НУГ «Цифровые технологии в неврологии»

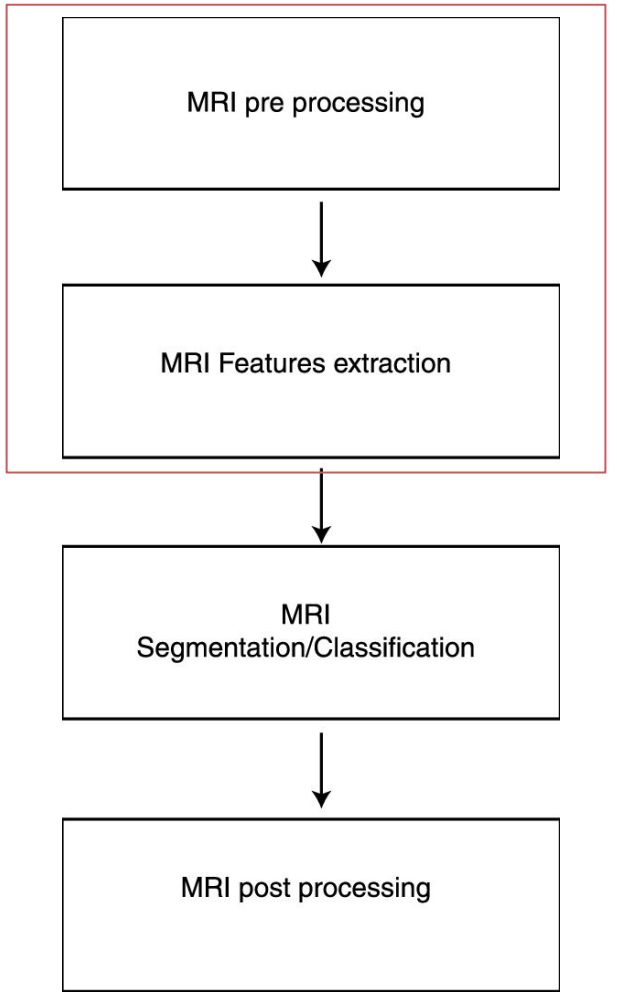
Пермь,
2023

Основные шаги по подготовке изображений для обучения систем компьютерного зрения

Елена Русакова

Предобработка изображений

– важный этап подготовки данных перед использованием в моделях машинного обучения.



Здесь перечислены некоторые основные шаги, которые могут быть необходимы в процессе предобработки изображений

1. Масштабирование
2. Нормализация - приведение значений пикселей к определенному диапазону
3. Цветовое преобразование
4. Устранение шума и выделение контуров
5. Выделение признаков
6. Геометрические преобразования
7. Обрезка

Набор, последовательность шагов - зависит от решаемой задачи

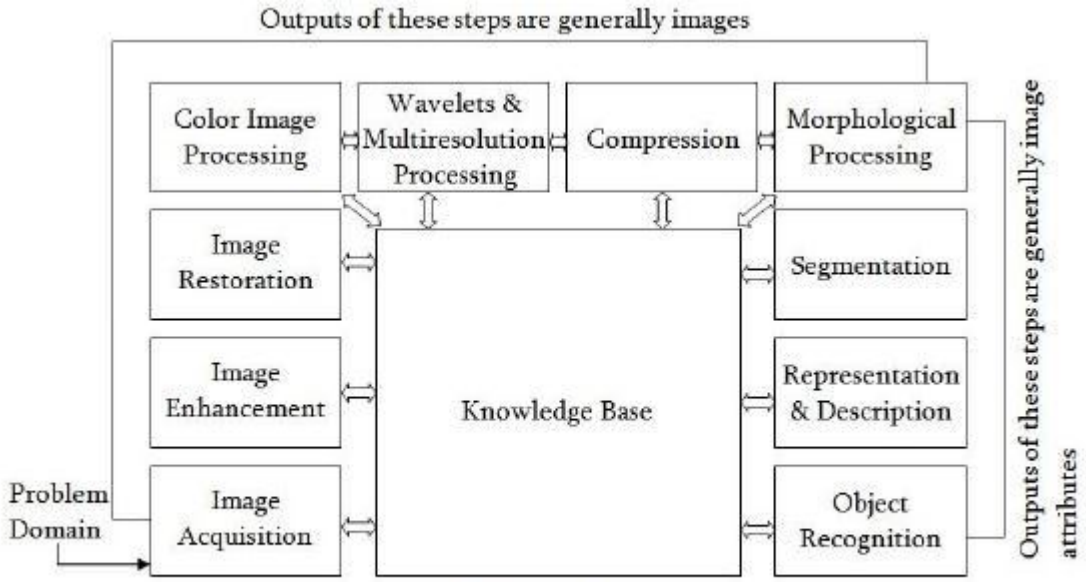


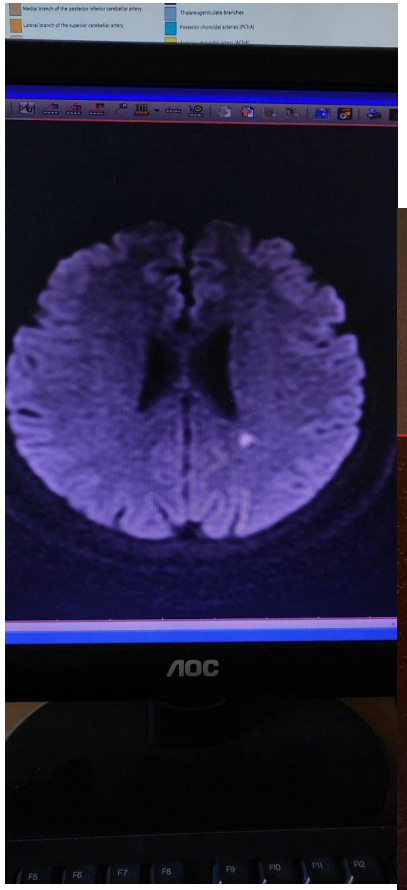
Figure 1

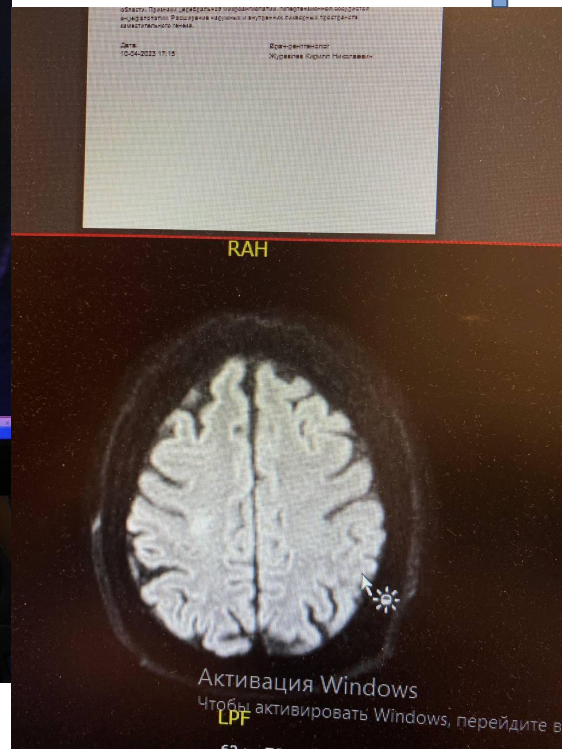
Зачем нам это нужно?

