинки.

Данная задача является сложной в структурном смысле. В интернет загружены миллиарды изображений, и использование сложных методов сравнения невозможно, потому что необходимо достигать высокой производительности.

3.2.2. Распознавание текста

Следующий пример — это технология распознавания теста. Необходимо найти изображение текста на картинке и представить его в виде текстовых данных, с которыми можно будет работать, например, в редакторе. Эта технология используется в разнообразных приложениях. В частности, это удобный способ вводить текст в онлайн-переводчик. Достаточно сфотографировать этикетку, текст на ней будет распознан, и переводчик выполнит перевод (рис. 3.4).

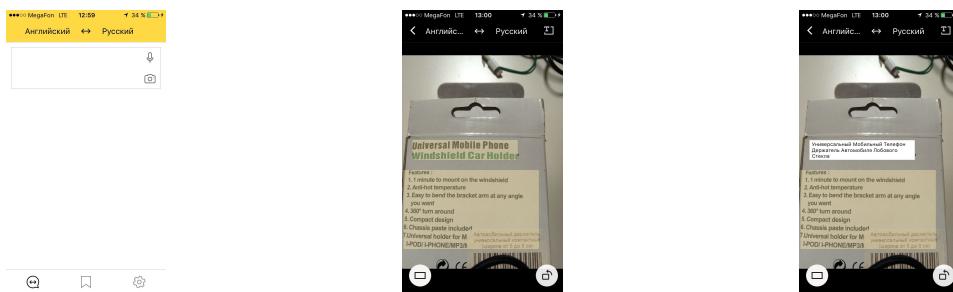


Рис. 3.4: Пример использования технологии распознавания речи в приложении-переводчике

3.2.3. Биометрия

Ещё одна задача, решаемая в рамках науки о компьютерном зрении, — это биометрия, распознавание людей. Для этого может использоваться изображение лица, радужная оболочка глаза, отпечатки пальцев. Однако в основном компьютерное зрение занимается распознаванием лиц. С каждым годом эта технология работает всё лучше и лучше, и находит широкое применение.

3.2.4. Видеоаналитика

В мире устанавливается всё больше камер: на дорогах для регистрации движения автомобилей или в общественных местах для отслеживания потоков людей и детектирования аномалий (например, оставленные вещи, нелегальные действия). Как следствие, возникает задача анализа огромного потока появившейся информации. Компьютерное зрение помогает в решении этой задачи. Оно позволяет детектировать номер автомобиля, его марку, нарушает ли он правила дорожного движения.

3.2.5. Анализ спутниковых снимков

В данный момент накоплен огромный массив спутниковых снимков. Используя эти данные, можно решать разнообразные задачи: улучшать карты, детектировать лесные пожары и другие проблемы, которые видны со спутника. Технологии компьютерного зрения шагнули в последнее время далеко вперёд, и с их использованием автоматизируется всё больше ручной работы в этой области.

3.2.6. Графические редакторы

Компьютерное зрение позволяет не только распознавать, что изображено на картинке. Также оно позволяет изменять и улучшать изображения. Таким образом, всё, что можно сделать с помощью графического редактора, относится к технологии компьютерного зрения.

3.2.7. 3D анализ

3D-реконструкция — ещё один пример задачи, решаемой с помощью компьютерного зрения. Используя множество изображений, сделанных в данном городе, можно восстановить форму зданий.

3.2.8. Компьютерное зрение и авто

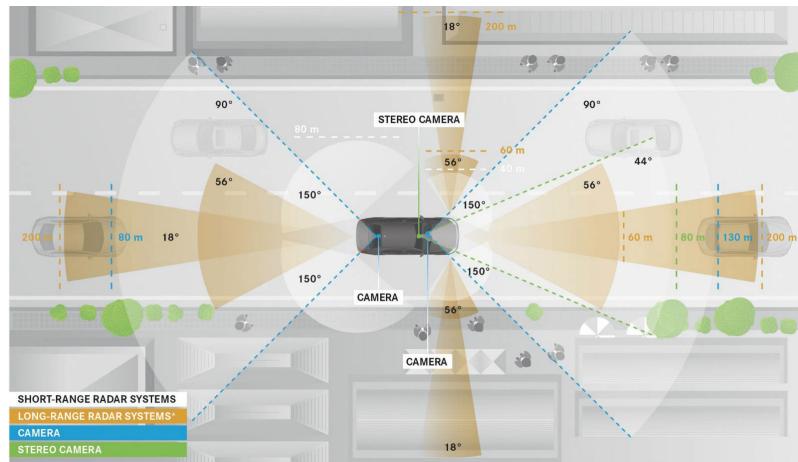


Рис. 3.5

Современные автомобили оснащены огромным количеством датчиков: несколько видеокамер, радары, стереокамера (рис. 3.5). Методы компьютерного зрения помогают анализировать информацию, получаемую с этих датчиков. С использованием этих методов созданы системы предотвращения ДТП, столкновения с пешеходами и предупреждения водителя о разнообразных препятствиях.

