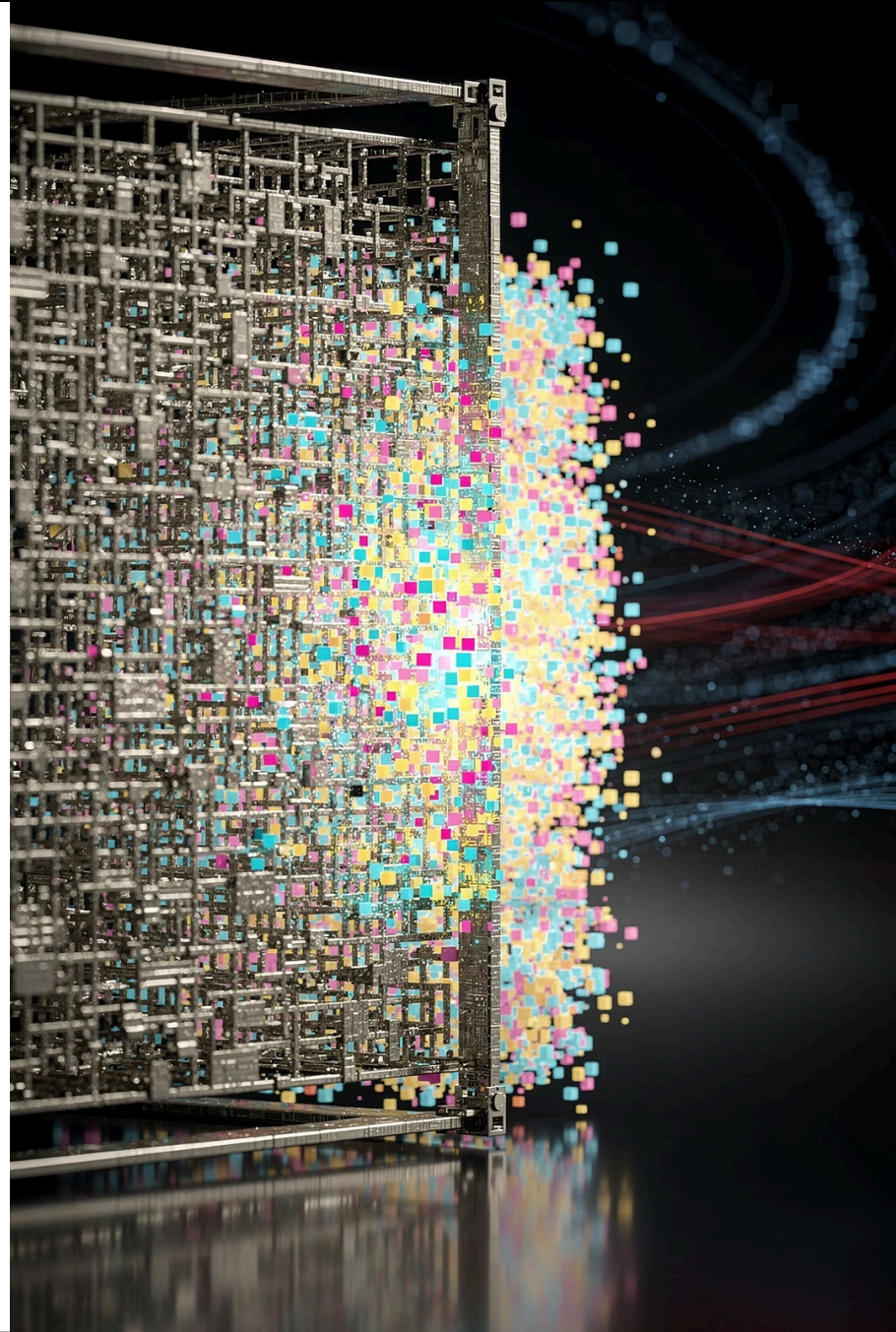
ЛЕКЦИЯ 2 COMPUTER VISION

Изображения как данные + базовый препроцессинг в OpenCV

Как компьютер «видит» картинку, зачем и как готовить данные перед подачей в модель, и какие ошибки подстерегают на каждом шаге.