Предобработка может значительно повлиять на качество модели машинного обучения.

Как правило, предобработка требуется в двух случаях:

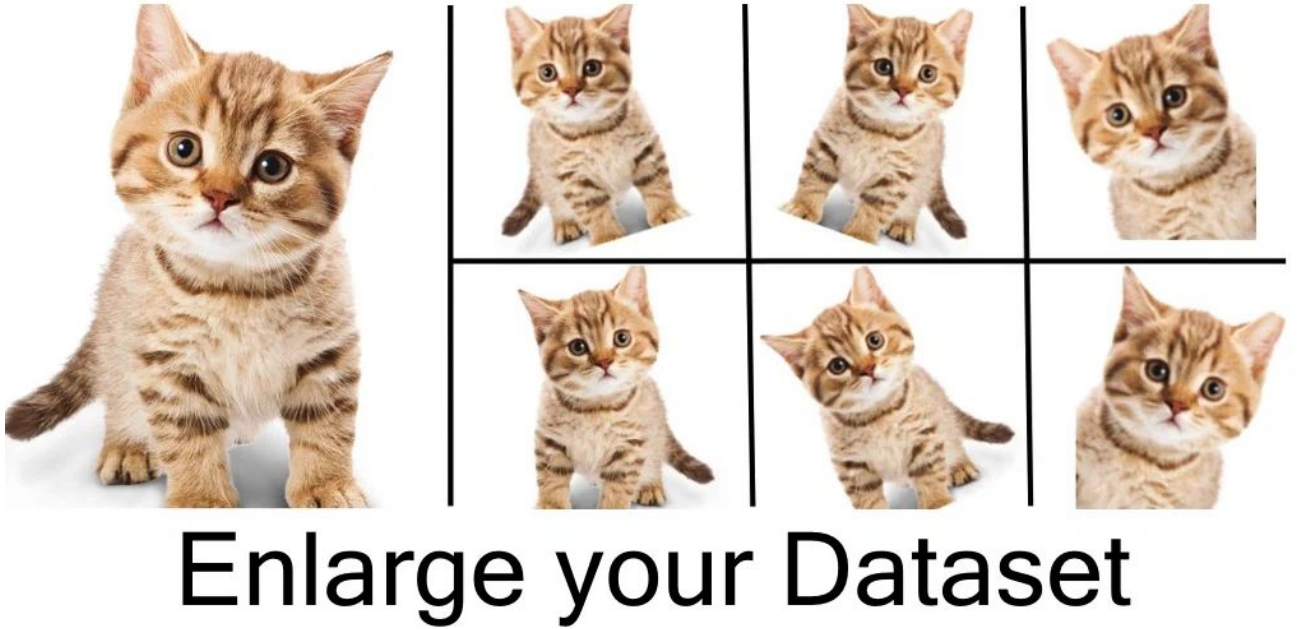
1. **Очистка данных**. Предположим, что на изображениях присутствуют некоторые артефакты. Чтобы облегчить обучение модели, артефакты необходимо удалить на этапе предобработки.





шум (грязь на экране), лишний фон, искаженная перспектива, цвет

2. **Дополнение данных (Аугментации).** Иногда небольших наборов данных недостаточно для качественного глубокого обучения модели. Аугментации могут быть весьма полезны для решения этой проблемы.



Библиотеки

Библиотека	**OpenCV** (Open Source Computer Vision Library)	**scikit-image**	**Pillow** (PIL, Python Imaging Library)	**ImageMagick**
Описание	Кросс-платформенная библиотека с открытым исходным кодом, предоставляющая множество алгоритмов компьютерного зрения для обработки изображений.	Библиотека для обработки изображений с открытым исходным кодом для языка программирования Python. Она расширяет функциональность SciPy, предлагая множество алгоритмов для обработки изображений, таких как фильтрация, сегментация, преобразование, выделение признаков и других.	Pillow - это форк библиотеки PIL, предназначенной для работы с изображениями в Python. Она предоставляет базовые функции для работы с изображениями.	Кросс-платформенная библиотека для обработки изображений с открытым исходным кодом, предоставляющая набор утилит командной строки и API для создания, редактирования, композиции и преобразования изображений
Плюсы	• Большое количество алгоритмов и функций • Кросс-платформенность, • Поддержка аппаратного ускорения (GPU)	• Интеграция с экосистемой Python для вычислений (NumPy, SciPy, matplotlib)	• Поддержка большого количества форматов изображений • Легковесная и быстрая для базовых операций обработки изображений	• Большое количество поддерживаемых форматов изображений • Кросс-платформенность
Минусы	Некоторые функции могут быть медленными или затратными в плане вычислений.	• Меньше функций и алгоритмов, чем в OpenCV • Отсутствие встроенной поддержки аппаратного ускорения (GPU).	Ограниченный набор алгоритмов и функций по сравнению с OpenCV и scikit-image.	• Сложность установки и интеграции с некоторыми системами • Меньше алгоритмов компьютерного зрения, чем в OpenCV и scikit-image.
Написана на	Написана на **C++ (часть на Python)** и предоставляет интерфейсы для Python, Java и других языков.	Написана на Python и тесно интегрирована с NumPy, SciPy и matplotlib	Написана на C и Python	Написана на C

Open CV

1. Основные операции с изображениями

2. Геометрические преобразования: масштабирование, поворот, перевод, аффинные и перспективные преобразования, отражение и деформация

3. Фильтрация изображений: размытие, сглаживание, детектирование границ и контуров, морфологические операции, улучшение резкости и устранение шума

4. Выделение признаков: SIFT, SURF, ORB, HOG, AKAZE, признаки Хаара

5. Сопоставление признаков: методы сопоставления признаков, такие как Brute-Force, FLANN, RANSAC

6. Сегментация изображений: выделение областей на основе цвета, текстуры или формы

7. Обнаружение объектов: каскады Хаара, HOG + SVM, алгоритмы на основе глубокого обучения (DNN)

