

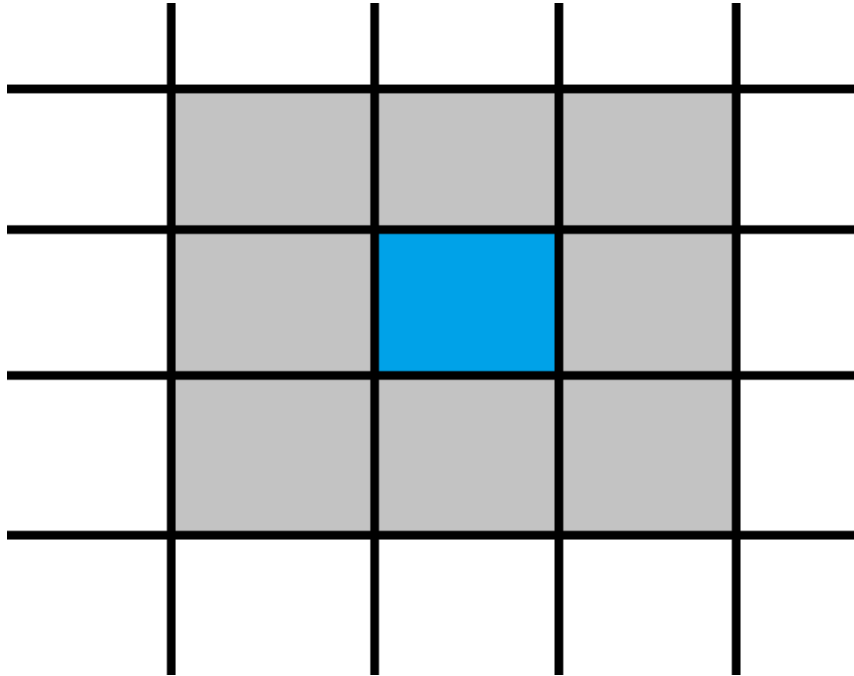
Основы работы с данными для ИИ

Лекция 10 Обработка изображений

Доц
Екатерина Александровна

Преподаватель кафедры информатики и вычислительного эксперимента

Что такое цифровое изображение

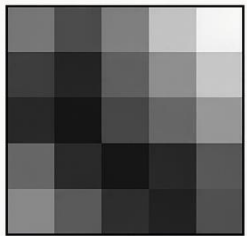


Цифровое изображение — это не «картинка» в привычном смысле, а множество маленьких элементов — пикселей.

Каждый пиксель имеет цвет, который можно записать числом или несколькими числами.

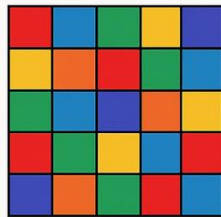
Если представить изображение как таблицу, то каждая строка и столбец — это пиксели. В Python мы будем работать с изображением как с обычным массивом (матрицей) чисел.

Чёрно-белое (градации серого)



Массив размера:
высота × ширина
В каждой ячейке —
яркость (например, 0.25)

Цветное (RGB)



Массив: высота
× ширина × 3
3 числа на пиксель
R, G, B

Градации серого:

- По сути это одно число, описывающее, насколько пиксель светлый или тёмный.
- Обычно 0 — чёрный, 255 — белый, промежуточные — оттенки серого.
- Математически: $I[y, x]$ — яркость в точке (x, y) .

Цветное изображение (RGB):

- Модель RGB основана на том, как формируется цвет в мониторах: светодиоды трёх цветов — красный, зелёный, синий.
- Каждый пиксель имеет три компоненты: $R[y, x]$, $G[y, x]$, $B[y, x]$.
- В памяти это либо «три слоя» (три матрицы), либо один трёхмерный массив.

