Обработка изображений

Цифровое изображение представляет собой двумерный массив пикселей, где каждый пиксель имеет определенное значение яркости или цвета. Основная задача обработки изображений — улучшение визуального качества, выделение значимой информации и подготовка данных для дальнейшего анализа.

Классификация методов обработки изображений

1. Пространственные методы обработки работают непосредственно с пикселями изображения. К ним относятся:

1.1 Линейная фильтрация:

- Сглаживающие фильтры (усредняющий, гауссовский) — уменьшают шум
- Фильтры повышения резкости (лапласиан, фильтр Собеля) — подчеркивают границы
- Медианный фильтр — эффективно удаляет импульсный шум

Нелинейные операции:

- Бинаризация — преобразование в черно-белое изображение
- Морфологические операции (эрозия, дилатация) — работа с формой объектов

Пространственные методы обработки изображений — это метод, которые работают непосредственно с пикселями изображения, манипулируя их значениями яркости или цвета. В отличие от частотных методов, пространственная обработка не требует преобразования изображения в другую область.

Математическая основа:

Изображение представляется как функция f(x,y), где x и y — пространственные координаты, а значение функции — интенсивность пикселя.

Классификация пространственных методов

1. Точечные операции (Point Processing)

Точечные операции — преобразования, при которых новое значение пикселя зависит только от его исходного значения.

Основные виды точечных операций:

Линейные преобразования:

- Контрастирование: $g(x,y) = \alpha \cdot f(x,y) + \beta$
- Негатив: $g(x,y) = L - f(x,y)$, где L — максимальная яркость
- Логарифмическое преобразование: усиливает темные области

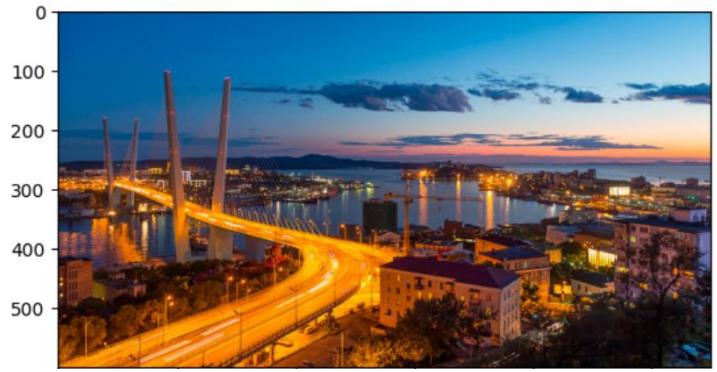
Нелинейные преобразования:

- Гамма-коррекция: $g(x,y) = c \cdot [f(x,y)]^{\gamma}$
- Пороговая обработка: создание бинарного изображения
- Гистограммная эквализация: улучшение контраста

Гамма коррекция

In [3]:
```
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
```

Out[3]:
```
<matplotlib.figure.AxesImage at 0x2131823b3d0>
```




```

In [6]:
```
import numpy as np
gamma=0.5
    # Нормализация изображения
image_normalized = img / 255.0
    # Применение гамма-коррекции
corrected = np.power(image_normalized, gamma)
    # Возврат к исходному диапазону
img2=np.uint8(corrected * 255)

plt.imshow(cv2.cvtColor(img2, cv2.COLOR_BGR2RGB))
```

Out[6]:
```
<matplotlib.figure.AxesImage at 0x2131accea30>
```


```

Локальные операции

Локальные операции используют информацию из окрестности пикселя для вычисления его нового значения.

Свёртка и фильтрация

Маска свертки (ядро) — небольшой массив чисел, который перемещается по изображению:

Ядро 3x3 для усреднения:

[1/9, 1/9, 1/9]

[1/9, 1/9, 1/9]

[1/9, 1/9, 1/9]

Процесс свертки:

1. Накладываем ядро на область изображения
2. Умножаем соответствующие элементы
3. Суммируем результаты
4. Записываем значение в центральный пиксель

Сглаживающие фильтры

Назначение: подавление шума, размытие, уменьшение детализации.

Усредняющий фильтр (Box Filter)

Ядро 3x3:

[1, 1, 1]

[1, 1, 1] × 1/9

[1, 1, 1]

Характеристики:

- Простая реализация
- Эффективен против гауссовского шума
- Вызывает значительное размытие границ

Гауссовский фильтр

Ядро 5x5 (σ=1.4):

[2, 4, 5, 4, 2]

[4, 9, 12, 9, 4]

[5, 12, 15, 12, 5] × 1/159

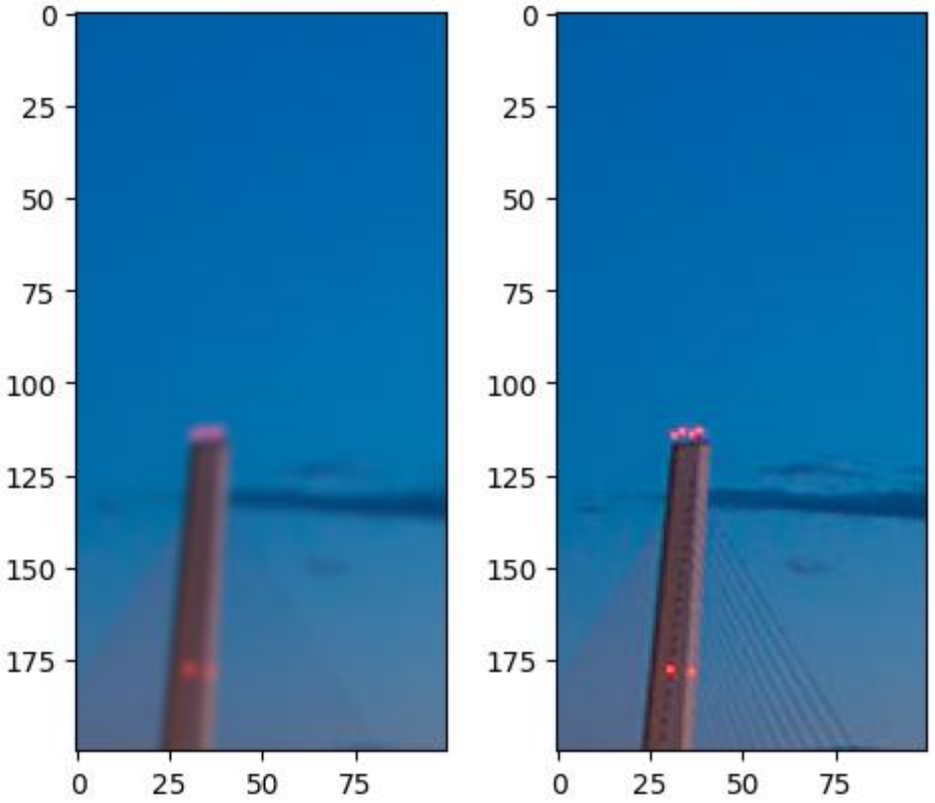
[4, 9, 12, 9, 4]