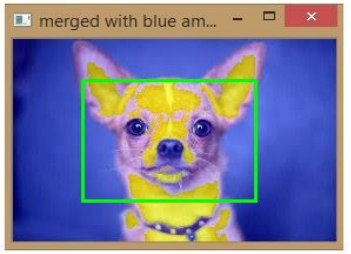
8. Отслеживание объектов: алгоритмы отслеживания объектов, такие как KLT, TLD, CSRT, для отслеживания движения объектов на видео

9. Калибровка камеры и 3D-реконструкция: методы для исправления искажений линз, определения внутренних и внешних параметров камеры, и восстановления 3D-сцены из изображений

Image Processing

OpenCV

python



[open cv - doc]

[история создания open cv]

since 1999 г
made in Нижний Новгород

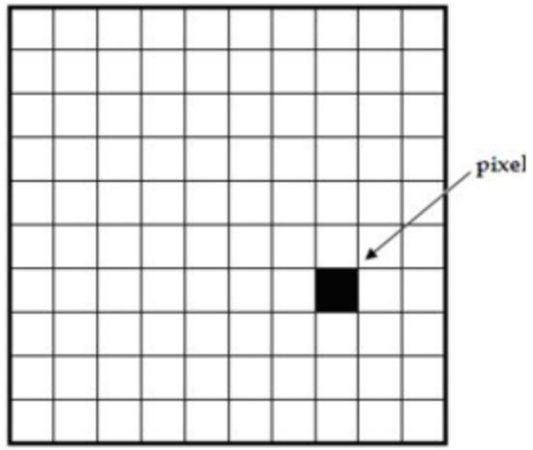
Один из инициаторов создания и открытия миру OpenCV — Гари Брадски

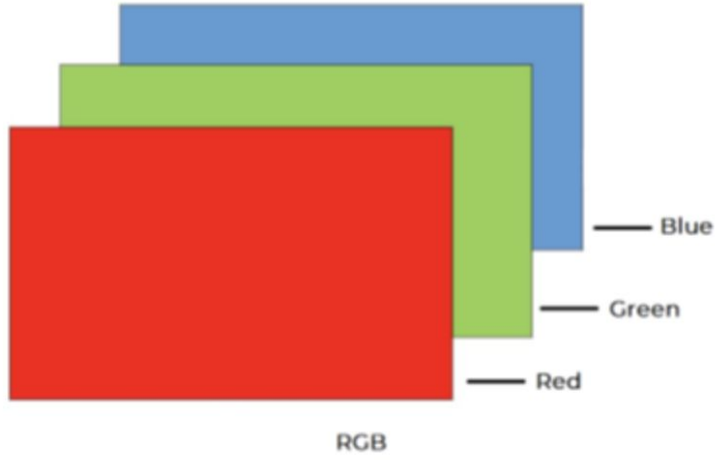
Изображение

Изображение – это массив пикселей, расположенных в столбцах и строках. Пиксели — это элементы изображения, которые содержат информацию о насыщенности.

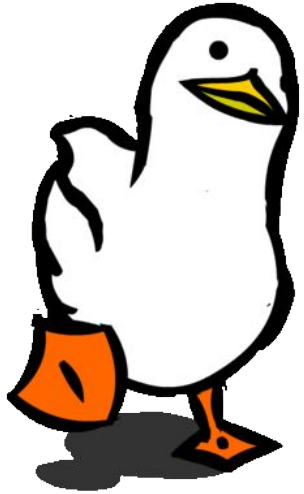
ЧБ – это одномерное пространство черных и белых точек, пикселей.

Цветное изображение обычно представляется в виде трех одномерных пространств, наложенных друг на друга





Рассмотрим внимательнее основные шаги



Масштабирование

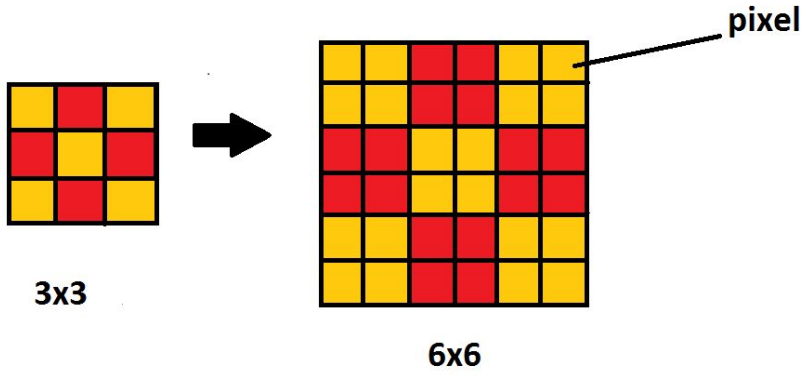
Важный этап, который заключается в изменении размеров изображения без потери качества.

1. **Ближайший сосед (Nearest Neighbor Interpolation)**: Этот метод заключается в выборе ближайшего пикселя в исходном изображении для каждого пикселя в новом изображении. Простой и быстрый, но может вызывать искажения и “ступенчатые края” у объектов на изображениях.