На практике:

- многие алгоритмы сначала переводят цветное изображение в оттенки серого, чтобы работать с одним каналом;
- если нужен цветовой анализ (поиск по цвету, сегментация), используют отдельные каналы или другие цветовые модели

```
In [6]: img = np.array(img)
img
Out[6]: array([[ 0,  0, 31],
               [ 0,  0, 32],
               [ 0,  0, 33],
               ...,
               [ 4,  0, 34],
               [ 3,  0, 32],
               [ 2,  0, 31]],

              [[ 0,  0, 31],
               [ 0,  0, 32],
               [ 0,  0, 33],
               ...,
```

Чаще всего изображение хранится в виде чисел от 0 до 255 — это тип `uint8`.

При обработке нам иногда удобнее перевести его в тип с плавающей точкой (`float`) и нормировать значения к $[0, 1]$, чтобы корректно применять формулы, не получить переполнения и т.д.

Это техническая деталь, но её полезно понимать: картинка — это не только форма и цвет, но и конкретные числовые типы в памяти

Типы данных: 0–255 и вещественные числа

В «сырых» изображениях пиксели чаще всего хранятся как **целые числа без знака на 8 бит**:

1 байт → 256 возможных значений (0–255).

Это удобно для хранения, экономит память, поддерживается многими форматами (JPEG, PNG).

Для обработки:

- часто переводим данные в тип float32 или float64,
- нормируем значения от 0 до 1: $\text{new} = \text{old} / 255.0$.

Почему так делают:

- легче писать формулы (например, умножение на 1.2 не приведёт к переполнению),
- проще контролировать диапазон,
- многие алгоритмы в библиотеках ожидают именно вещественные числа.

Если на uint8 добавить 10 к 250, получится не 260 (его быть не может), а 4 (переполнение по модулю 256).



Библиотека Python Pillow является ответвлением более старой библиотеки под названием PIL. PIL расшифровывается как Python Imaging Library, и она позволяла Python работать с изображениями. PIL приостановила работу в 2011 году и поддерживается только Python 2. Если использовать описание разработчиков, то Pillow — это удобный "переходник" PIL, который поддерживает жизнь библиотеки и включает поддержку Python 3

В Python есть много модулей для работы с изображениями и их обработки. Если вы хотите работать с изображениями напрямую, оперируя их пикселями, вы можете использовать NumPy и SciPy. Другими популярными библиотеками для обработки изображений являются OpenCV, scikit-image и Mahotas. Некоторые из этих библиотек быстрее и мощнее, чем Pillow. Однако Pillow остается важным инструментом для работы с изображениями. Она предоставляет функции обработки, аналогичные тем, которые можно найти в программном обеспечении Photoshop. Pillow часто является предпочтительным вариантом для задач обработки изображений высокого уровня, которые не требуют более продвинутых навыков обработки.

NumPy:

стандарт де-факто для научных вычислений в Python;
предоставляет многомерные массивы и быстрые операции над ними;
все серьёзные библиотеки обработки изображений под капотом опираются на NumPy.

Pillow:

удобна для «бытовых» вещей: загрузить картинку из файла, изменить размер, повернуть, сохранить в другом формате;
хорошо подходит для демонстраций и простых задач.

OpenCV:

возникла как библиотека компьютерного зрения на C++, но есть официальный Python-интерфейс;
умеет: фильтрацию, поиск контуров, детектирование объектов, работу с видео, камерами и др.;
немного «низкоуровневая» по интерфейсу, но очень мощная.

matplotlib:

не обрабатывает изображения, а **показывает** их;
полезна для отладки: посмотреть результат каждого шага.

В Python есть много модулей для работы с изображениями и их обработки. Если вы хотите работать с изображениями напрямую, оперируя их пикселями, вы можете использовать NumPy и SciPy. Другими популярными библиотеками для обработки изображений являются OpenCV, scikit-image и Mahotas. Некоторые из этих библиотек быстрее и мощнее, чем Pillow. Однако Pillow остается важным инструментом для работы с изображениями. Она предоставляет функции обработки, аналогичные тем, которые можно найти в программном обеспечении Photoshop. Pillow часто является предпочтительным вариантом для задач обработки изображений высокого уровня, которые не требуют более продвинутых навыков обработки.

```
from PIL import Image
import matplotlib.pyplot as plt
import numpy as np

img = Image.open("image.jpg")          # открыть файл
plt.imshow(img)                        # показать
plt.axis("off")
plt.show()

img_array = np.array(img)              # в массив
```

- Изменение размера (масштабирование)
- Поворот
- Обрезка (crop)
- Отражение (flip)



Геометрические преобразования. Основные операции



Все эти операции можно рассматривать как **преобразование координат**: для каждого пикселя выходного изображения вычисляем координаты соответствующего пикселя во входном; при этом могут понадобиться разные формы интерполяции, если целые координаты не совпадают.