Зачем вообще нужен препроцессинг?

Для модели машинного обучения «картинка» — это **матрица чисел**. Нейросеть не видит котиков и машины — она потребляет тензоры с пиксельными значениями. Качество этих чисел напрямую определяет качество предсказаний.

Единый формат

Приведение к одному размеру, числу каналов и диапазону значений — без этого батч не соберётся

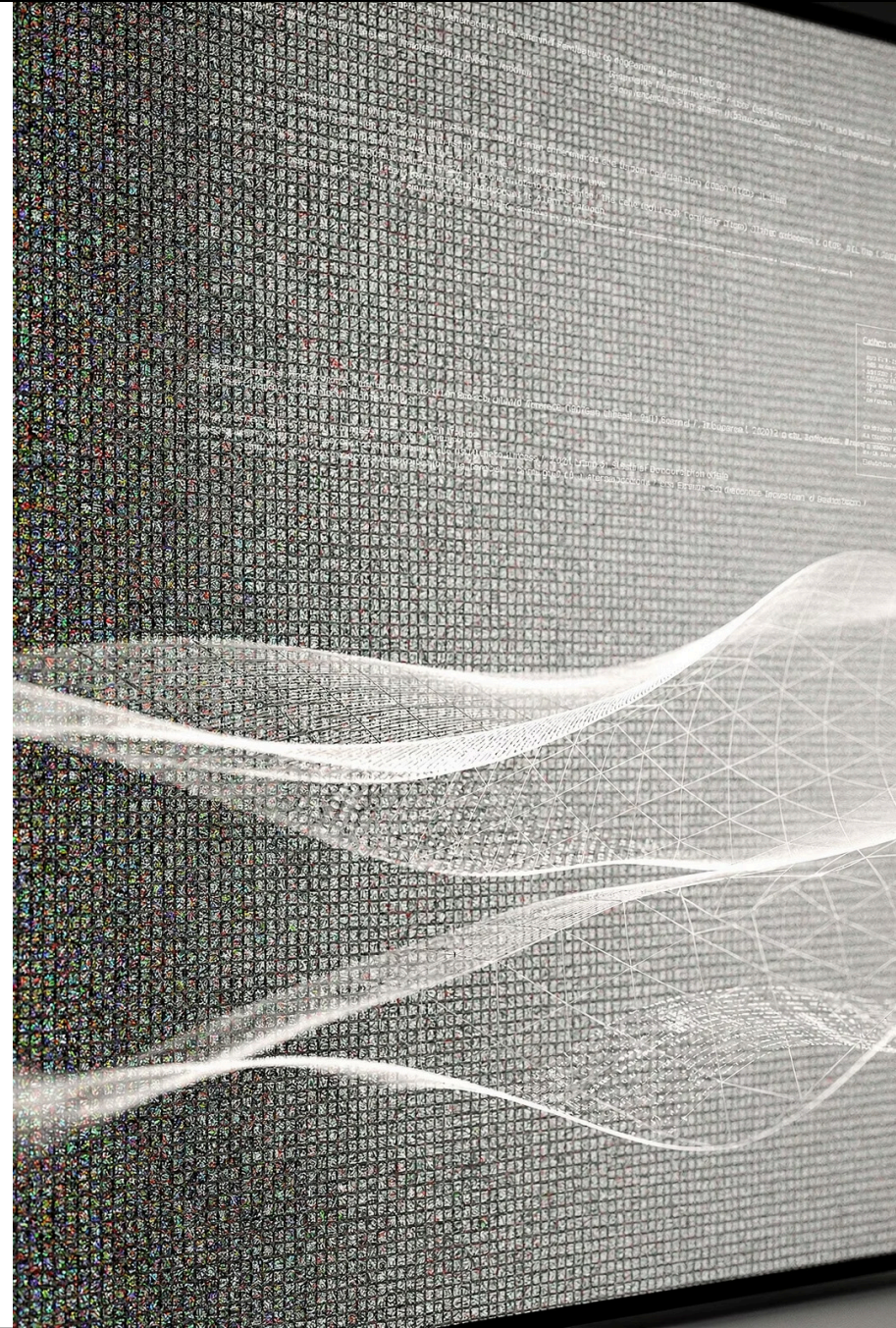
Устойчивость

Повышение робастности к шуму, условиям съёмки, освещению и артефактам камеры

Скорость

Меньше пикселей — быстрее проход через сеть. Оптимизация пайплайна экономит часы обучения

⚠️ **Риск:** чрезмерный «улучшайзинг» может **убить полезный сигнал** и ухудшить качество модели. Каждая трансформация должна быть обоснована задачей.