Nearest Neighbor Interpolation

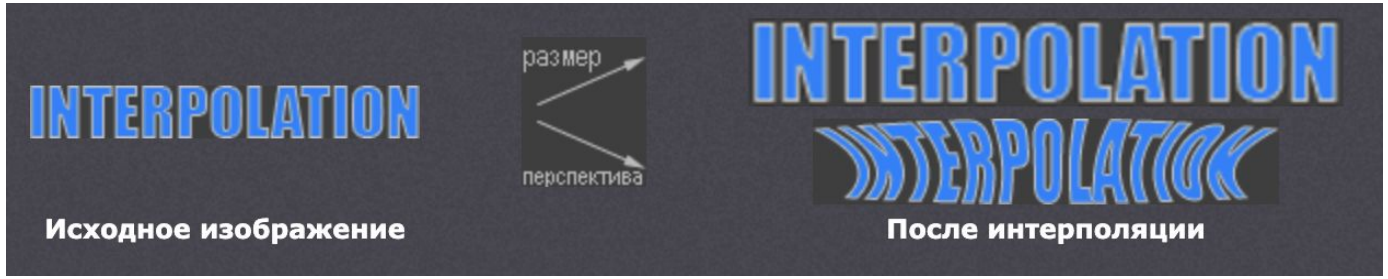
Original		
10	4	22
2	18	7
9	14	25

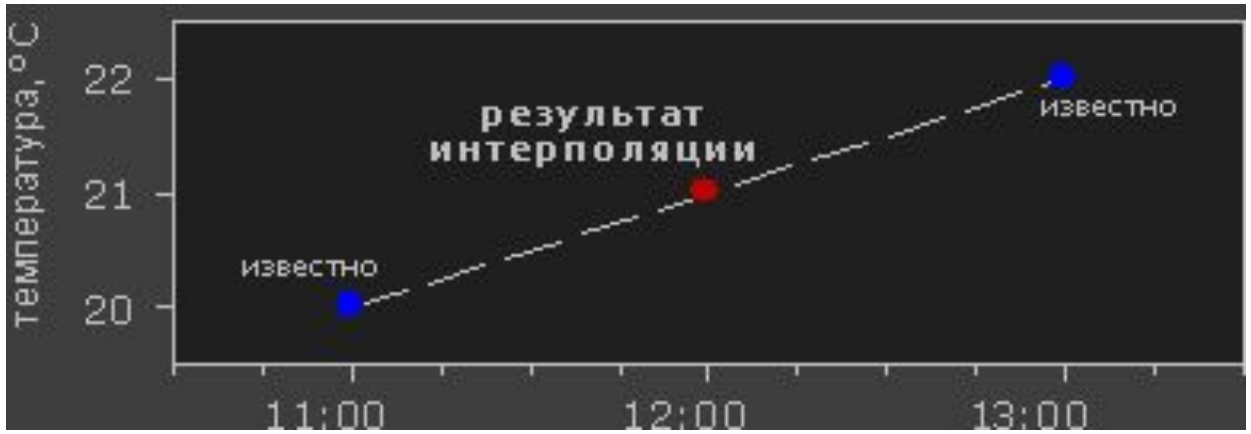
Result					
10	10	4	4	22	22
10	10	4	4	22	22
2	2	18	18	7	7
2	2	18	18	7	7
9	9	14	14	25	25
9	9	14	14	25	25



Интерполяция

Интерполяция изображений происходит во всех цифровых фотографиях на определённом этапе, будь то дематризация или масштабирование. Она происходит всякий раз, когда изменяется размер или развёртка изображения из одной сетки пикселей в другую

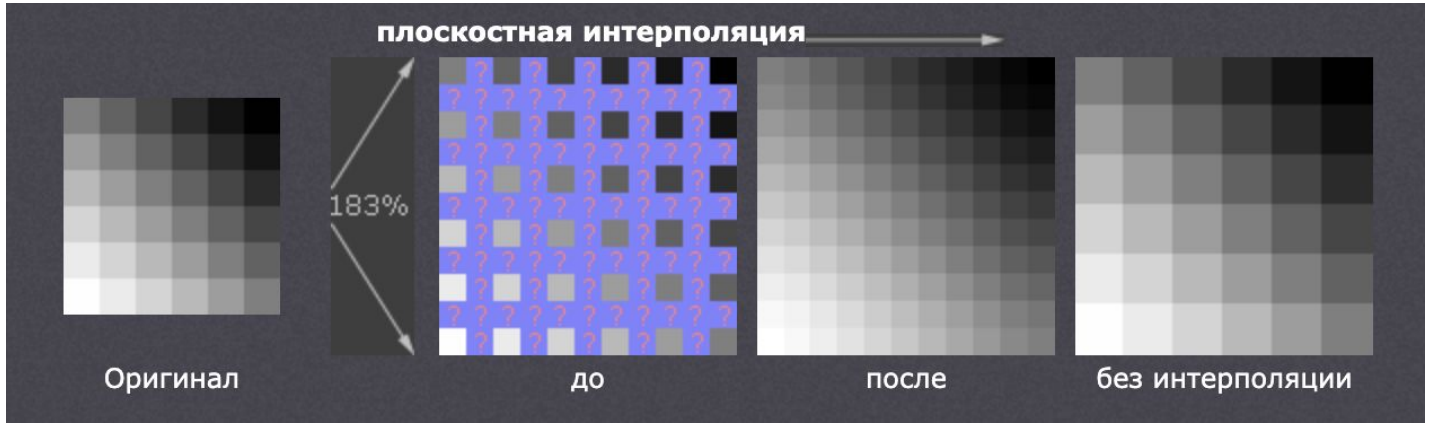




Суть интерполяции заключается в использовании имеющихся данных для получения ожидаемых значений в неизвестных точках. Чем больше мы знаем об окружающих пикселях, тем лучше сработает интерполяция

Интерполяция

Интерполяция изображений работает в двух измерениях и пытается достичь наилучшего приближения в цвете и яркости пикселя, основываясь на значениях окружающих пикселей. Следующий пример иллюстрирует работу масштабирования.

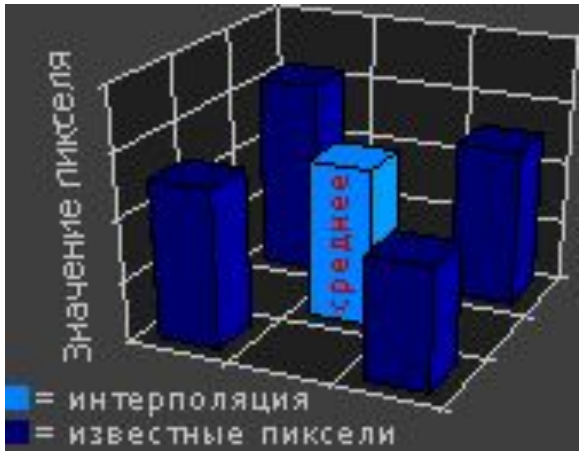


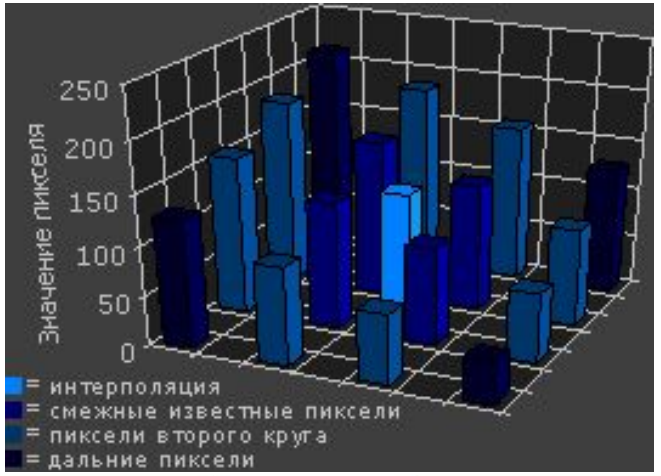
Общепринятые алгоритмы интерполяции можно поделить на две категории: адаптивные и неадаптивные. Адаптивные методы изменяются в зависимости от предмета интерполяции (резкие границы, гладкая текстура), тогда как неадаптивные методы обрабатывают все пиксели одинаково

Масштабирование

3. **Билинейная интерполяция**: Здесь для каждого нового пикселя используется линейное сочетание значений **четырех окружающих пикселей** из исходного изображения. Билинейная интерполяция обеспечивает сглаживание и более гладкий вид изображения, чем метод ближайшего соседа.