Кроме того, сейчас активно развивается область автопилотируемых автомобилей. В них технологии компьютерного зрения используются для ориентирования в пространстве.

3.3. "Низкоуровневое" зрение

Методы компьютерного зрения используются для решения задач, которые условно можно разделить на простые и сложные. Сложные задачи отвечают на вопросы, какой объект изображён на картинке, к какому

классу он относится. Для решения этих задач чаще всего используются методы машинного обучения. При решении простых задач производятся манипуляции непосредственно с пикселями, используются эвристики, а методы машинного обучения, как правило, не применяются. Этот раздел посвящён задачам "простого" компьютерного зрения, "низкоуровневому" зрению. Стоит отметить, что они часто используются как составная часть более сложных задач распознавания. Например, предобработка картинки позволяет алгоритмам машинного обучения лучше распознавать, что на ней изображено.

3.3.1. Библиотеки

OpenCV — это самая популярная библиотека, используемая для решения задач "низкоуровневого" компьютерного зрения. В ней содержится огромное количество алгоритмов, есть интерфейсы для языков C++ и python. Другая известная библиотека — skimage, она активно используется в скриптах на языке python. В этом курсе для демонстрации примеров будет использоваться библиотека opencv.

3.3.2. Представление изображений



0	3	2	5	4	7	6	9	8
3	0	1	2	3	4	5	6	7
2	1	0	3	2	5	4	7	6
5	2	3	0	1	2	3	4	5
4	3	2	1	0	3	2	5	4
7	4	5	2	3	0	1	2	3
6	5	4	3	2	1	0	3	2
9	6	7	4	5	2	3	0	1
8	7	6	5	4	3	2	1	0

Рис. 3.6: Представление изображения в памяти компьютера

Прежде всего необходимо разобраться, как картинки представляются в памяти компьютера. Если человек видит некие цельные картины и объекты, то в памяти компьютера это числа (рис. 3.6). Изображение состоит из пикселей. Если оно не цветное, то каждый пиксель описывается яркостью (для однобайтовых изображений яркость принимает значения от 0 до 255). Если в картинке есть цвет, описание пикселя состоит из трёх чисел. Самое простое и интуитивно понятное из них — это яркости изображения в красном, зелёном и синем канале. Существуют также и другие представления цвета, о них речь будет идти позже.

3.3.3. Арифметические операции

Итак, картинки — это матрицы чисел. В случае чёрно-белых картинок это матрицы размера высота на ширину картинки. В случае цветных изображений у матрицы появляется ещё одна размерность, чаще всего она равна трём.

В opencv используется такое же представление матриц, как в библиотеке numpy. Это означает, что для них можно использовать стандартные арифметические операции, например, сложение. Однако не всё 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]
```



Рис. 3.7

Работу этих операций можно продемонстрировать на примере рисунка 3.7. Сначала изображение нужно сделать серым.

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



Рис. 3.8

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 (рис. 3.8):

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

Такое преобразование эквивалентно увеличению яркости картинки. Можно не прибавлять, а умножать на некий коэффициент (рис. 3.8):

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

Умножение картинки эквивалентно увеличению её контрастности. Можно попробовать использовать больший коэффициент (рис. 3.9а):



(a)



(b)

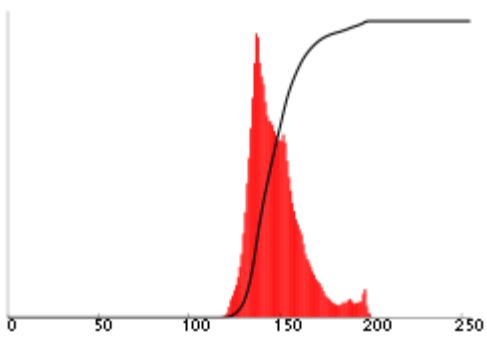
Рис. 3.9: Результат умножения картинки на число

