Основные методы обработки изображений и их практическое применение

Московцев Андрей

План

1. Введение в обработку изображений
2. Базовые операции над изображениями
    1. Характеристики изображения
    2. Цветовые пространства
    3. Чтение/запись/отображение
    4. Изменение размера/обрезка
3. Фильтрация и улучшение изображений
    1. Сглаживание и размытие (Averaging, Gaussian, Median Blur)
    2. Детектирование краёв (Sobel, Canny Edge detector)
    3. Пороговая фильтрация (Binary, Otsu’s method)
    4. Морфологические операции (Erosion, Dilation, Opening, Closing)
4. Нахождение и извлечение признаков
    1. Contour Detection (cv2.findContours)
    2. Corner Detection (Harris Corner, Shi-Tomasi)
    3. Key points & descriptors (SIFT, SURF, ORB)
5. Настройка виртуального окружения и установка библиотек
6. Практическая часть

Введение в обработку изображений

Изображение – это просто матрица

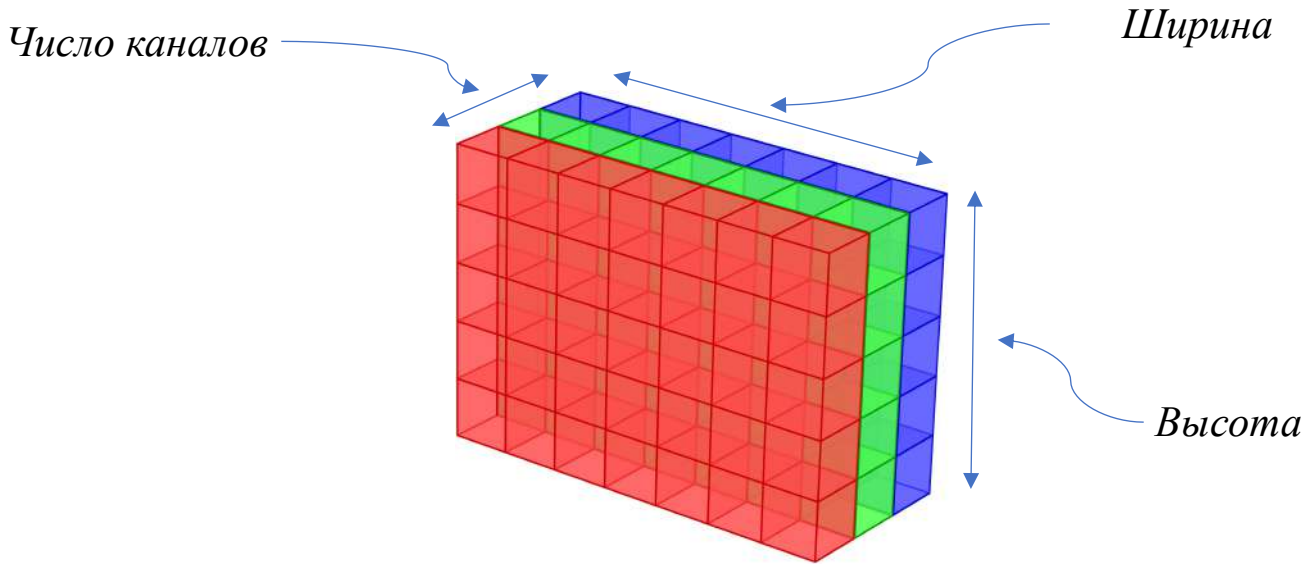
Изображение – по сути трёхмерная матрица с размерностью (N, Height, Width)

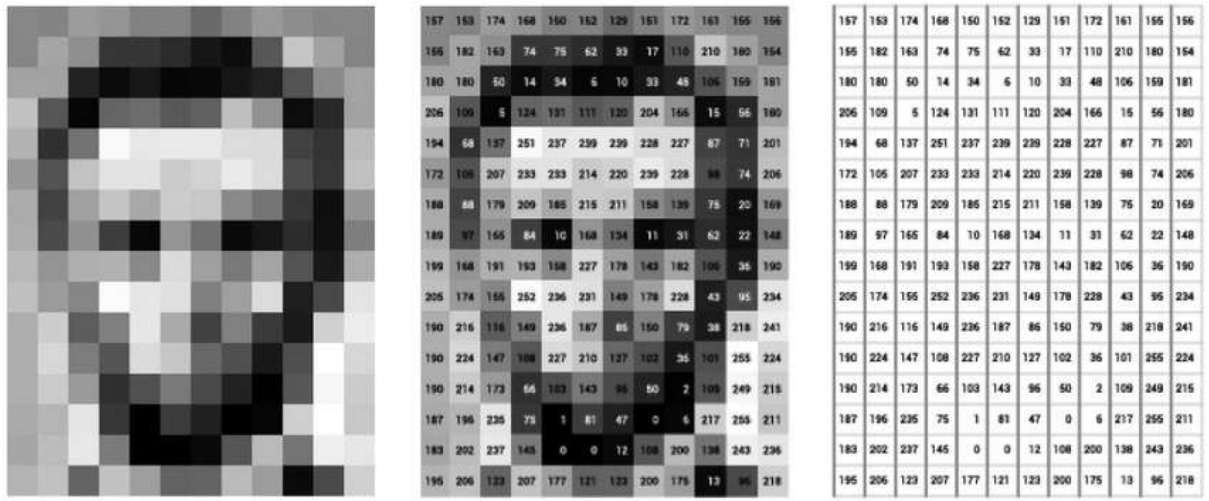
Элементы матрицы – значения ”интенсивности” пикселей, которые изменяются в диапазоне [0;255]