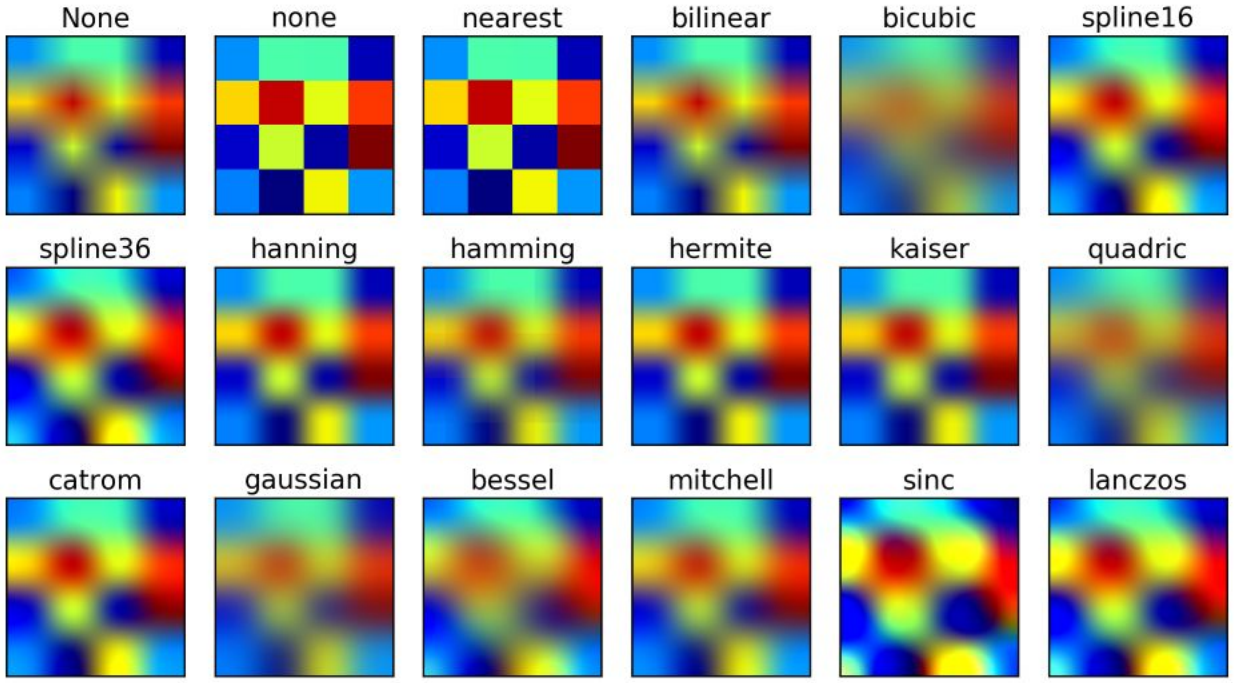
4. **Бикубическая интерполяция**: Этот метод применяет полиномиальную интерполяцию второго порядка, используя **16 соседних пикселей**. Бикубическая интерполяция предоставляет еще более сглаженное изображение, но может быть медленнее, чем билинейная интерполяция.



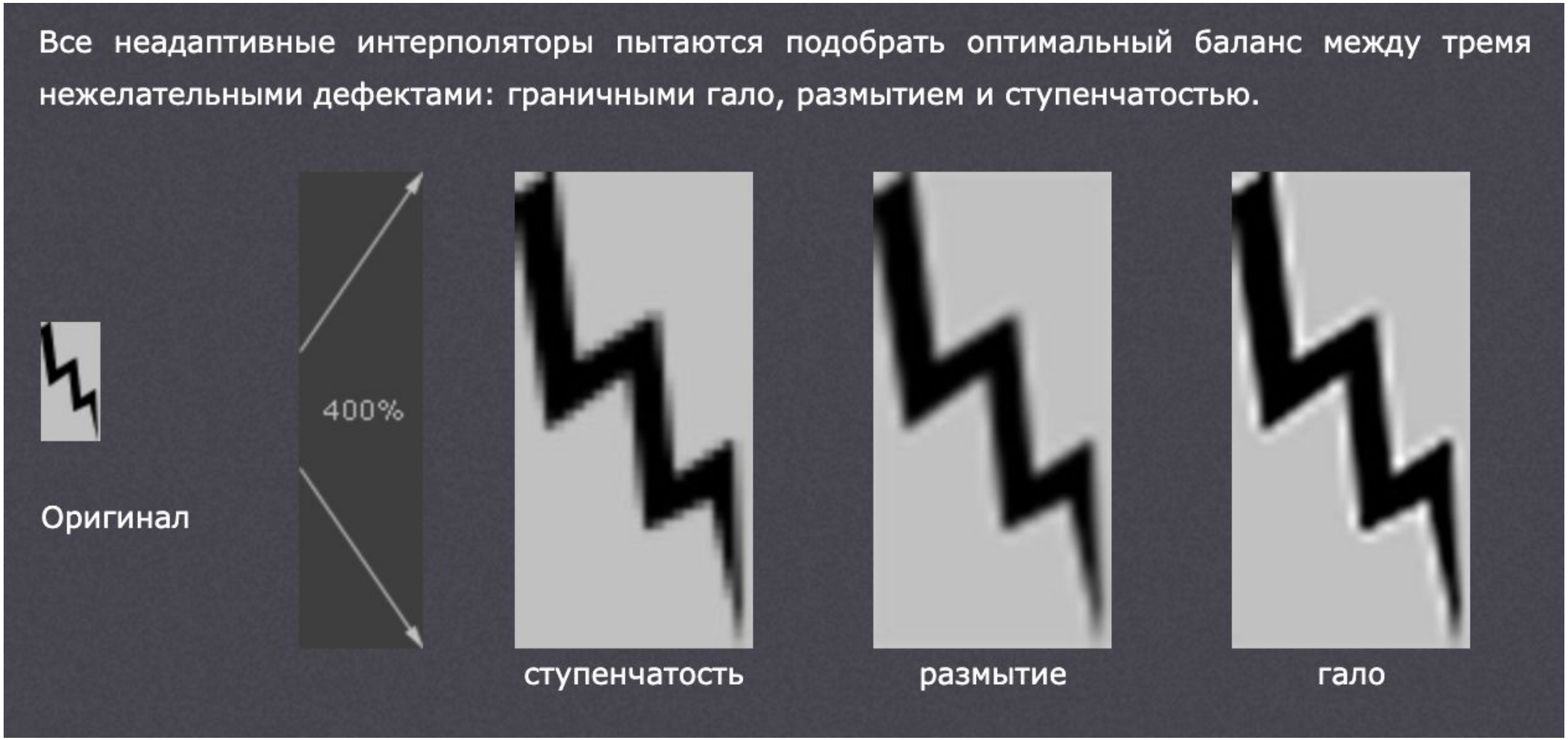


Масштабирование

2. **Lanczos resampling:** Этот алгоритм использует оконную синхронизацию, основанную на функции Ланцоша, для определения весов при интерполяции. Это дает очень хорошие результаты, сохраняя границы объектов и не размывая детали, однако, он более вычислительно затратный



Масштабирование и дефекты



Масштабирование в Open CV

1. Ближайший сосед (Nearest Neighbor) - cv2.INTER_NEAREST
2. Билинейная интерполяция - cv2.INTER_LINEAR.
3. Бикубическая интерполяция - cv2.INTER_CUBIC.
4. Lanczos resampling - cv2.INTER_LANCZOS4.