Изображение как тензор: структура данных

Прежде чем обрабатывать изображение, нужно точно понимать, как оно представлено в памяти. Это фундамент, на котором строится весь пайплайн.

Shape и порядок осей

В NumPy/OpenCV стандартный формат — `H × W × C` (высота, ширина, каналы). В PyTorch используется `C × H × W` — каналы идут первыми. Перепутать порядок осей — одна из самых частых ошибок у начинающих.

Dtype и диапазон

`uint8` хранит значения 0–255, а `float32` обычно нормализуется в диапазон 0–1 или по статистике датасета. Смешение типов — источник трудноуловимых багов.

Каналы изображения

Grayscale — 1 канал яркости

RGB / BGR — 3 канала цвета

🔥 Частые ошибки

- Перепутали RGB ↔️ BGR
- Забыли нормализацию
- Неверный порядок каналов при конвертации NumPy → Torch

OpenCV basics: чтение, запись и цветовые ловушки

OpenCV — де-факто стандарт для низкоуровневой работы с изображениями. Однако у библиотеки есть исторически сложившаяся особенность, которая ловит новичков снова и снова.

cv2.imread()

Загружает изображение с диска. По умолчанию возвращает массив в формате **BGR**, а не RGB!

cv2.cvtColor()

Конвертация каналов: `COLOR_BGR2RGB` и обратно. Обязательный шаг перед визуализацией в matplotlib или подачей в модель

cv2.imwrite()

Сохраняет изображение. Ожидает BGR — если подать RGB, цвета на диске будут инвертированы

> Почему BGR? OpenCV создавался в 1999 году, когда BGR был стандартом для камер и Windows API. Эта «историческая» особенность сохраняется до сих пор ради обратной совместимости.

Геометрия: размер, аспект и масштабирование

Изменение размера (resize) — самая частая операция в препроцессинге. Но наивный resize может уничтожить пропорции объекта или потерять мелкие детали, критичные для задачи.

Простой resize

«Ломает» пропорции объекта. Квадрат становится прямоугольником, лица — вытянутыми. Модель учится на искажённых данных.

Resize + Pad (Letterbox)

Масштабируем с сохранением аспекта, добавляя отступы. Используется в YOLO и других детекторах. Пропорции сохранены.

Center / Random Crop

Вырезаем нужный регион. Center crop — для инференса, random crop — для аугментации при обучении. Риск: объект может не попасть в кадр.

INTER_AREA

Лучший выбор для **уменьшения**. Усредняет пиксели, минимизируя муар

INTER_LINEAR

Быстрая билинейная интерполяция для **увеличения**. Хороший баланс скорости и качества

INTER_CUBIC

Бикубическая интерполяция — качественнее, но медленнее. Для случаев, когда важна детализация

ГЛАВА 2 ЗНАЧЕНИЯ ПИКСЕЛЕЙ

Нормализация, цвет и освещение

Нормализация значений — мост между «сырым» изображением и моделью. Цветовые пространства — инструмент для борьбы с вариативностью освещения.

Нормализация значений

Первый шаг: `uint8 → float32`, затем деление на 255 для перевода в диапазон [0, 1]. Для pretrained моделей (ImageNet) применяют формулу:

$$x_{norm} = \frac{x - \mu}{\sigma}$$

Используют **mean** = [0.485, 0.456, 0.406] и **std** = [0.229, 0.224, 0.225] — статистику ImageNet. Если backbone обучался на других данных, нужно использовать **его** статистику.