RBG изображение: (3, Height, Width)

Grayscale изображение: (1, Height, Width)

Обработка изображений = проведение операций пиксельными данными изображений для извлечения признаков или улучшения качества изображений

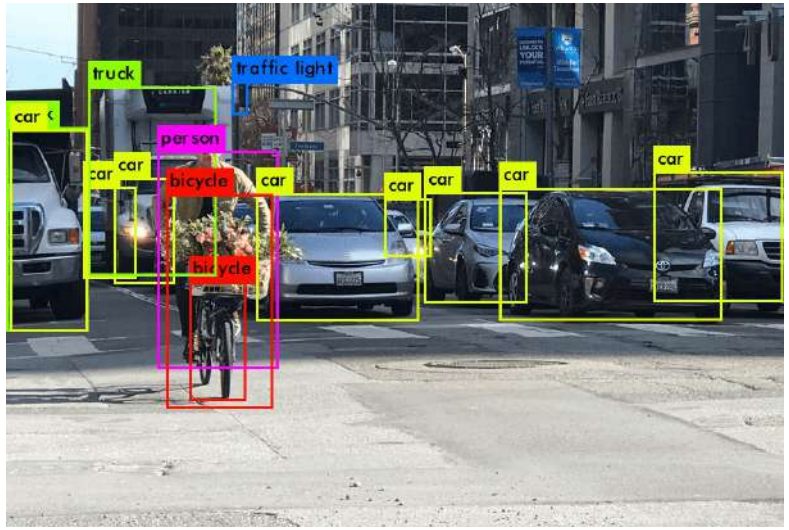


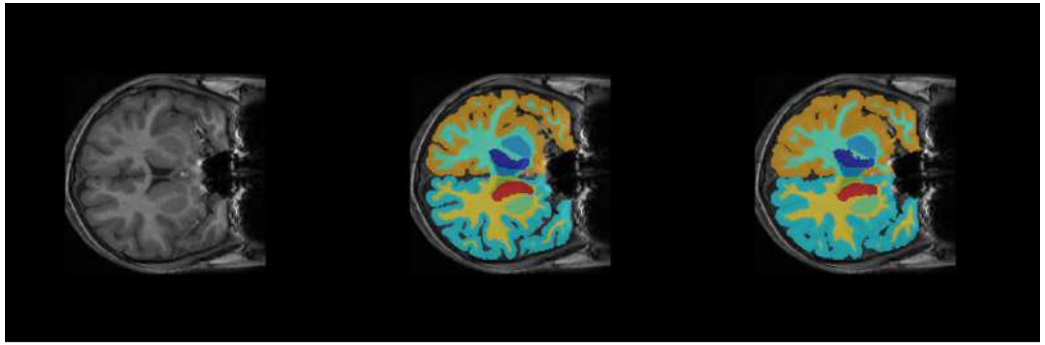


Области применения

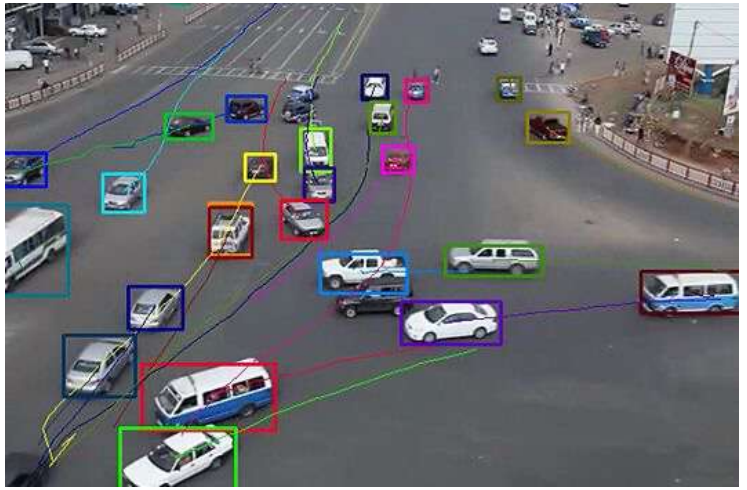
1. Детектирование и распознавание лиц (Face Detection and Recognition) – автоматическое распознавание людей в толпе
2. Автономное вождение (Self-driving cars) – Tesla и др.
3. Анализ медицинских снимков (Medical Imaging) – автоматическое обнаружение опухолей на медицинских снимках
4. Сканирование документов и распознавание символов (Document Scanning and OCR) – автоматическая обработка документов
5. Контроль качества продукции (Industrial Inspection) – дефектоскопия на производственных линиях
6. Виртуальная реальность (Augmented Reality) – Pokemon Go
7. Видеонаблюдение и трекинг (Video Surveillance and Tracking) – системы безопасности, выявление подозрительной активности
8. Анализ спутниковых и аэрофотоснимков (Satellite and Image Analysis) – анализ сельскохозяйственной продукции
9. Генерация изображений (Image generation) – создание контента, DALLE, MidJourney, Kandinski

Примеры





2-D Slice from Input Volume Predicted Segmentation Map Ground Truth



Что такое OpenCV?