Чтобы использовать эти алгоритмы масштабирования в OpenCV, вам нужно будет вызвать функцию cv2.resize() с соответствующей константой для параметра interpolation.

```
import cv2

# Загрузка исходного изображения
input_image = cv2.imread('input_image.jpg')

# Задание нового размера изображения
new_width, new_height = 200, 200

# Масштабирование изображения с использованием бикубической интерполяции
resized_image = cv2.resize(input_image, (new_width, new_height), interpolation=cv2.INTER_CUBIC)

# Сохранение полученного изображения
cv2.imwrite('resized_image.jpg', resized_image)
```

Нормализация

Нормализация обеспечивает приведение значений пикселей изображений к определенному диапазону, что улучшает сходимость и стабильность обучения

1. **Минимаксная нормализация:** Этот метод преобразует значения пикселей таким образом, что минимальное значение становится 0, а максимальное – 1. Вычисляем следующим образом: *(image - min_value) / (max_value - min_value)*.

Минимаксная нормализация применяется, когда значения пикселей имеют разный диапазон, и они должны быть приведены к одному масштабу.

```
import cv2

image = cv2.imread('input_image.jpg', cv2.IMREAD_GRAYSCALE)
normalized_image = cv2.normalize(image, None, alpha=0, beta=1, norm_type=cv2.NORM_MINMAX,
dtype=cv2.CV_32F)
```

Нормализация

2. Z-преобразование (стандартизация): Это метод нормализации, который приводит изображение к нулевому среднему и единичному стандартному отклонению. Вычисляем следующим образом: (image - mean) / std_dev.

Z-преобразование полезно, когда важно сохранить структуру данных, такую как контрастность и текстуры, при обработке изображений.

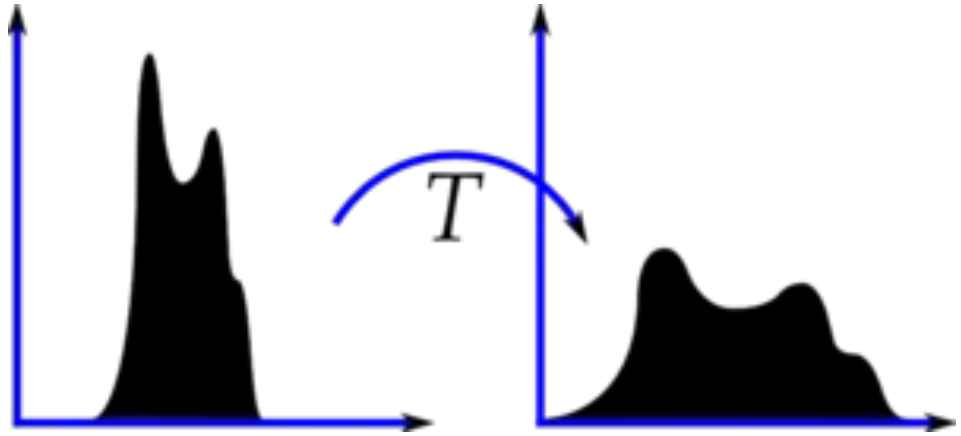
```
import cv2
import numpy as np

image = cv2.imread('input_image.jpg', cv2.IMREAD_GRAYSCALE)
mean, std_dev = cv2.meanStdDev(image)
normalized_image = (image - mean) / std_dev
```

Нормализация

3. Нормализация на основе гистограммы: Этот метод позволяет равномерно распределить значения пикселей по всему доступному диапазону. Применяется гистограмма изображения, после чего выполняется преобразование для получения равномерного распределения.Обычно это улучшает контрастность изображения.

```
img = cv.imread('wiki.jpg', cv.IMREAD_GRAYSCALE)
assert img is not None, "file could not be read, check with os.path.exists()"
equ = cv.equalizeHist(img)
res = np.hstack((img,equ)) #stacking images side-by-side
cv.imwrite('res.png',res)
```

image



Нормализация

4. Нормализация яркости и контраста: Этот метод заключается в корректировке яркости (среднего значения пикселей) и контраста (стандартное отклонение пикселей) изображения. Применяется, когда необходимо улучшить визуальное качество изображений с низким контрастом.

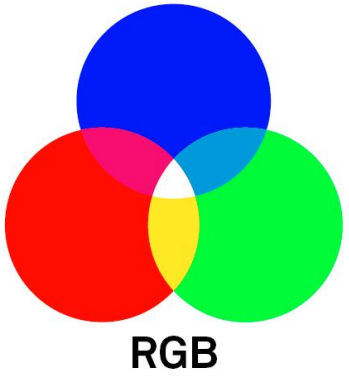
```
import cv2

image = cv2.imread('input_image.jpg', cv2.IMREAD_GRAYSCALE)
alpha = 1.5  # коэффициент контраста
beta = 50    # яркость
contrast_image = cv2.addWeighted(image, alpha, np.zeros(image.shape, image.dtype), 0, beta)
```

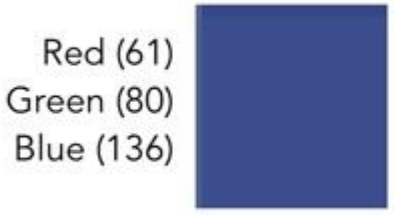
Цветовое преобразование

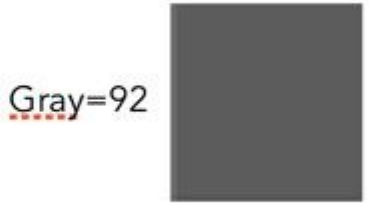
Цветовые преобразования являются важным шагом обработки изображений в компьютерном зрении, так как разные задачи могут требовать различных представлений цвета.