[2, 4, 5, 4, 2]

Преимущества:

- Сохраняет края лучше, чем усредняющий фильтр
- Изотропное сглаживание
- Теоретическое обоснование

```
denoised = cv2.GaussianBlur(img, (5, 5), 1.4)

plt.imshow(cv2.cvtColor(denoised[0:200,250:350,:], cv2.COLOR_BGR2RGB))
plt.show()
plt.imshow(cv2.cvtColor(img[0:200,250:350,:], cv2.COLOR_BGR2RGB))
```



Фильтры повышения резкости (Sharpening Filters)

Принцип действия: усиление высокочастотных компонент.

Лапласиан (Laplacian Filter)

Вариант 1:	**Вариант 2:**
[0, -1, 0]	**[-1, -1, -1]**
[-1, 4, -1]	**[-1, 8, -1]**
[0, -1, 0]	**[-1, -1, -1]**

Формула усиления резкости:

$g(x,y) = f(x,y) + k \cdot \nabla^2 f(x,y)$, где k — коэффициент усиления

Фильтр Собеля (Sobel Filter)

Gx (вертикальные края):	**Gy (горизонтальные края):**
[-1, 0, 1]	**[-1, -2, -1]**
[-2, 0, 2]	**[0, 0, 0]**
[-1, 0, 1]	**[1, 2, 1]**

Градиент изображения:

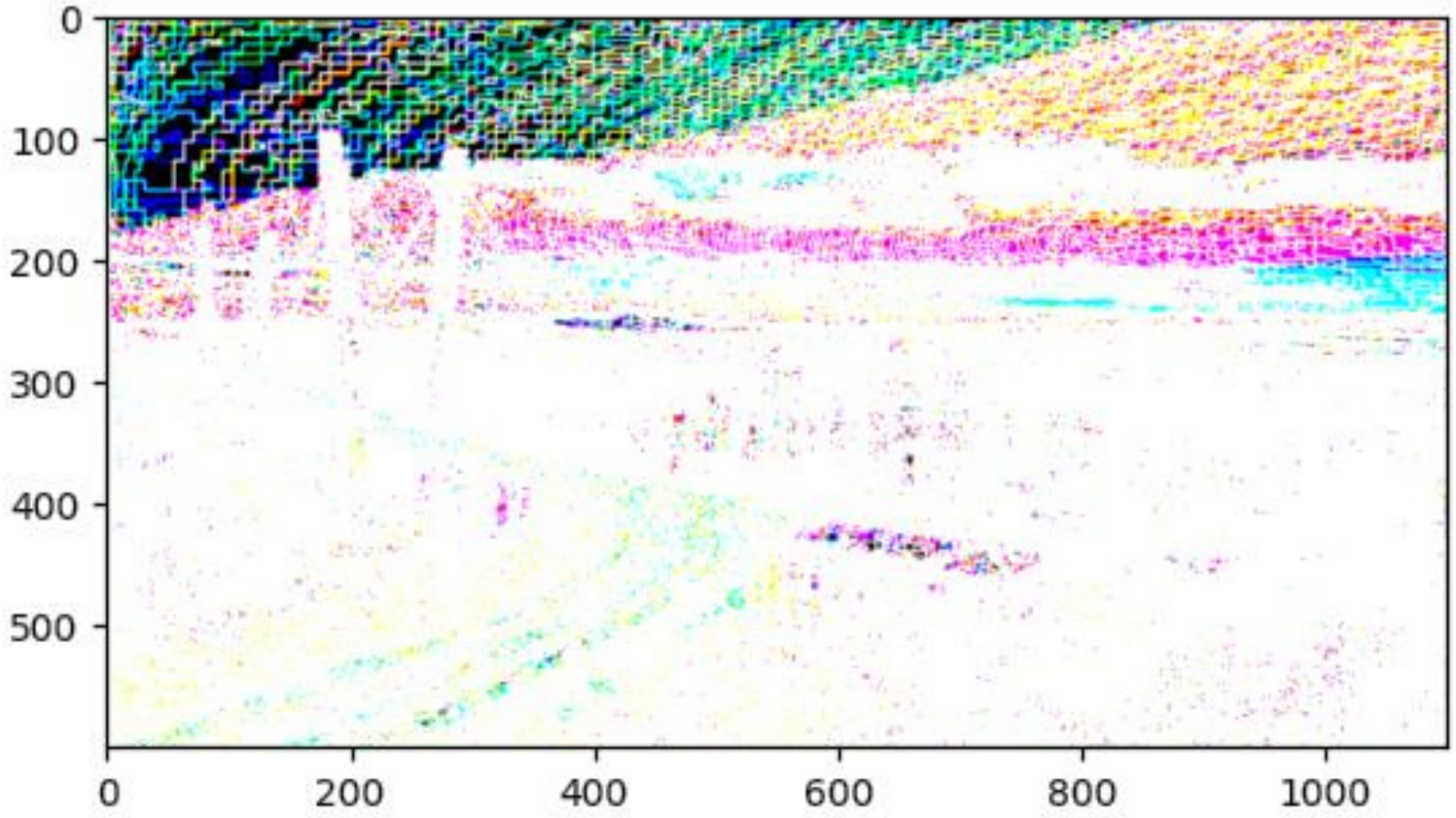
$\nabla f = [Gx, Gy]$

Величина градиента: $|\nabla f| = \sqrt{(Gx^2 + Gy^2)}$

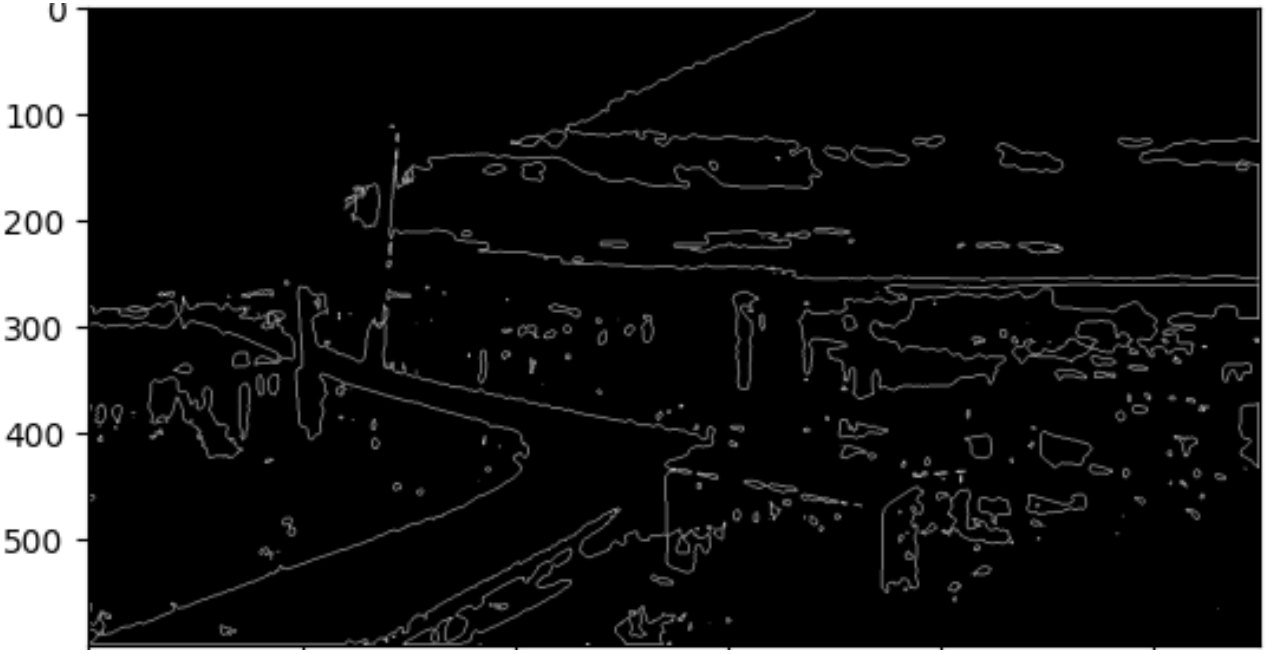
Направление градиента: $\theta = \text{atan2}(Gy, Gx)$

```
sobelx = cv2.Sobel(denoised, cv2.CV_64F, 1, 0, ksize=3)
sobely = cv2.Sobel(denoised, cv2.CV_64F, 0, 1, ksize=3)
edges = np.sqrt(sobelx**2 + sobely**2)
```

```
plt.imshow(cv2.cvtColor(edges.astype('float32'), cv2.COLOR_BGR2RGB))
```



```
filterd_image  = cv2.medianBlur(img,7)
img_grey = cv2.cvtColor(filterd_image,cv2.COLOR_BGR2GRAY)

#set a thresh
thresh = 100

#get threshold image
ret,thresh_img = cv2.threshold(img_grey, thresh, 255, cv2.THRESH_BINARY)

#find contours
contours, hierarchy = cv2.findContours(thresh_img, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)

#create an empty image for contours
img_contours = np.uint8(np.zeros((img.shape[0],img.shape[1])))

cv2.drawContours(img_contours, contours, -1, (255,255,255), 1)

plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))  # исходное
plt.show()
plt.imshow(cv2.cvtColor(img_contours, cv2.COLOR_BGR2RGB))  # выводим итоговое
```



Медианный фильтр (Non-linear Filter)

Принцип работы: замена значения пикселя медианой значений в окрестности.

Алгоритм:

1. Выбираем окно заданного размера (3x3, 5x5)
2. Сортируем значения пикселей в окне
3. Выбираем медианное значение
4. Присваиваем его центральному пикселю

Преимущества:

- Эффективное подавление импульсного шума (salt & pepper)
- Сохранение резких границ
- Не создает новых значений (в отличие от линейных фильтров)

Морфологические операции

Эрозия (Erosion): Если в некоторой позиции каждый единичный пиксел структурного элемента совпадет с единичным пикселом бинарного изображения, то выполняется **логическое сложение** центрального пиксела структурного элемента с соответствующим пикселом выходного изображения. Если структурный элемент начинает вылезать за пределы изображения, то все точки элемента должны помещаться в исходный объект

- Уменьшает размер объектов
- Удаляет мелкие детали
- Разделяет соединенные объекты

Структурный элемент 3x3:

[1, 1, 1]

[1, 1, 1]

[1, 1, 1]

Дилатация (Dilation): обратна эрозии. структурный элемент перемещается по изображению, и если хотя бы один пиксель из области, соответствующей ядру, совпадает с белым (ярким), то центральный пиксель тоже становится белым. Это приводит к расширению светлых областей.

- Увеличивает размер объектов
- Заполняет небольшие отверстия - Соединяет близко расположенные объекты

Составные морфологические операции

Открытие (Opening): эрозия + дилатация

- Удаляет мелкие объекты
- Сохраняет форму крупных объектов

Закрытие (Closing): дилатация + эрозия

- Заполняет небольшие отверстия
- Сглаживает контуры

Практические аспекты реализации

Обработка границ изображения

Проблема: при применении фильтров возникают артефакты на границах.