```
gray_mul = cv2.multiply(gray, 1.8)
```

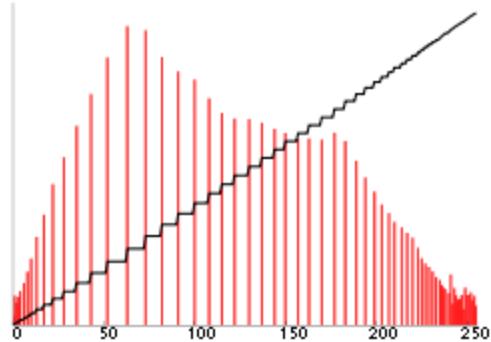
Ранее был рассмотрен пример того, как меняется изображение после сложения или умножения на некоторое число. Именно так и работают алгоритмы изменения яркости и контраста во многих популярных графических редакторах и мобильных приложениях. Однако для этой цели можно использовать и более сложные функции.

3.3.4. Эквилизация гистограммы

Пример такого подхода — это эквилизация гистограммы. В данном случае гистограмма — это представление картинки, показывающее, сколько в ней пикселей той или иной яркости. На рисунке 3.10а показана гистограмма какого-то изображения. Чёрная линия — это кумулятивная гистограмма, отвечающая на вопрос, у какого количества пикселей яркость меньше значения x .



(a)



(b)

Рис. 3.10: Гистограмма до эквилизации (а) и после (б)

В результате эквилизации гистограмма картинки растягивается таким образом, чтобы кумулятивная гистограмма была близка к линейной функции. Результат эквилизации изображения показан на рисунке 3.11. Он получен с помощью следующей функции:

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

Видно, что в результате контраст изображения улучшился: тёмные области стали темнее, яркие — светлее.

Чтобы произвести эквилизацию гистограммы, нужно применить некое преобразование к яркости пикселей. Конкретный вид преобразования можно попробовать получить самостоятельно, это хорошая задачка на алгоритмы.



Рис. 3.11: Результат эквилилизации гистограммы

3.3.5. Блендинг

Блендинг — это ещё один пример применения простых арифметических операций к картинкам. Ставится задача скомбинировать два изображения. Можно попробовать их сложить. Но в таком случае, если объекты наложатся друг на друга, то получится каша.



Рис. 3.12: Совмещение двух изображений

Пусть для одной картинки известно, где расположен объект, а всё 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 достаточно требовательно к качеству вырезания картинки. Если по краям неаккуратно обрезан фон, то по краям будет видна некрасивая белая полоса (рис. 3.12a).

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 смещивания двух картинок, и построить маску таким образом, что её значение будет тем больше, чем дальше пиксель от белого. В таком случае там, где на исходном изображении располагаются белые пятна, будут браться пиксели со второго изображения, и неаккуратное вырезание объекта будет не так заметно. На рис. 3.12 показано, как такое простое преобразование помогает избавиться от этой проблемы.

Существуют более сложные алгоритмы блендинга. В случаях, когда требуется скопировать объект с неоднородным фоном и вставить его в другое изображение (рис. 3.13), простые методы, смешивающие цвета не помогают. Существуют более хитрые методы, использующие оптимизацию, чтобы определить, где находится



Рис. 3.13: Источник: Pérez, Patrick, Michel Gangnet, and Andrew Blake. "Poisson image editing."

объект, а где — фон. Затем свойства объекта переносятся без изменений, а свойства фона берутся с картинки, на которую вставляется объект.

3.3.6. Цветовые пространства

Ранее уже упоминались цветовые пространства. Самое интуитивно понятное и популярное из них — представление цвета в каналах красного, зелёного и синего, RGB.

HSV

Другой пример цветового пространства — это HSV (рис. 3.14). Компонентами этого пространства являются тон (hue), насыщенность (saturation) и значение (value). 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 (рис. 3.14). Насыщенность принимает значение 0, если картинка серая.

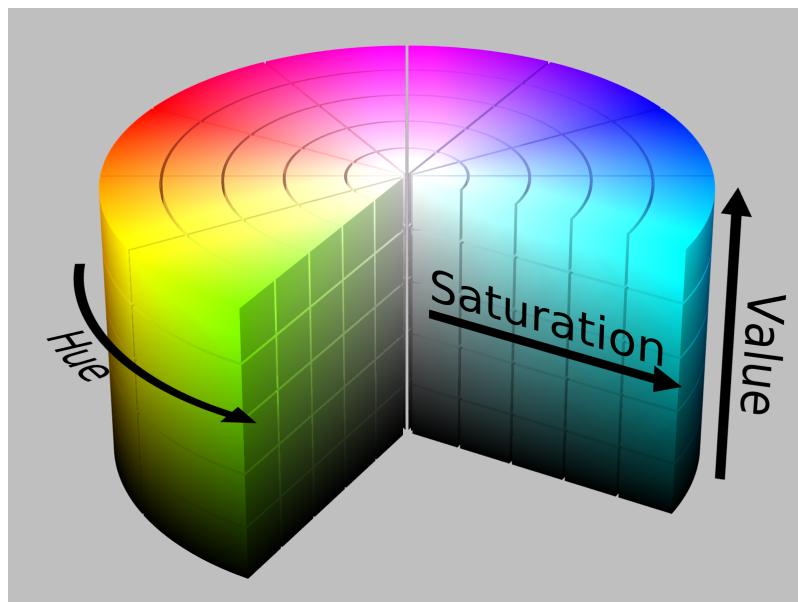


Рис. 3.14: Цветовое пространство HSV

На рисунке 3.15 показано, как меняется картинка из примера при умножении канала насыщенности на 1.5 в цветовом пространстве HSV. В результате цвета стали более сочными и насыщенными. Это выполняется при помощи следующего кода:

```
hsv = cv2.cvtColor(img1,
```

```

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

```



(a)

(b)

Рис. 3.15: Фото до (а) и после (б) умножения канала насыщенности на 1.5 в цветовом пространстве HSV

3.3.7. Каскады Хаара — детектор лиц

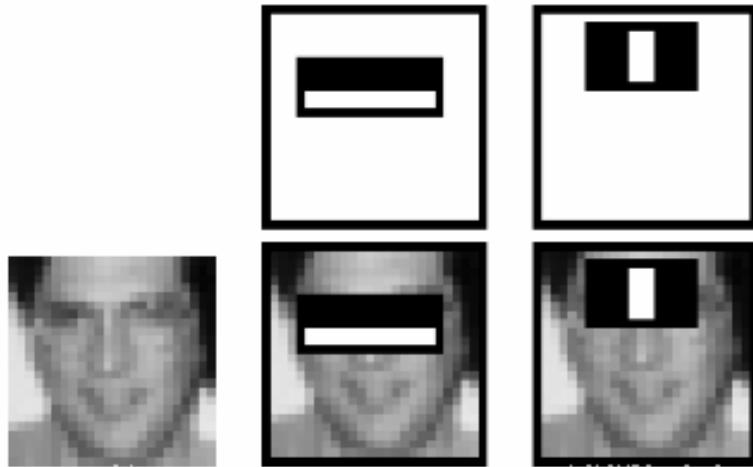


Рис. 3.16: Каскады Хаара

Одна из задач, решаемых наукой о компьютерном зрении, — это детекция лиц. Один из первых удачных методов решения — это каскады Хаара. Применяя этот метод, из картинки можно вычленять достаточно простые признаки. Для этого необходимо использовать несколько прямоугольников (рис. 3.16). Пиксели, попадающие в белый прямоугольник берутся со знаком плюс, в чёрный — со знаком минус. Все значения суммируются, и получается одно число. Выбор прямоугольников и коэффициентов для них осуществляется с помощью алгоритма AdaBoost. У лица имеются некоторые паттерны, и в итоге каскад такого рода фильтров показывает, есть ли внутри него лицо, или нет.

Сейчас созданы методы детекции лиц, превосходящие по качеству каскады Хаара. Тем не менее этот подход достаточно простой, и его легко найти готовым к использованию. Если не требуется решать задачу с высоким качеством, а получить детектор нужно быстро и просто, каскады Хаара из библиотеки opencv — это отличный вариант.

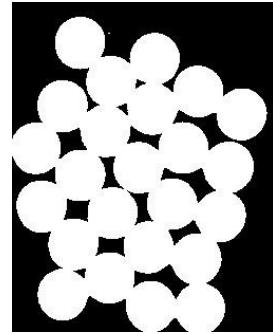
3.3.8. Сегментация

Задачу сегментации также можно достаточно просто решить. Один из способов — отрезать пиксели выше какого-то порога и назначить их объекту, а пиксели ниже порога — фону. Пример показан на рисунке 3.20. На этом изображении требуется разделить монеты и фон. Видно, что монеты намного темнее, поэтому достаточно подобрать такую границу, чтобы они все оказались ниже. Код из opencv, который позволяет это сделать:

```
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)
```



(a)



(b)

Рис. 3.17: Сегментация изображения

3.4. Линейная фильтрация изображений

Важный класс преобразования изображений — это линейные фильтры. С их помощью решаются задачи поиска границ, уголков, удаления шумов.

3.4.1. Скользящее среднее — свёртка

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

Рис. 3.18

Проще всего объяснить, что такое линейная фильтрация, на примере. Пусть требуется подсчитать среднее в окне 3×3 для каждого пикселя. Вычисление среднего можно записать в следующем виде:

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

Переписав его в следующем виде, можно получить выражение для свёртки:

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

где f — это изображение (двухмерная функция, характеризующая картинку), k, l — координаты пикселя, f — яркость пикселя, h — ядро свёртки (матрица 3×3 , состоящая из единиц, рис. 3.18). Если ядро свёртки — матрица, как на рисунке 3.18, то свёртка — это скользящее среднее.

В opencv произвести такую свёртку можно следующим образом:

```

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

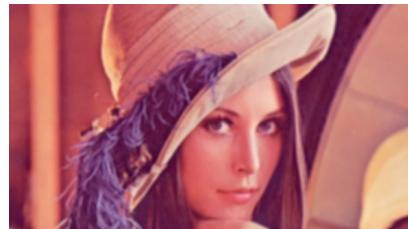
```

Картинка при этом становится более размытой (рис. 3.19). Также размытие изображения можно получать при помощи свёртки с гауссовской функцией (рис. 3.19):

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



(a)



(b)



(c)

Рис. 3.19: (a) — фото до, (b) — после применения скользящего среднего, (c) — после применения Гауссовского размытия

3.4.2. Детекция границ

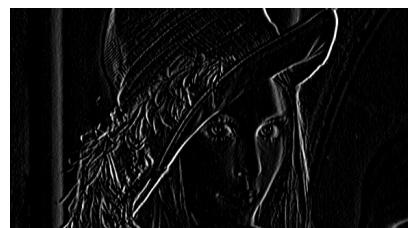
Свёртки также можно применять для детекции границ. С помощью свёрток 3.20a, 3.20b можно получить вертикальные и горизонтальные границы изображения (рис. 3.21a, 3.21b). Если объединить результаты этих двух свёрток, можно получить все границы (рис. 3.21c).

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

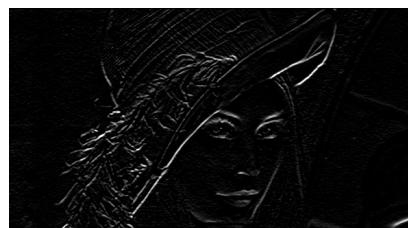
(a)

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

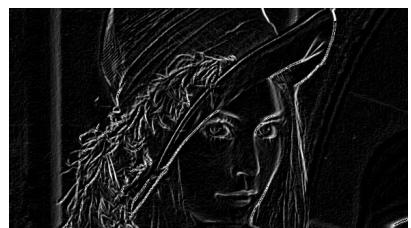
(b)



(a)



(b)



(c)

Рис. 3.21: Границы изображения, полученные с помощью применения свёртки. (a) — горизонтальные, (b) — вертикальные, (c) — все границы, полученные объединением результатов двух свёрток.

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.

3.4.3. Корреляция

Другой пример линейного преобразования — это корреляция. Она очень похожа на свёртку, но записывается немного в другом виде:

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

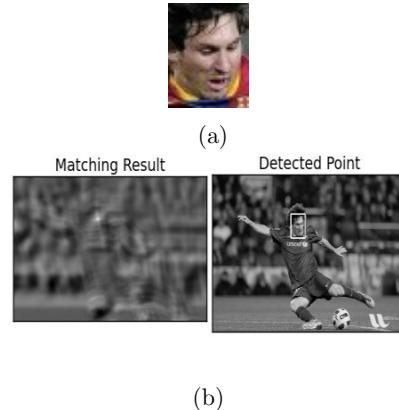


Рис. 3.22: (a) — изображение, которое требуется найти. (b) — слева — применение корреляции к картинке для поиска изображения, справа — картинка, содержащая искомое изображение

В отличии от свёртки, корреляция используется, чтобы показать меру похожести двух изображений. Это может быть использовано для поиска объектов. Например, требуется найти лицо футболиста (рис. 3.22a). На рисунке 3.22b слева показан результат применения корреляции для поиска лица. Белое пятно — это место, где оно найдено. Корреляцию можно использовать с различными параметрами: нормировать, применять её различные вариации.

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

3.4.4. Резюме

В этой части шла речь о линейной фильтрации. Позднее будет показано, что самые важные компоненты глубоких нейронных сетей состоят из свёрток, одного из способов линейной фильтрации. По этой причине важно получить понимание того, что такое свёртка, и для чего она используется. Далее речь пойдёт о том, как свёрточные нейронные сети помогают решать более сложные задачи компьютерного зрения.