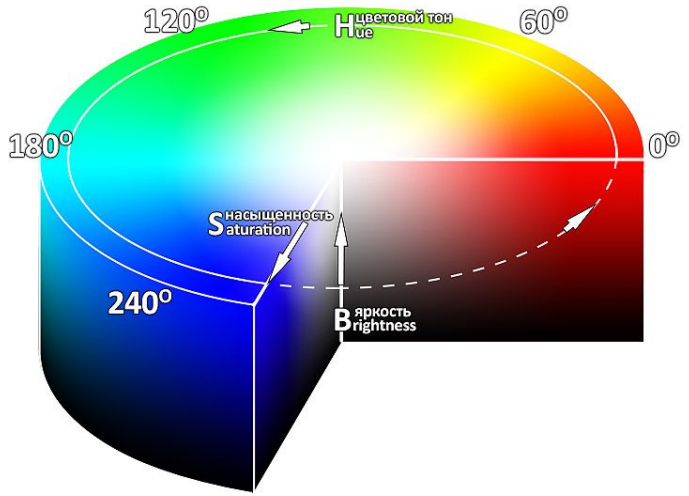
1. Преобразование **в оттенки серого (Grayscale)**: Преобразование цветного изображения в черно-белое путем усреднения значений каналов RGB. Это упрощает обработку изображений и уменьшает количество данных.

2. Преобразование **RGB в HSV (Hue, Saturation, Value)**: Преобразование цветового пространства RGB в HSV для лучшего представления цветовой информации. HSV разделяет информацию о цветовых оттенках, насыщенности и яркости, что может быть полезно при работе с цветами.

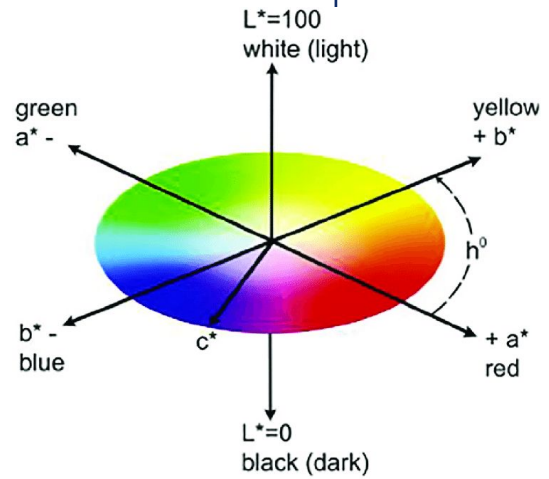






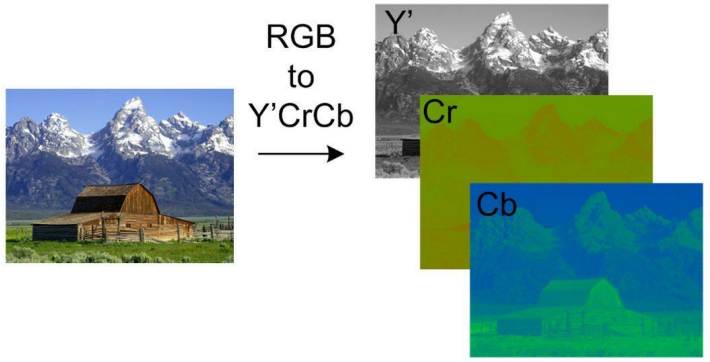


Цветовое преобразование

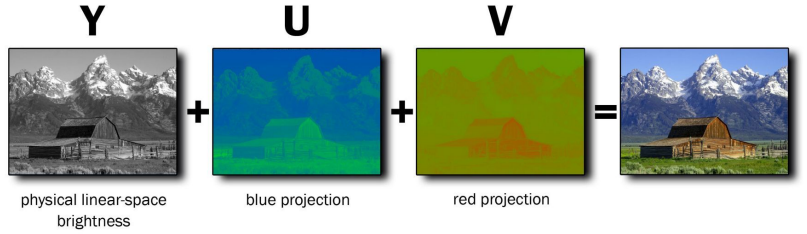


3. Преобразование **RGB в Lab (CIELAB)**: Преобразование из цветового пространства RGB в Lab, которое дает перцептивно равномерное представление цветов. Lab делит информацию на яркостный канал (L) и два цветовых канала (a и b).

4. Преобразование **RGB в YCbCr**: Преобразование из цветового пространства RGB в YCbCr, которое обеспечивает разделение яркостной информации (Y) от цветовой информации (Cb и Cr). Это полезно при сжатии изображений и видео.



5. Преобразование **RGB в YUV**: Преобразование из цветового пространства RGB в YUV, которое также разделяет яркостную информацию (Y) от цветовой информации (U и V). YUV часто используется в системах видео и телевизорах.



Цветовое преобразование в Open CV

```
import cv2


image = cv2.imread('input_image.jpg')
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
```

Для преобразования цвета мы используем функцию cv.cvtColor(input_image, flag), где флаг определяет тип преобразования.