- Открытая библиотека для обработки изображений и задач компьютерного зрения;
- Разработана Intel в 1999
- Поддержка C++, python, Java;
- Более 2500 алгоритмов для работы с изображениями:
  - детектирование и распознавание лиц;
  - детекция объектов;
  - трекинг движений объектов;
  - нахождение похожих снимков по имеющимся в базе данных;
  - создание облака точек
  - …

[официальная документация]

Традиционное компьютерное зрение vs глубокое обучение

Traditional CV	**Deep Learning**
Manual feature design	Learns features automatically
Works with few images	Needs large datasets
Fast and lightweight	Requires GPU’s / heavy compute
Interpretable	Often a black box
Struggles with complex variations	Handles complexity well

Базовые операции над изображениями

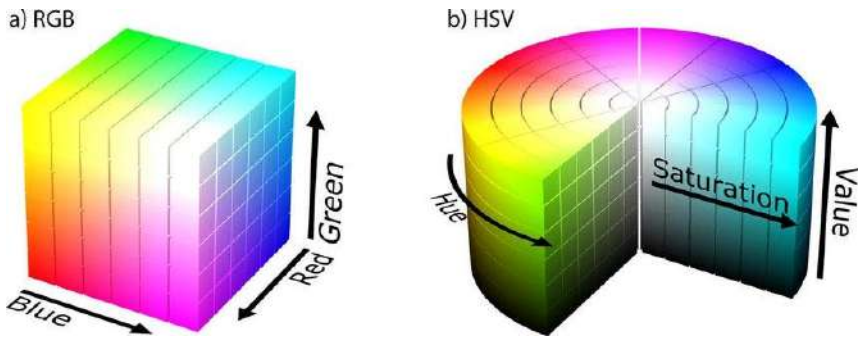
Базовая структура кода

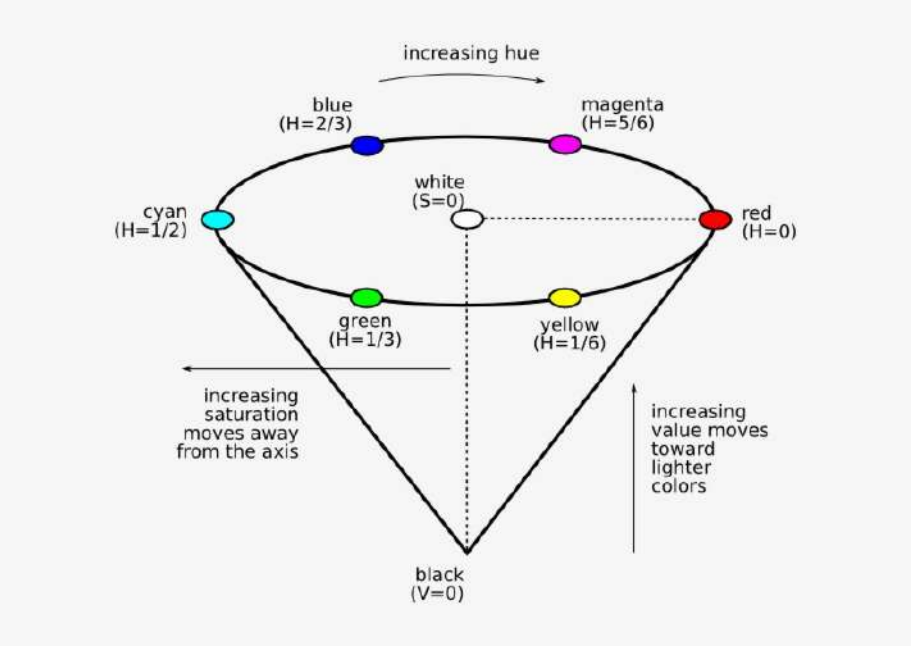
```
import cv2

# Read an image
img = cv2.imread("image.jpg")   # Returns a NumPy array
print(type(img))    # <class 'numpy.ndarray'>
# Save image
cv2.imwrite("copy.jpg", img)
```

Note: OpenCV reads images in BGR order by default.