Масштабирование:

уменьшение — нужно, чтобы экономить ресурсы, приводить разные изображения к одному размеру;
увеличение — используется реже, может приводить к размытию.

Поворот:

поворот на 90° можно сделать просто перестановкой индексов массива;
произвольный угол требует интерполяции и «растягивания» изображения.

Обрезка:

на уровне массива — это просто взятие подматрицы по диапазону индексов.

Отражение:

по горизонтали или вертикали — это замена порядка индексов по одной из осей.
В библиотеках всё это делается одной строчкой, но за этим стоит именно работа с координатами.

Яркость и контраст

Яркость: «сдвиг» значений

Контраст: «растяжение» значений

Простейшие формулы:

$$\text{new} = \text{old} + b$$

$$\text{new} = a * \text{old}$$

Вывод изображений с matplotlib

```
In [3]: def show(img, title=None):  
        plt.figure(figsize=(6, 6))  
        plt.imshow(img)  
  
        if title is not None:  
            plt.title(title)  
  
        plt.axis('off')
```

Функция принимает два параметра: `img` — массив, который вы получили из исходной фотографии, и `title` — заголовок, который по умолчанию имеет значение `None`. Размер фигуры, где выводим изображение, устанавливаем как 6×6. Заголовок изображения можно установить с помощью функции `title()`. Выключите пометки на осях, указав `'off'` в функции `axis()`.

Как сделать изображение черно-белым

```
In [4]: def to_black(img):  
        return img.convert('L')
```

Мы представили изображение в RGB-формате, то есть каждый пиксель имеет три значения: красный (R), зеленый (G) и синий (B). Оттенки серого получаются, когда все эти три значения равны. Есть известная формула яркости пикселя:

$$Y = 0.299R + 0.587G + 0.114B$$

Но Pillow использует свои коэффициенты

Переход в оттенки серого

Человек по-разному воспринимает разные цвета:

к зелёному глаз более чувствителен, к синему — менее.

Поэтому простое среднее $(R + G + B) / 3$ не всегда даёт визуально правильную яркость.

В телевизионных и графических стандартах используются взвешенные коэффициенты (например, как на слайде).

Переход в оттенки серого:

уменьшает объём данных (1 канал вместо 3),

упрощает многие алгоритмы (поиск контуров, бинаризация),

но теряет информацию о цвете.

В коде это опять же сводится к операции над массивом: линейная комбинация трёх каналов.

Как добавить фильтр «Негатив»

Значения цветов могут меняться от 0 до 255. Негатив — эффект инвертирования цветов.

Достигается он просто: от 255 нужно отнять значение цвета. Так как все операции над массивом NumPy выполняются поэлементно, то мы отнимем от 255 текущий массив:

```
In [8]: def to_negative(img):  
        return 255 - img
```

С помощью функции `show()` оценим результат:



Как отзеркалить изображение

Следующая задача — отражение изображения слева направо (т. е. правая и левая стороны изображения меняются местами). В NumPy есть функция *fliplr*.

```
In [10]: def reverse(img):  
         return np.fliplr(img)
```

Этот фрагмент кода не изменяет массив, а возвращает новый, что и нужно. Результат работы:



Как добавить эффект

Идея этого эффекта проста — нужно сделать изображение как можно синее. Для этого замените каждое третье «значение пикселя» на максимальное — 255.

```
In [12]: def blue(img):  
         arr = np.copy(img)  
         arr[..., 2] = 255  
         return arr
```

Здесь нужно поменять сам массив, и, чтобы не потерять исходное изображение, скопируйте последовательность с помощью `np.copy()`.