Цветовые пространства

RGB / BGR

Универсальный формат. Три канала — красный, зелёный, синий

HSV / HLS

Удобен для цветовых порогов и стабилен к изменениям освещения

YCrCb

Отделяет яркость (Y) от цвета. Полезен для выравнивания контраста

⚡ **CLAHE** (адаптивное выравнивание гистограммы) повышает локальный контраст, но при неаккуратном использовании создаёт артефакты и уничтожает полутона. Применяйте с осторожностью и всегда визуализируйте результат.

Шум, сглаживание и работа с ROI

Реальные изображения содержат шум от сенсора камеры, сжатия и условий съёмки. Фильтрация помогает — но может навредить, если применяется без понимания задачи.

1

Типы шумов

Gaussian — равномерный шум сенсора. **Salt & Pepper** — «битые» пиксели. **Motion blur** — смазывание от движения камеры или объекта

2

Фильтры OpenCV

`GaussianBlur` — сглаживание гауссовым ядром. `medianBlur` — лучший против salt & pepper. `bilateralFilter` — сохраняет края при удалении шума

3

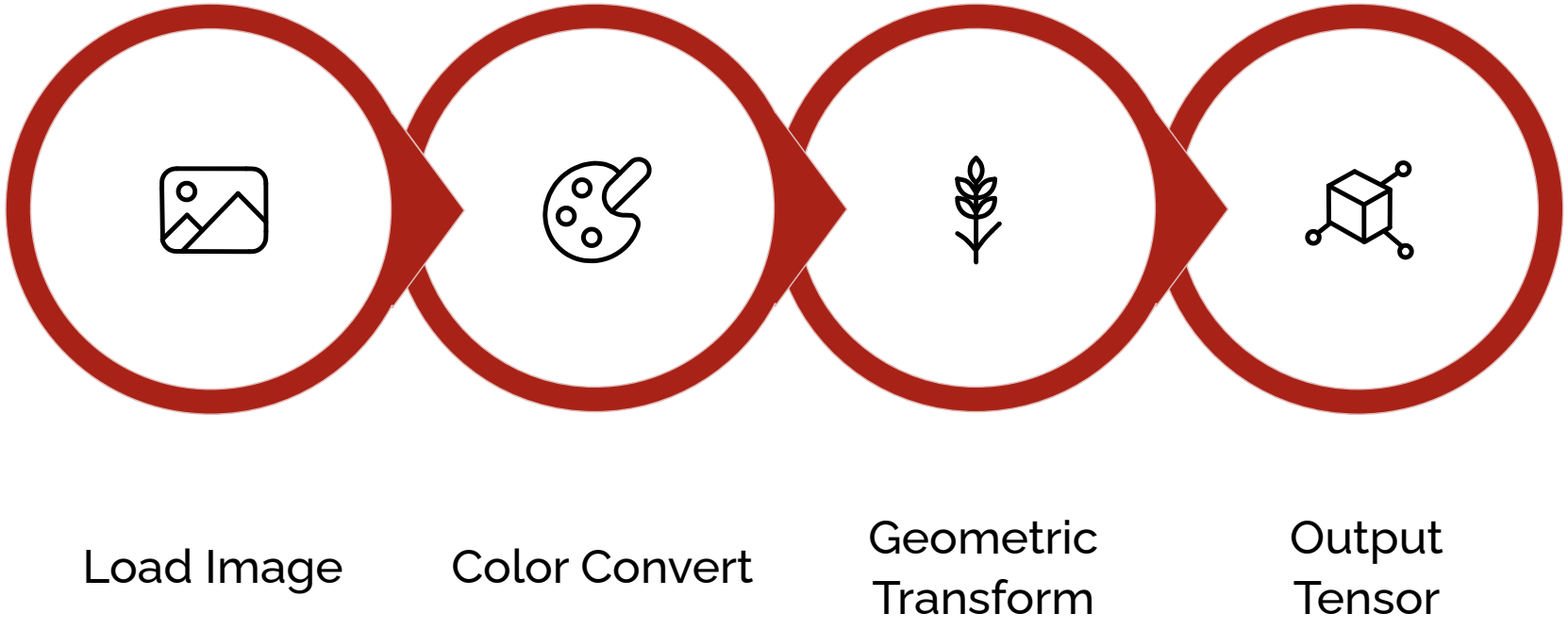
Crop и ROI

Вырезание области интереса (Region of Interest) — зона кассы, документ, лицо. Помогает модели сфокусироваться, но объект может выходить за границы — метка станет неверной

> Правило: применяйте фильтры **только** если есть доказуемая польза для конкретной задачи. Если фильтр удаляет текст, мелкие дефекты или тонкие линии — он вредит.