Цветовые пространства

1. **Grayscale** (оттенки серого)
   - Изображение без цвета, только яркость
   - Один канал: значения от 0 (чёрный) до 255 (белый)
   - Упрощает вычисления, зачастую используется перед обработкой изображений (пороговая фильтрация, выделение контуров)
2. **RGB** (Red, Green, Blue)
   - Самое распространённое цветовое пространство
   - Каждому пикселю сопоставлены три компоненты: красная, зелёная и синяя
   - Используется для отображения на экранах
   - В OpenCV по умолчанию используется BGR, а не RGB
3. **HSV** (Hue, Saturation, Value)
   - Отделяет информацию о **цвете (Hue)** от **насыщенности (Saturation)** и **яркости (Value)**;
   - **Hue** (тип цвета, напр. красный, циан) – измеряется в градусах, в OpenCV отмасштабирован под [0, 180]
   - **Saturation** (насыщенность, интенсивность) – описывает чистоту/приглушённость цвета. 0 – полностью приглушён (серый), 255 – максимальная интенсивность цвета
   - **Value** (яркость) – описывает, насколько ярким или тёмным является цвет. 0 – полностью чёрный, 255 – максимальная яркость цвета
   - Удобен для задач, где важно работать с конкретными цветами, например выделение цвета по объекту





Свойства изображений и цветовые пространства

Image properties

```
img.shape # (height, width, channels)
img.dtype # usually uint8
img.size # total number of pixels

# grayscale image: shape (H,W)
# color image (BGR): shape = (H, W, 3)
```

Color space

```
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)

# BGR – default in OpenCV
# RGB – used in matplotlib
# Grayscale – intensity only
# HSV – better for color based filtering
```

Отображение картинок

С помощью OpenCV (например, если работаем через IDE):

```
cv2.imshow("My Image", img)
cv2.waitKey(0) # Wait for a key press
cv2.destroyAllWindows()
```

С помощью matplotlib (если работаем в jupyter-ноутбуках):

```
import matplotlib.pyplot as plt
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB)) # Convert BGR → RGB
plt.axis('off')
plt.show()
```

Note:

- cv2.imshow() requires GUI backend; doesn't work in some notebook environments (use matplotlib instead).
- cv2.waitKey() is essential — otherwise, the window will not display or will close immediately

Изменение размера, обрезка, поворот

```
resized = cv2.resize(img, (300, 300)) # Resize to exact dimensions
resized_ratio = cv2.resize(img, (0, 0), fx=0.5, fy=0.5) # Keep aspect ratio
```

```
cropped = img[100:300, 200:400] # [y1:y2, x1:x2]
cv2.imshow("Cropped", cropped)
```

```
(h, w) = img.shape[:2]
center = (w // 2, h // 2)
M = cv2.getRotationMatrix2D(center, angle=45, scale=1.0)
rotated = cv2.warpAffine(img, M, (w, h))
```

Фильтрация и улучшение изображений

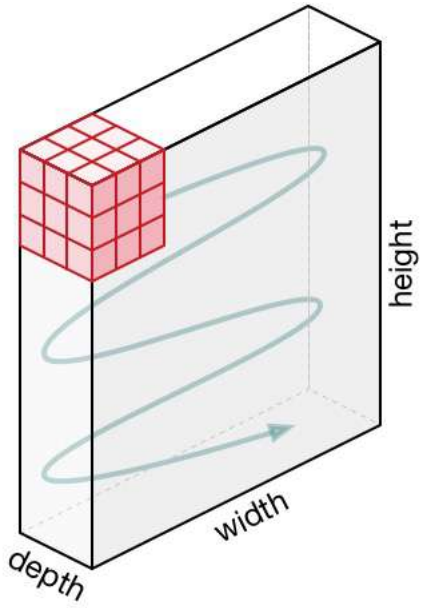
Операция свёртки при работе с изображениями

исходное изображение

$$g[i,j] = \sum_{m=1}^{M}\sum_{n=1}^{N} f[m,n] \cdot h[i-m, j-n]$$

Итоговый результат

ядро свёртки



0	0	0	0	0	0	...
0	156	155	156	158	158	...
0	153	154	157	159	159	...
0	149	151	155	158	159	...
0	146	146	149	153	158	...
0	145	143	143	148	158	...
...	...	...	...	...	...	...

Input Channel #1 (Red)

0	0	0	0	0	0	...
0	167	166	167	169	169	...
0	164	165	168	170	170	...
0	160	162	166	169	170	...
0	156	156	159	163	168	...
0	155	153	153	158	168	...
...	...	...	...	...	...	...

Input Channel #2 (Green)

0	0	0	0	0	0	...
0	163	162	163	165	165	...
0	160	161	164	166	166	...
0	156	158	162	165	166	...
0	155	155	158	162	167	...
0	154	152	152	157	167	...
...	...	...	...	...	...	...

Input Channel #3 (Blue)