Методы решения:

1. Игнорирование границ (уменьшение выходного изображения)
2. Зеркальное отражение (padding mirror)
3. Продолжение нулями (zero padding)
4. Периодическое продолжение (wrap around)

Оптимизация вычислений

Разделяемые фильтры:

Некоторые фильтры можно представить как последовательность одномерных операций:

Gaussian 2D = Gaussian_x * Gaussian_y

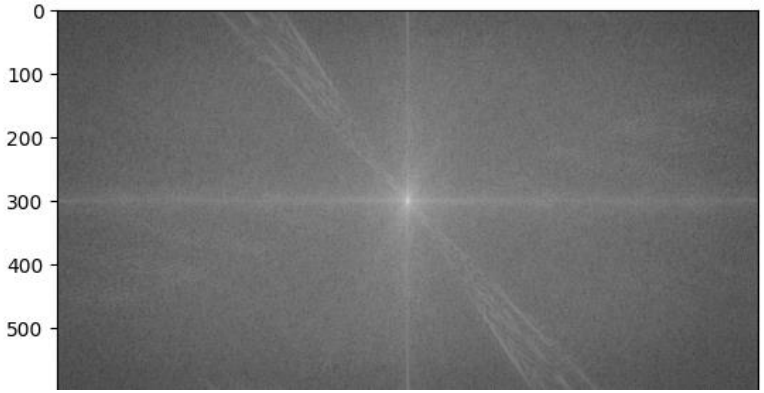
Частотные методы обработки

Частотные методы основаны на преобразовании изображения в частотную область с помощью:

Преобразование Фурье:

- Позволяет анализировать частотные характеристики
- Эффективно для фильтрации периодических шумов
- Используется в алгоритмах сжатия изображений

```
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
plt.show()
#img_df=cv2.dft(img)
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
# Compute the discrete Fourier Transform of the image
fourier = cv2.dft(np.float32(gray), flags=cv2.DFT_COMPLEX_OUTPUT)
# Shift the zero-frequency component to the center of the spectrum
fourier_shift = np.fft.fftshift(fourier)
# calculate the magnitude of the Fourier Transform
magnitude = 20*np.log(cv2.magnitude(fourier_shift[:,:,0],fourier_shift[:,:,1]))
# Scale the magnitude for display
magnitude = cv2.normalize(magnitude, None, 0, 255, cv2.NORM_MINMAX, cv2.CV_8UC1)
# Display the magnitude of the Fourier Transform
plt.imshow(cv2.cvtColor(magnitude, cv2.COLOR_BGR2RGB))
```

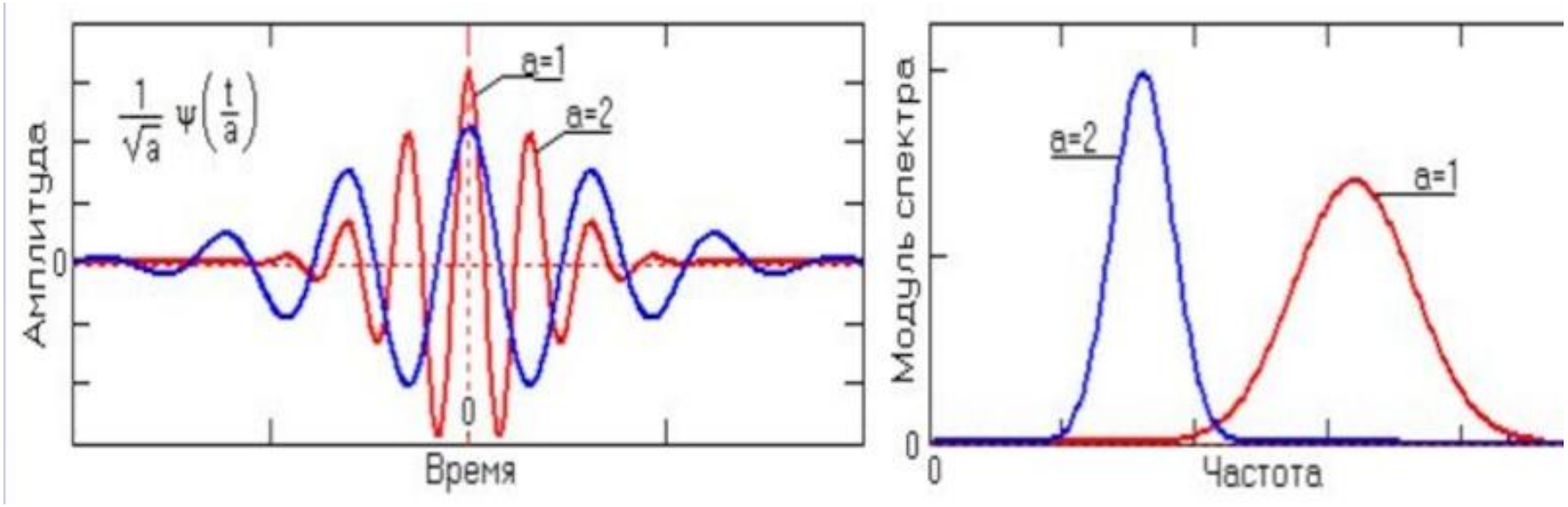


Вейвлет-преобразование:

- Обеспечивает многомасштабный анализ
- Сохраняет пространственную информацию
- Широко применяется в сжатии JPEG2000

Вейвлет-преобразование: математические основы и практические приложения

Вейвлет-преобразование (Wavelet Transform) — это математический инструмент для анализа сигналов, который позволяет изучать их как во временной (пространственной), так и в частотной области одновременно. В отличие от преобразования Фурье, которое дает только частотную информацию, вейвлет-анализ сохраняет временную локализацию сигнала.



Определение вейвлет-функции

Вейвлет (wavelet — "маленькая волна") — это функция $\psi(t)$, удовлетворяющая условиям:

1. Нулевое среднее: $\int\psi(t)dt = 0$
2. Конечная энергия: $\int|\psi(t)|^2dt < \infty$
3. Нормировка: обычно $\int|\psi(t)|^2dt = 1$

Непрерывное вейвлет-преобразование (CWT)

Формула CWT:

$W(a,b) = (1/\sqrt{|a|}) \int f(t) \cdot \psi^*((t-b)/a)\, dt$

где:

- $a > 0$ — параметр масштаба (scale)
- b — параметр сдвига (translation)
- ψ^* — комплексно-сопряженная вейвлет-функция

Дискретное вейвлет-преобразование (DWT)

Дискретизация параметров:

- $a = 2^j$ (дискретный масштаб)
- $b = k \cdot 2^j$ (дискретное положение)

Быстрое вейвлет-преобразование основано на концепции фильтров:

- HPF (High-Pass Filter) — выделяет высокочастотные компоненты (детали)
- LPF (Low-Pass Filter) — выделяет низкочастотные компоненты (аппроксимация)