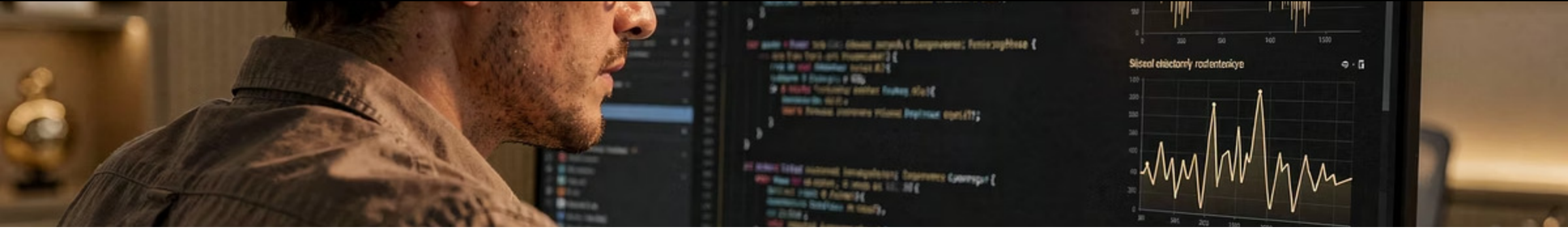
Пайплайн препроцессинга как функция

Препроцессинг — не набор случайных операций, а **строгая функция** с контрактами на вход и выход. Правильно спроектированный пайплайн — основа воспроизводимости экспериментов.



Пайплайн должен иметь два режима работы: **детерминированный** для тестирования и инференса и **случайный** для обучения с аугментациями.

Строгие контракты	Протоколирование	preprocess(img, mode)
Фиксированные shapes, dtype и диапазоны значений на входе и выходе	Возможность записать параметры каждой трансформации для отладки	Единая точка входа: `mode="train"` со случайностью, `mode="eval"` — строго фиксировано



Sanity checks и связь с обучением

Даже идеальный пайплайн может сломаться при рефакторинге, обновлении библиотек или смене данных. Проверки — обязательная часть рабочего процесса.

Быстрые проверки

- Визуализация «до/после»

  Показать 8–16 примеров. Цвета нормальные? Пропорции сохранены?

- Диапазон значений

  Проверить min/max — ожидаем [0, 1] или [-2.1, 2.6] после нормализации. Нет NaN/inf?

- Формирование батча

  DataLoader собирает batch без ошибок? Все тензоры одного shape?

Место в ML-пайплайне

Классическая цепочка PyTorch:

`Dataset → Transform → DataLoader`

Train transforms включают случайные аугментации: random crop, flip, color jitter. **Eval transforms** — строго детерминированные: resize, center crop, нормализация.

> 🚫 Никогда не применяйте аугментации в eval/test! Случайные трансформации делают метрики невоспроизводимыми и искажают оценку модели.

ИТОГИ | ЛЕКЦИЯ 2

Что вы должны уметь после этой лекции

Подведём итоги: четыре ключевых навыка, которые формируют фундамент работы с изображениями в задачах компьютерного зрения.

Чтение и цветовая корректность

Загрузить изображение через OpenCV, распознать BGR-формат, корректно конвертировать в RGB для визуализации и подачи в модель

Геометрия без потерь

Привести изображение к нужному размеру, сохранив пропорции — через letterbox, crop или pad, с правильным выбором интерполяции

Нормализация под модель

Перевести значения в float32, нормализовать по статистике backbone-сети (ImageNet mean/std или кастомной)

Единый пайплайн + проверка

Собрать все шаги в функцию с режимами train/eval, протестировать на примерах и убедиться в корректности визуально и программно

Мини-правило на память: «Если не можешь объяснить трансформацию — не применяй»