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

Kernel Channel #1

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

Kernel Channel #2

0	1	1
0	1	0
1	-1	1

Kernel Channel #3

308 + −498 + 164 + 1 = −25

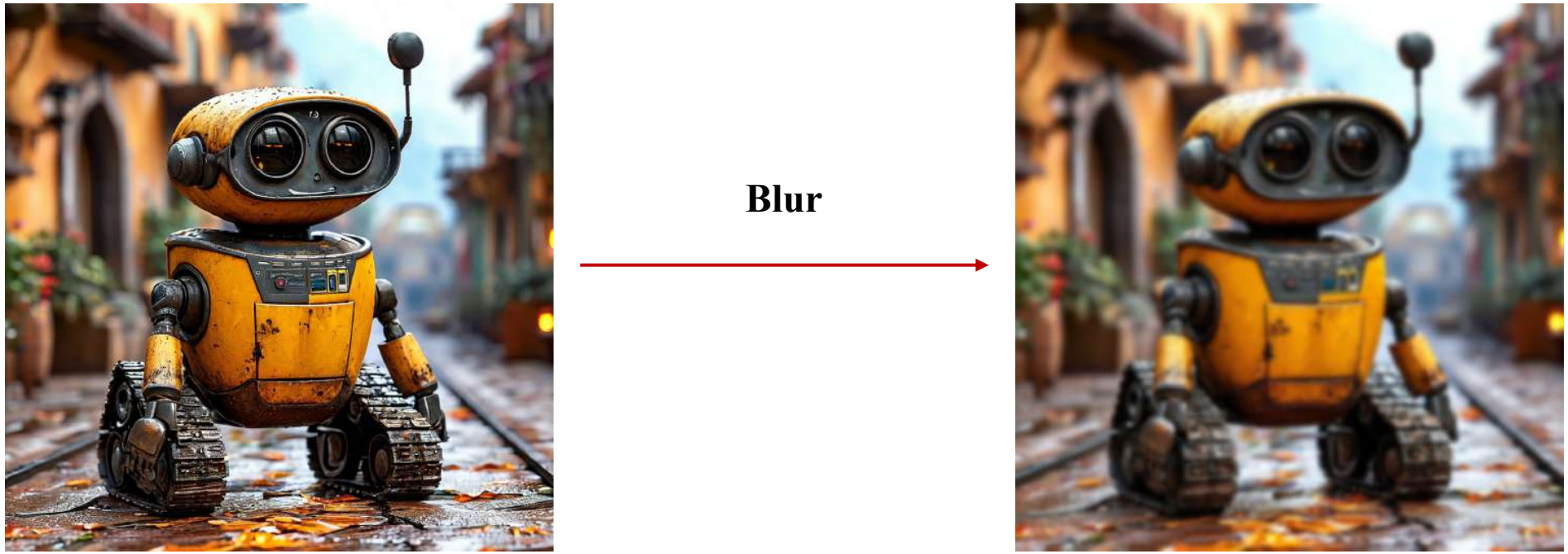
Bias = 1

Output

-25				...
				...
				...
				...
...	...	...	...	...

Размытие изображений

Размытие (blurring) - распространенная техника обработки изображений, используемая для уменьшения шума или деталей в изображении. Обычно это достигается путем усреднения значений пикселей в окрестностях каждого пикселя, что создает эффект плавности. Размытие часто используется в предварительной обработке для таких задач, как обнаружение краев или сегментация, чтобы уменьшить нежелательные детали или шум.



Техники размытия изображений

1. Размытие усреднением (Average Blurring)

Метод: вычисление средней величины интенсивности пикселей внутри ядра свёртки. Полученное значение присваивается центральному пикселю.

Эффект: простое сглаживание с единым весом для всех пикселей в ядре

2. Медианное размытие (Median Blurring)

Метод: заменяет значение центрального пикселя на медиану значений пикселей в ядре.

Эффект: эффективно удаляет ”salt-and-pepper” шум на изображении, сохраняя целыми края

3. Гауссово размытие (Gaussian Blurring)

Метод: применение Гауссова ядро свёртки к изображению, давая больший вес центральным пикселям, и меньший вес – удалённым от центра.

Эффект: более естественное и плавное размытие по сравнению с Average Blurring

Пример использования: часто используется от избавления от Гауссовского шума

4. Билатеральный фильтр (Bilateral Blurring)

Метод: применение двух Гауссовых ядер свёртки к изображению – по положению в пространстве (удалённость от центрального пикселя) и по интенсивности (отличие от центрального пикселя)

Эффект: сглаживает однородные области, сохраняя чёткими края

Пример использования: задачи, где необходимо сохранять чёткими края изображения

Билатеральный фильтр. Формула

$$g[i,j] = \frac{1}{W_{mn}} \sum_m \sum_n f[m,n] \cdot n_{\sigma_s}[i-m, j-n] \cdot n_{\sigma_b}(f[m,n] - f[i,j])$$

$$n_{\sigma_s}[m,n] = \frac{1}{2\pi\sigma_s^2} e^{-\frac{1}{2}\left(\frac{m^2+n^2}{\sigma_s^2}\right)}$$