Другой вариант конструкции:

```
arr[:, :, 2] = 255
```

Выводим результат:

Blue effect



Размытие (сглаживание) и шум

Реальные изображения почти всегда содержат шум:

электронный шум матрицы камеры,

сжатие,

дефекты освещения.

Сглаживание (blur):

применяем к каждому пикселю «окно» (например, 3×3),

новое значение — среднее/взвешенное среднее соседних пикселей.

Примеры фильтров:

усредняющий (mean filter) — простой, но может смазывать границы;

гауссов — использует веса по нормальному распределению, даёт более мягкое размывание.

Важно подчеркнуть компромисс:

сильное сглаживание убирает шум, но вместе с ним и детали;

обычно подбирают параметры фильтра экспериментально.

```
import cv2
import matplotlib.pyplot as plt

# 1. Читаем изображение с диска
img_bgr = cv2.imread("image.jpg") # считывается в формате BGR
img = cv2.cvtColor(img_bgr, cv2.COLOR_BGR2RGB) # переводим в RGB для
корректного отображения

# 2. Применяем сглаживание (размытие)

# Простое усредняющее размытие 5×5
blur_mean = cv2.blur(img, (5, 5))

# Гауссово размытие 5×5
blur_gauss = cv2.GaussianBlur(img, (5, 5), 0)
```

`cv2.blur` — простой фильтр: каждый пиксель заменяется средним значением в окне 5×5 вокруг него.

`cv2.GaussianBlur` — более «умный» фильтр: ближние пиксели учитываются сильнее, дальние слабее (по гауссовому закону).

Оба варианта уменьшают шум, но одновременно слегка размывают детали. Размер окна (5, 5) можно менять — чем он больше, тем сильнее размытие.

Основные манипуляции с использованием модуля Image



Мы можем оперировать изображением не только с помощью обрезки и изменения размера.

Другим распространенным методом является поворот или отражение . Вы можете использовать метод `.transpose()` для некоторых преобразований.

Есть семь опций, которые вы можете передать в качестве аргументов `.transpose()`:

- `Image.FLIP_LEFT_RIGHT`: переворачивает изображение слева направо, в результате чего получается зеркальное отображение.
- `Image.FLIP_TOP_BOTTOM`: переворачивает изображение сверху вниз.
- `Image.ROTATE_90`: поворачивает изображение на 90 градусов против часовой стрелки.
- `Image.ROTATE_180`: поворачивает изображение на 180 градусов.
- `Image.ROTATE_270`: изображение поворачивается на 270 градусов против часовой стрелки, что соответствует повороту на 90 градусов по часовой стрелке.
- `Image.TRANSPOSE`: перемещает строки и столбцы, используя верхний левый пиксель в качестве источника, при этом верхний левый пиксель в перемещенном изображении такой же, как и в исходном изображении.
- `Image.TRANSVERSE`: перемещает строки и столбцы, используя нижний левый пиксель в качестве источника, при этом нижний левый пиксель остается фиксированным относительно исходной и измененной версиями.

```
from PIL import Image

filename = "buildings.jpg"
with Image.open(filename) as img:
    img.load()

converted_img = img.transpose(Image.FLIP_TOP_BOTTOM)
converted_img.show()
```

Основные манипуляции с использованием модуля Image



Часто вам может понадобиться обрезать и изменить размер изображения. Класс Image имеет два метода, которые можно использовать для выполнения этих операций: `.crop()` и `.resize()`:

Аргумент `.crop()` должен состоять из четырех элементов, определяющих левый, верхний, правый и нижний края области, которую вы хотите обрезать. Система координат, используемая в Pillow, назначает координаты (0, 0) пикселю в верхнем левом углу. Это та же система координат, которая обычно используется для двумерных массивов. Кортеж из четырех элементов представляет собой следующую область изображения:

```
cropped_img = img.crop((300, 150, 700, 1000))
cropped_img.size
(400, 850)

cropped_img.show()

low_res_img = cropped_img.resize(
    (cropped_img.width // 4, cropped_img.height // 4))
low_res_img.show()
```