Основные типы вейвлетов

1. Вейвлет Хаара (Haar Wavelet)

Простейший вейвлет:

$$\psi(t) = \begin{cases} 1, & 0 \le t < 0.5 \\ -1, & 0.5 \le t < 1 \\ 0, & \text{иначе} \end{cases}$$

Преимущества: простота, быстрые вычисления

Недостатки: разрывы производной, плохая частотная локализация

Применение в обработке изображений

1. Сжатие изображений (JPEG2000)

Принцип работы:

1. Разложение изображения на вейвлет-коэффициенты
2. Квантование коэффициентов
3. Энтропийное кодирование

Преимущества перед JPEG:

- Лучшее качество при высоких степенях сжатия
- Отсутствие блочных артефактов
- Прогрессивная передача данных

Основные алгоритмы обработки изображений

1. Алгоритмы сегментации

Сегментация — разделение изображения на значимые области:

Пороговая обработка:

- Простой и эффективный метод
- Основан на гистограмме яркостей
- Автоматический выбор порога (метод Оцу)

Алгоритм водораздела:

- Моделирует процесс затопления рельефа
- Эффективен для разделения соприкасающихся объектов
- Чувствителен к шуму и локальным минимумам

2. Алгоритмы выделения границ

Выделение границ — фундаментальная задача компьютерного зрения:

Оператор Кэнни:

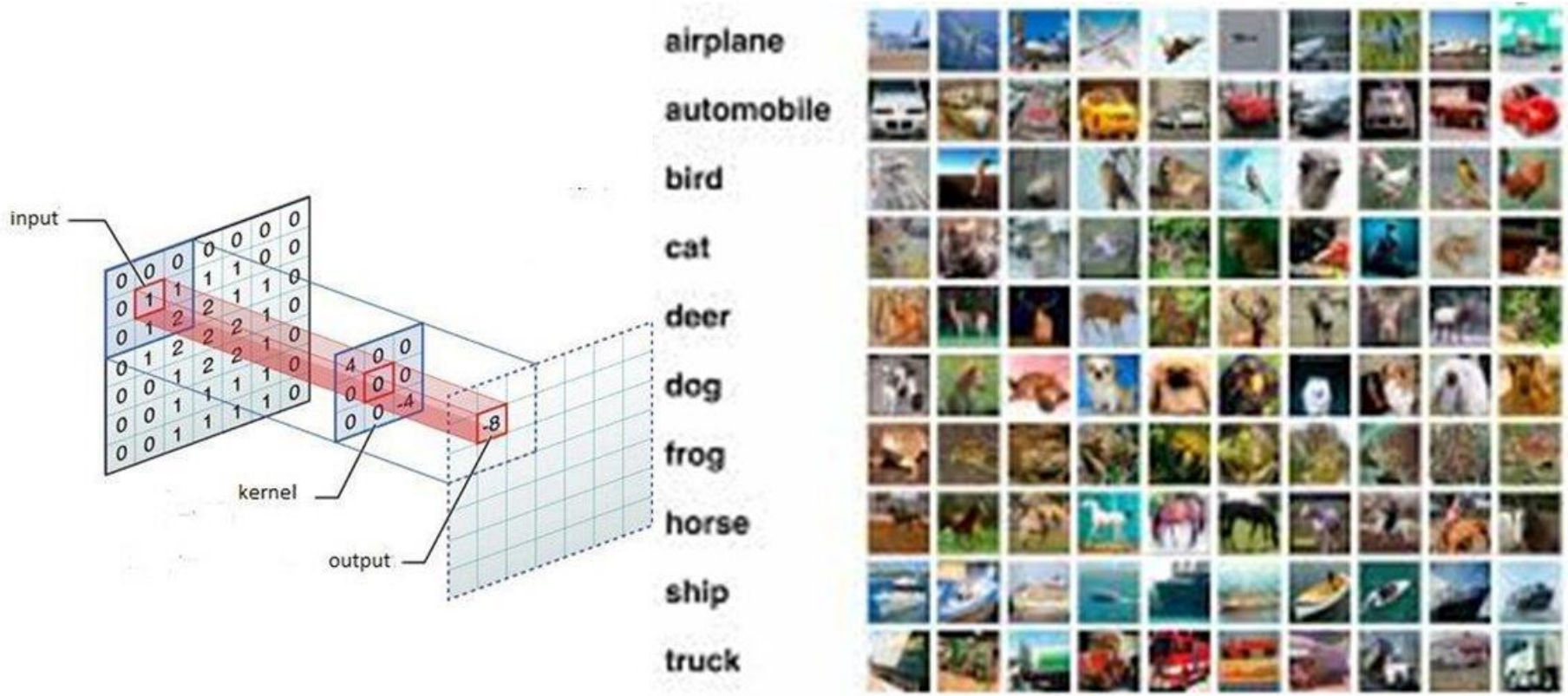
- Многоэтапный алгоритм (сглаживание, градиент, подавление немаксимумов)
- Высокая точность обнаружения границ
- Устойчивость к шуму

Детектор Собеля и Превитта:

- Основаны на вычислении градиента яркости
- Просты в реализации
- Широко используются в реальных приложениях