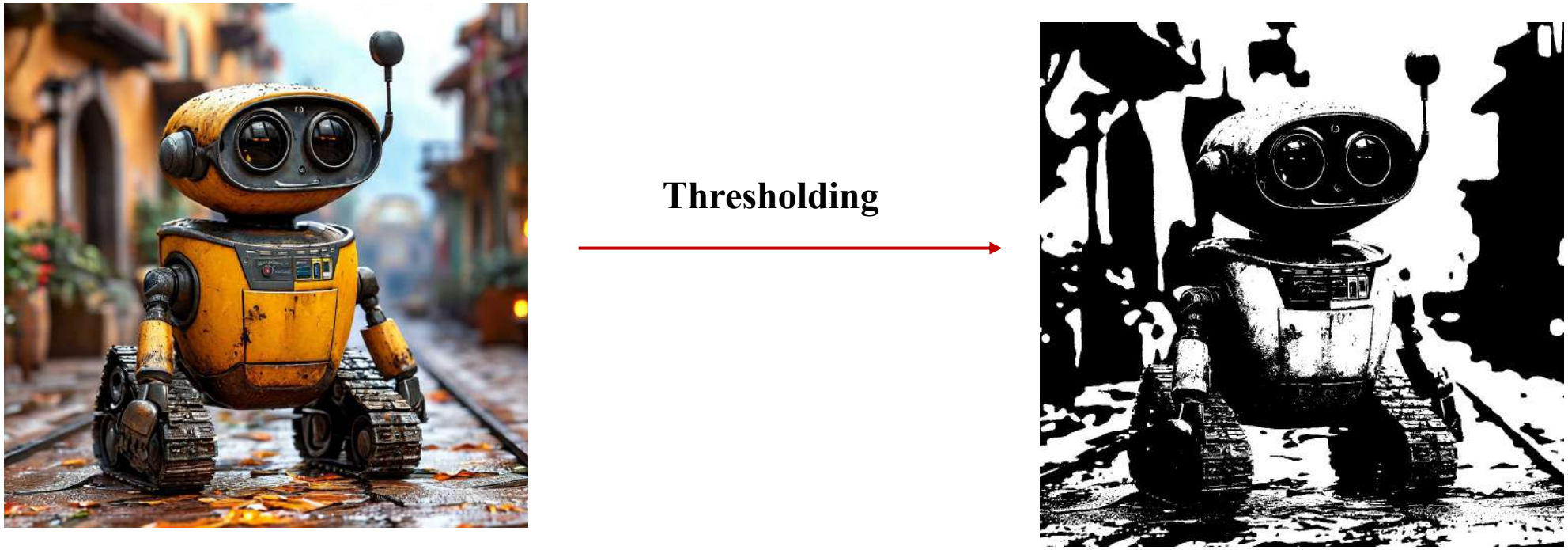
пространственный Гауссиан

$$n_{\sigma_b}(k) = \frac{1}{2\pi\sigma_b^2} e^{-\frac{1}{2}\left(\frac{k^2}{\sigma_b^2}\right)}$$

яркостный Гауссиан

Пороговая фильтрация

Пороговая фильтрация (thresholding) — это базовый метод обработки изображений, используемый для разделения пикселей на две группы: те, которые соответствуют заданному порогу, и те, которые ему не соответствуют. Обычно пороговая фильтрация применяется к **градациям серого** (одноканальным изображениям), преобразуя изображение в бинарное (чёрно-белое).



Основной принцип пороговой фильтрации

Каждому пикселю изображения назначается новое изображение на основе его интенсивности:

- Если значение пикселя больше или равно заданному порогу, то оно изменяется на значение верхнего порога (обычно выбирают максимально возможное значение – **255** (белый цвет))
- В противном случае пиксель принимает значение нижнего порога (выбирают минимально возможное значение – **0** (чёрный цвет))

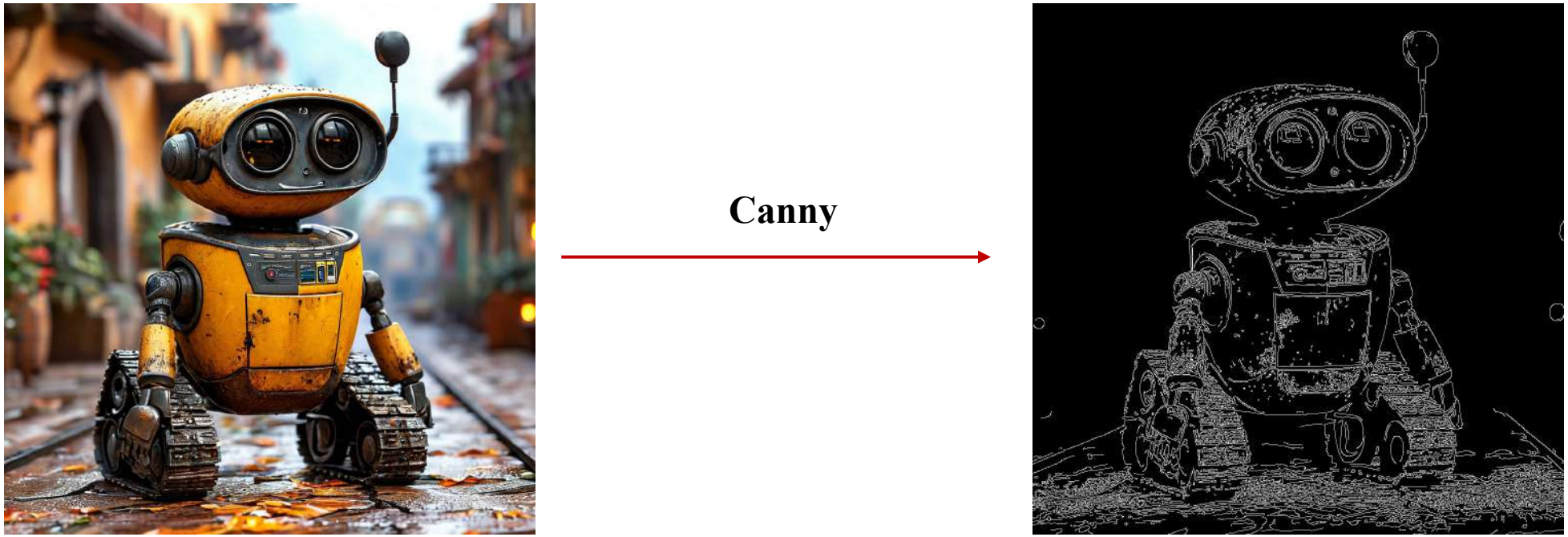
Математическая формула:

$$\mathrm{dst}(x, y) = \begin{cases} \mathrm{maxVal}, \text{если } \mathrm{src}(x, y) \geq T \\ 0, \text{если } \mathrm{src}(x, y) < T \end{cases}$$

где T – порог, maxVal – значение для белого цвета

Детектирование краёв: алгоритм Canny

Алгоритм Кэнни (Canny Edge Detection) - это популярный метод обнаружения краёв на изображении. Он был предложен Джоном Кэнни в 1986 году и до сих пор используется благодаря своей эффективности и точности. Алгоритм выполняется в несколько этапов.



Этапы алгоритма Canny

1. **Сглаживание изображения:** перед обработкой изображение сглаживается с помощью Гауссова размытия для снижения шума, который может приводить к ложным краям
2. **Выделение градиентов:** алгоритм вычисляет градиенты яркости (изменение интенсивности) с помощью операторов Собеля
3. **Немаксимальное подавление (Non-Maximum Supression)**: убираются пиксели, которые не являются локальными максимумами вдоль направления градиента
4. **Двойной порог:** для определения значимости границ применяются два порога
   - **Высокий порог:** пиксели, градиенты которых выше этого значения, считаются достоверными краями
   - **Низкий порог:** пиксели, градиенты которых между низким и высоким порогом, считаются потенциальными краями
5. **Соединение краёв (Edge Tracking by Hysteresis):** пиксели, прошедшие высокий порог, соединяются с пикселями, прошедшими низкий порог, если они связаны

Вычисление градиента:

$$G_x = \frac{\partial I}{\partial x} \qquad G_y = \frac{\partial I}{\partial y}$$

Величина градиента:

$$G = \sqrt{G_x^2 + G_y^2}$$

Угол направления:

$$\theta = \tan^{-1}\left(\frac{G_y}{G_x}\right)$$

Морфологические преобразования

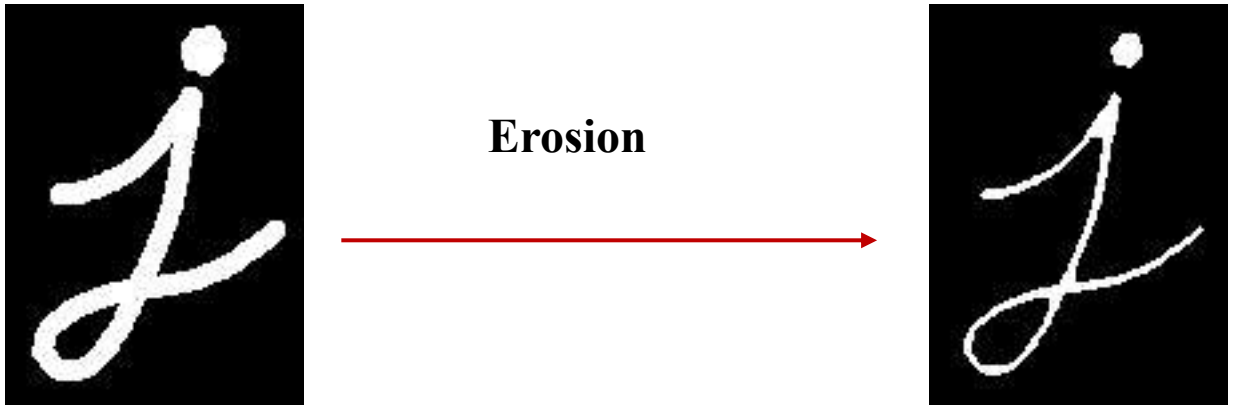
Морфологические преобразования – операции, основанные на изменении формы изображений. Обычно они выполняются на бинарных изображениях. Структурный элемент или ядро задаваемой конфигурации применяется к изображению для изменения его геометрии – в особенности границ.

Особенно полезны после применения пороговой фильтрации или детектирования краёв для:

- Удаления шума на изображениях;
- Заполнения отверстий в фигурах;
- Отделения связных компонент

Erosion

- The kernel slides through the image (as in 2D convolution). A pixel in the original image (either 1 or 0) will be considered 1 only if all the pixels under the kernel is 1, otherwise it is eroded (made to zero).
- So what happends is that, all the pixels near boundary will be discarded depending upon the size of kernel. So the thickness or size of the foreground object decreases or simply white region decreases in the image.
- It is useful for removing small white noises (as we have seen in colorspace chapter), detach two connected objects etc.




```
import cv2
import numpy as np
img = cv.imread('j.png', cv.IMREAD_GRAYSCALE)
kernel = np.ones((5,5),np.uint8)
erosion = cv.erode(img,kernel,iterations = 1)
```

Dilation

It is just opposite of erosion. Here, a pixel element is '1' if at least one pixel under the kernel is '1'. So it increases the white region in the image or size of foreground object increases. Normally, in cases like noise removal, erosion is followed by dilation. Because, erosion removes white noises, but it also shrinks our object. So we dilate it. Since noise is gone, they won't come back, but our object area increases.

It is also useful in joining broken parts of an object.

```
dilation = cv2.dilate(img,kernel,iterations = 1)
```

Opening

Opening is just another name of **erosion followed by dilation**. It is useful in removing noise, as we explained above. Here we use the function, **cv.morphologyEx()**

```
opening = cv2.morphologyEx(img, cv.MORPH_OPEN, kernel)
```

Closing

Closing is reverse of Opening, **Dilation followed by Erosion**.

It is useful in closing small holes inside the foreground objects, or small black points on the object.

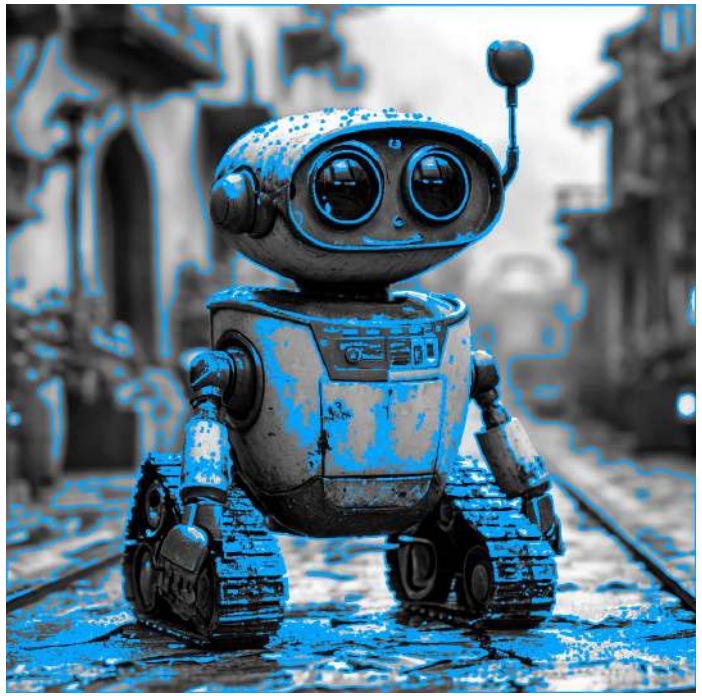
```
closing = cv.morphologyEx(img, cv.MORPH_CLOSE, kernel)
```

Нахождение и извлечение признаков

Нахождение контуров изображения

Контуры представляют собой линии, соединяющие последовательные точки одного и того же цвета или интенсивности. В компьютерном зрении они используются для анализа форм объектов, сегментации изображений и других задач. Библиотека OpenCV предоставляет удобный метод cv2.findContours для обнаружения контуров на бинарных изображениях.

findContours



Полезные ссылки

1. Ноутбук с материалами занятия: [ссылка]
2. Шпаргалка по используемым командам: [ссылка]
3. Полезный блог по теме обработки изображений: [ссылка]

Задание

1. Склонируйте учебный репозиторий. Для этого откройте терминал и введите команду:
   git clone https://github.com/mskv99/learn-opencv.git
2. Загрузите изображение ”ducks.jpeg” из папки “images” учебного репозитория
3. Выполните следующие шаги:
   1. Примените пороговую фильтрацию для выделения объектов
   2. Найдите контуры объектов
   3. Нарисуйте найденные контуры на исходном изображении
   4. Отобразить полученный результат
4. Получите два максимальных по длине (периметру) контура. Отобразите только их на изображении. В центре каждого из двух контуров подпишите его длину в виде ”Perimeter: number”
5. Создайте итоговое изображение, где к каждому объекту будет подписана его площадь

Итог

1. **Основы работы с изображениями**: изображения представлены как матрицы, где нулевое измерение отражает цветовые каналы, а первые два — размеры изображения.

2. **Ключевые операции обработки изображений**:

- **Пороговая фильтрация** позволяет выделять объекты, преобразуя изображение в бинарное.
- **Размытие** (среднее, гауссово, медианное, билатеральное) используется для снижения шума и сглаживания.
- **Алгоритм Canny** эффективен для обнаружения границ.

3. **Работа с контурами**: нахождение и анализ контуров помогает сегментировать объекты и получать их характеристики (площадь, периметр).

4. **Практика с OpenCV**: студенты освоили базовые операции, включая чтение, запись, отображение изображений и применение фильтров.

5. **Навыки применения**: полученные знания и инструменты создают основу для решения практических задач в области компьютерного зрения, таких как сегментация, классификация и анализ объектов.