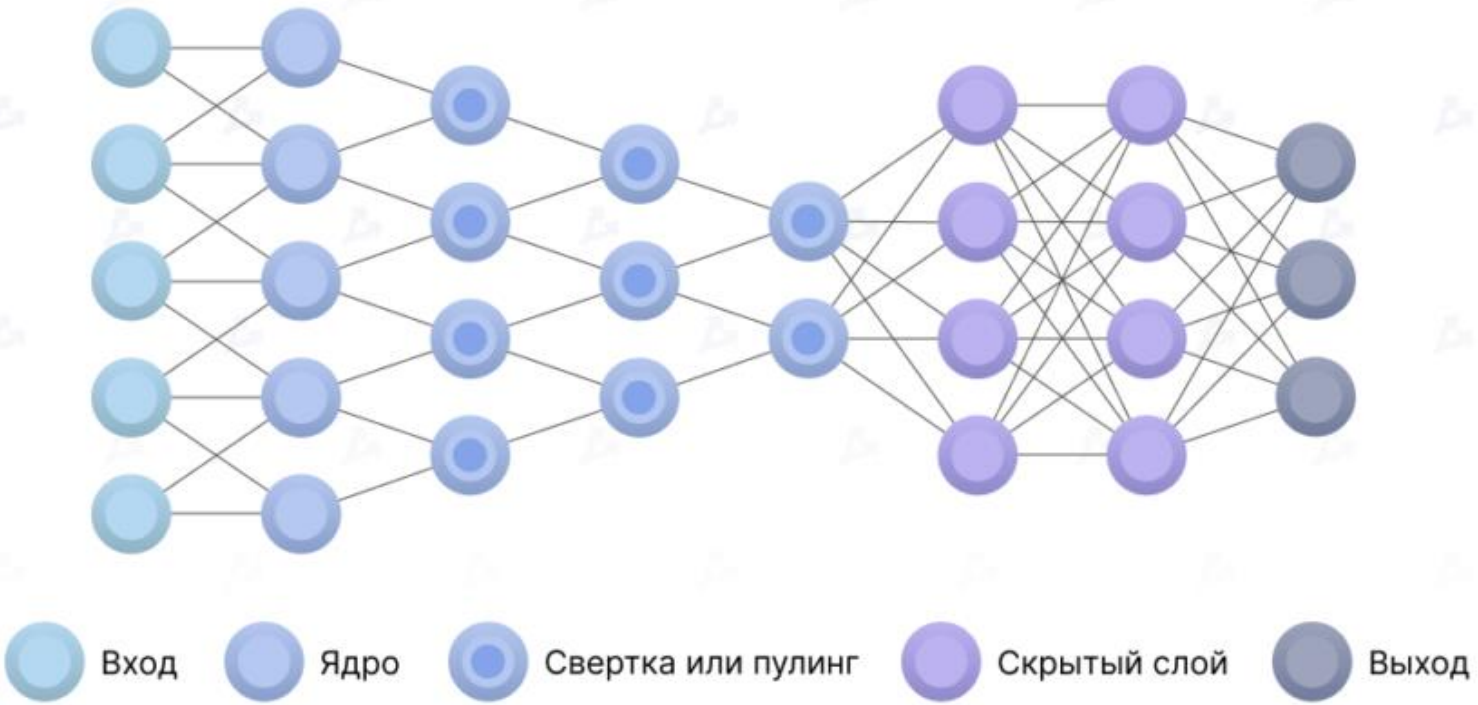
Современные тенденции и методы

1. Глубокое обучение в обработке изображений

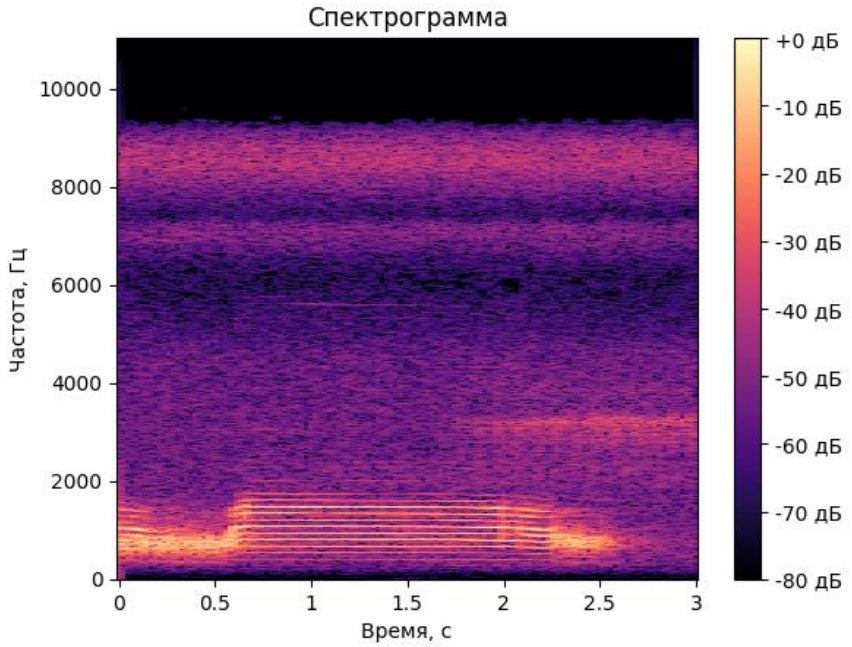


СВЁРТОЧНЫЕ НЕЙРОННЫЕ СЕТИ

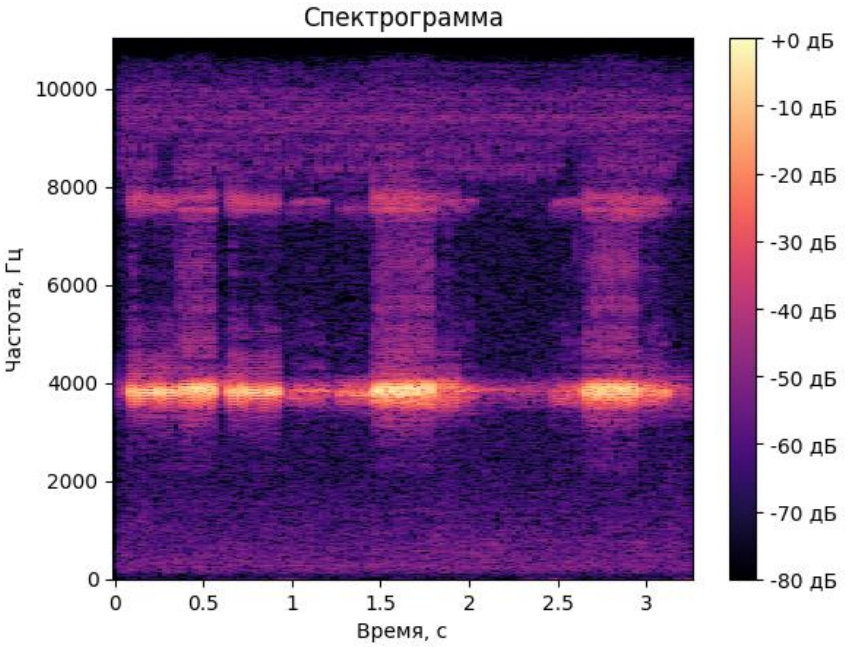
Сверточная нейросеть (CNN)



Сверточная нейронная сеть для обработки звука через изображения



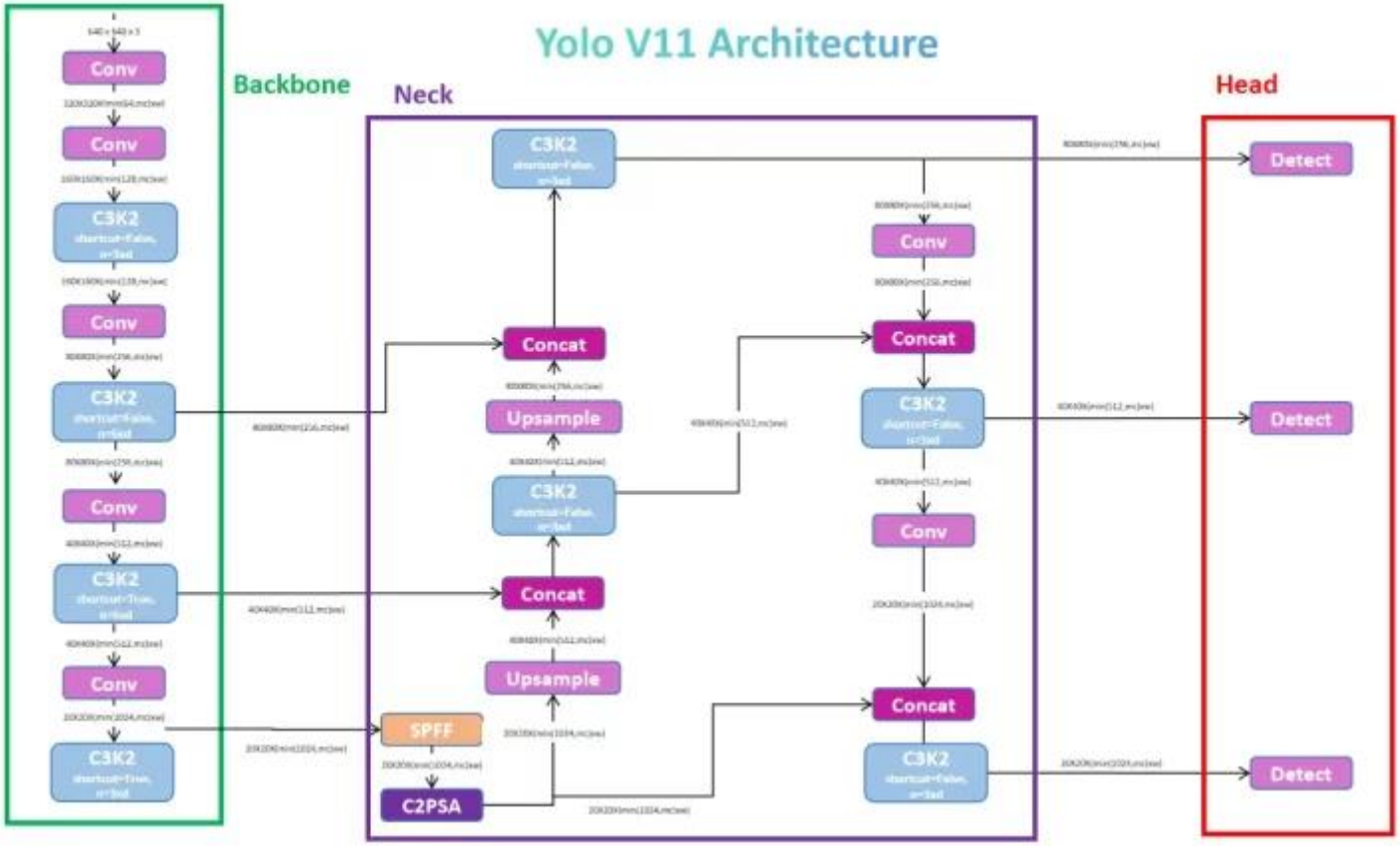
Спектрограмма голоса птицы-капуцина



Спектрограмма звука сверчков

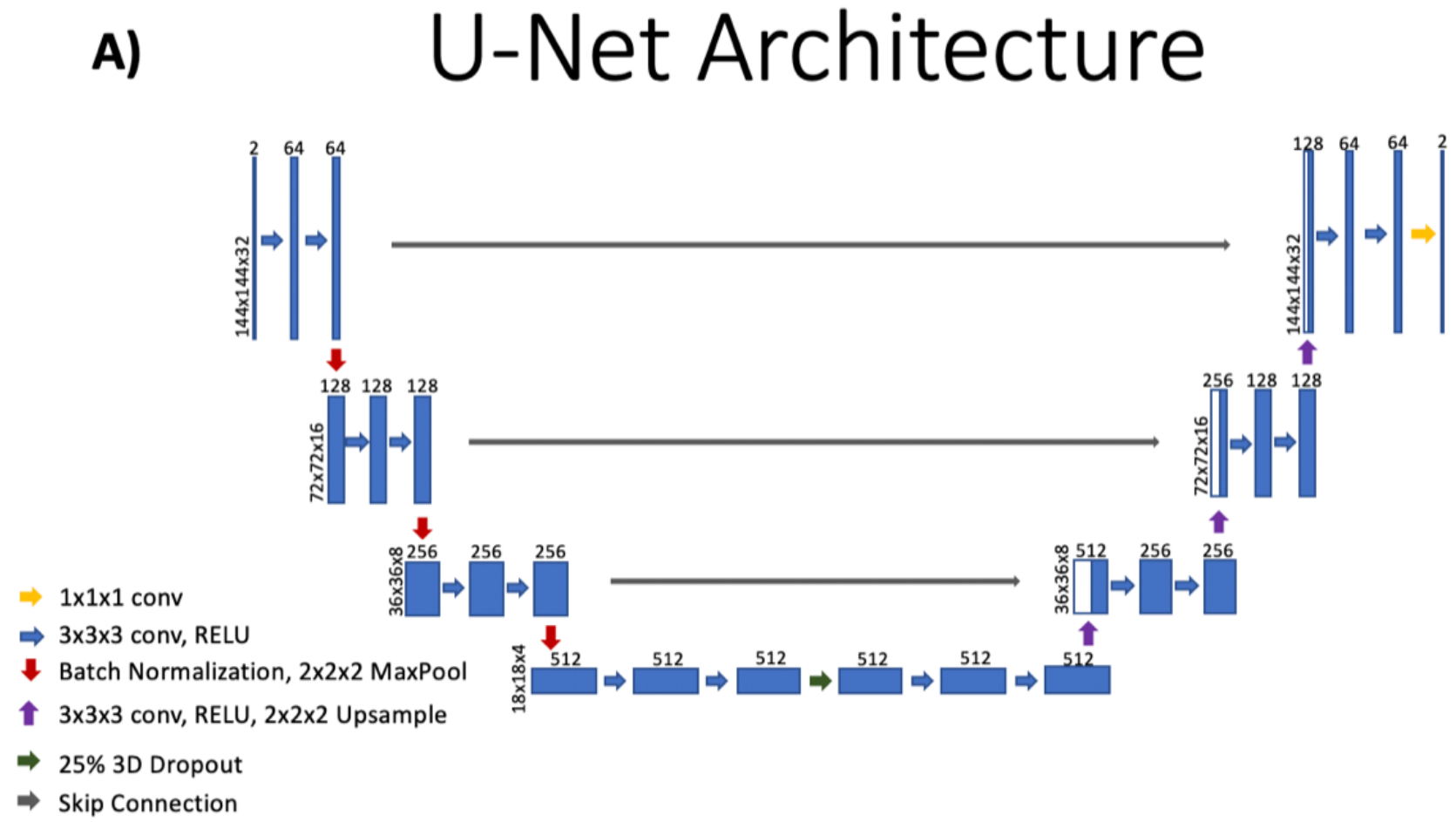
Обнаружение объектов

- YOLO — для обнаружения объектов

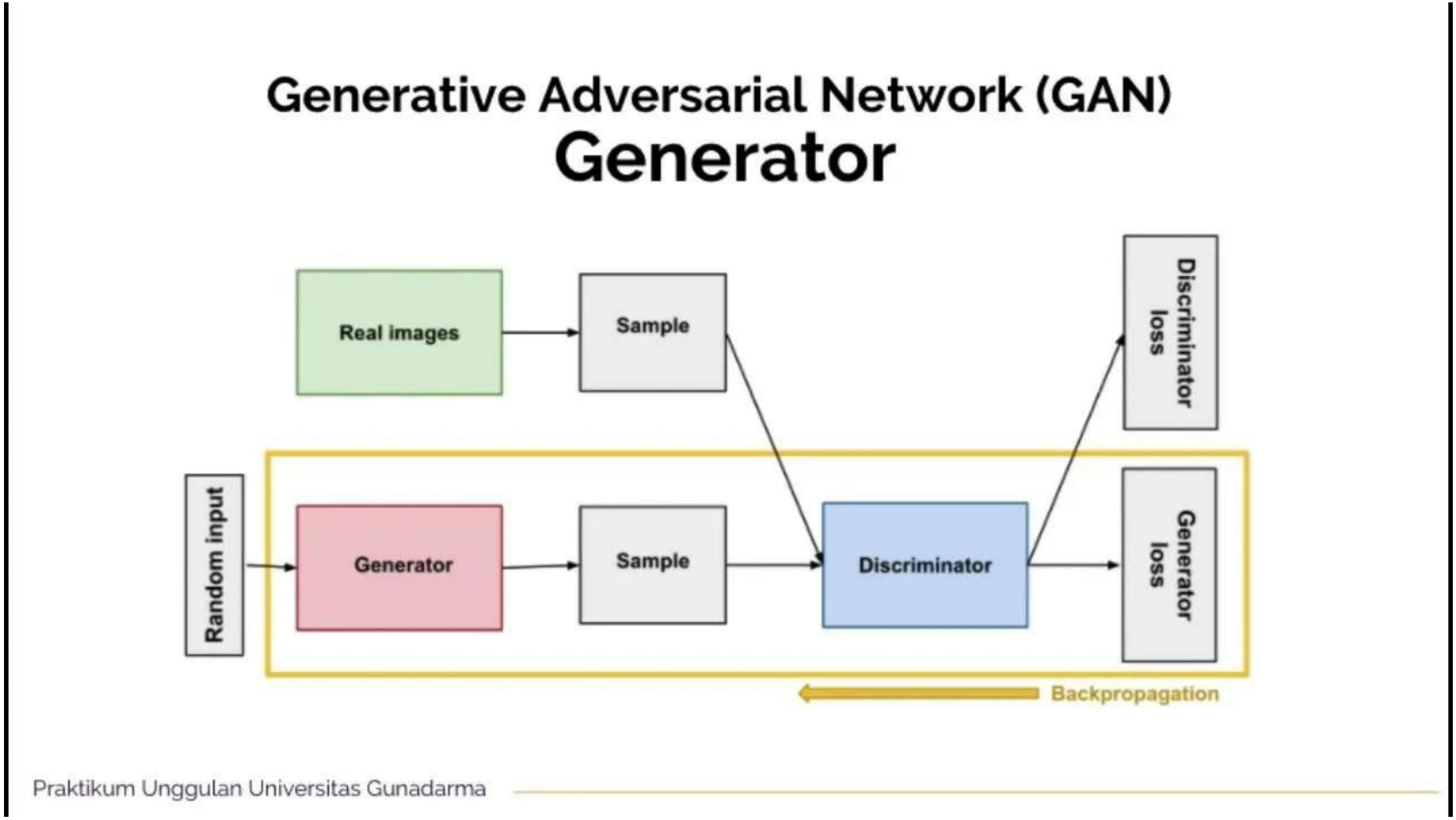


Сегментация изображений

U-NET



- GAN — для генерации и восстановления изображений



Практические аспекты реализации

1. Выбор инструментов

Библиотеки обработки изображений:

- OpenCV — промышленный стандарт
- Scikit-image — для научных исследований
- PIL/Pillow — базовые операции

2. Оптимизация алгоритмов

Критерии оптимизации:

- Скорость обработки
- Требования к памяти
- Точность результатов

Техники оптимизации:

- Пирамидальное представление
- Распараллеливание вычислений
- Аппаратное ускорение (GPU)