cv2.COLOR_BGR2GRAY

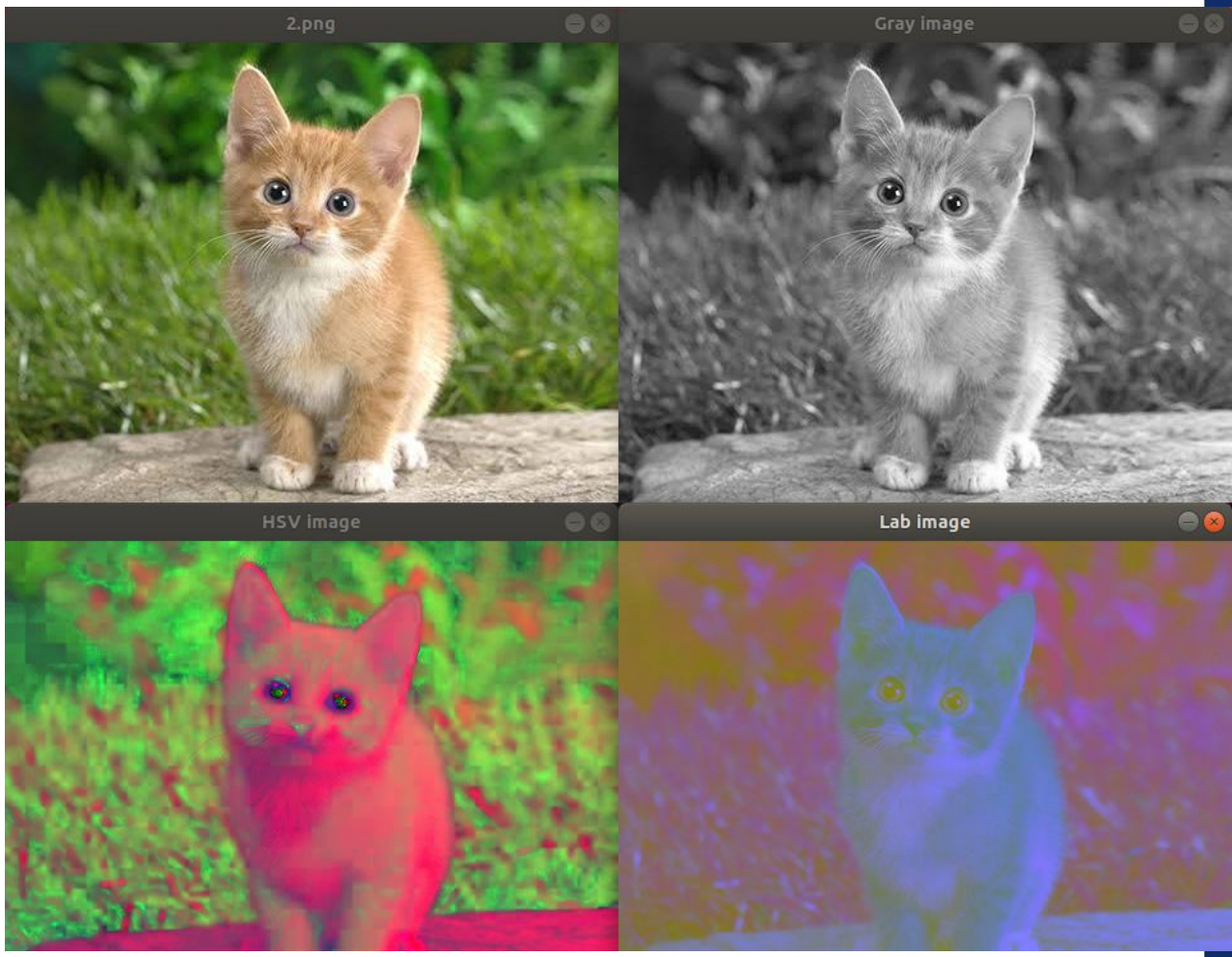
cv2.COLOR_BGR2RGB

cv2.COLOR_BGR2HSV

cv2.COLOR_BGR2HLS

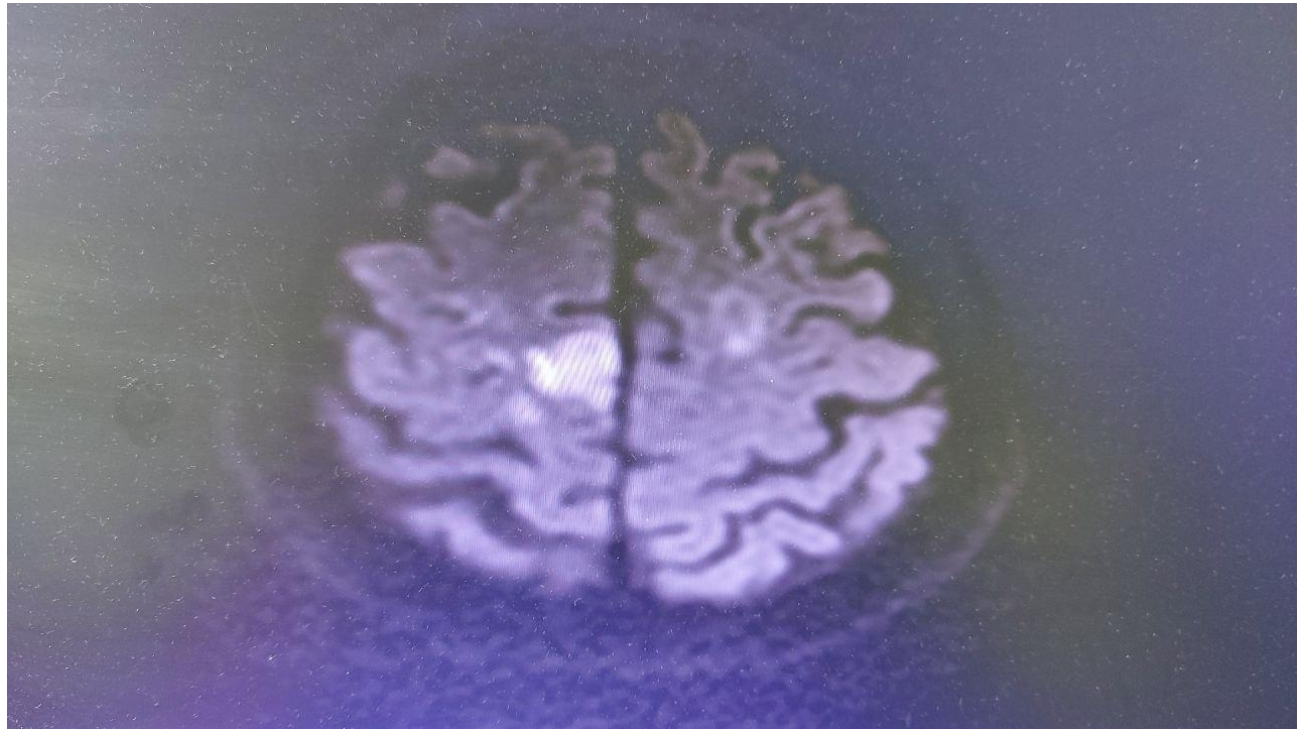
cv2.COLOR_BGR2Lab

cv2.COLOR_BGR2YUV



Устранение шума

– это процесс сглаживания или фильтрации изображений для уменьшения случайных вариаций в значениях пикселей. Это может существенно улучшить качество изображения и сделать его более пригодным для дальнейшей обработки и анализа



- **Блюр (размытие)** часто используется для сглаживания неравномерных значений пикселей изображения, обрезая самые высокие значения. В алгоритмах компьютерного зрения, данный метод используется для улучшения структуры изображения в различных масштабах

- **Морфологическая обработка** изображений - Этот метод позволяет удалить дефекты из бинарных изображений. Иногда области, созданные в результате простой обработки, могут быть искажены шумом. Морфологическая же обработка позволяет сглаживать изображения.

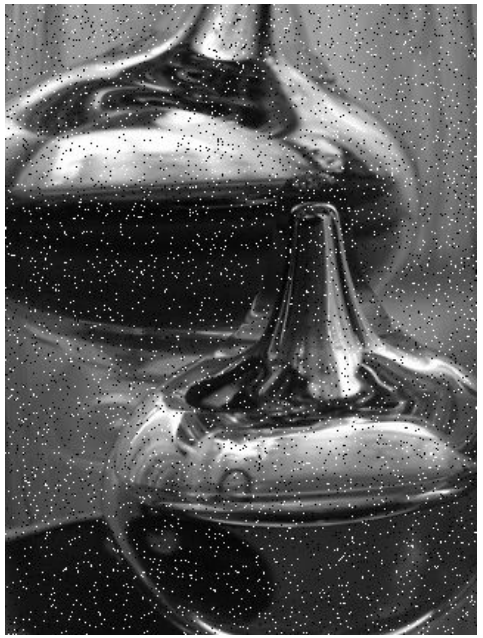
Размытие

1. Сглаживание с использованием среднего фильтра (**Mean filter**): Применение скользящего окна (ядра) к изображению и замена каждого пикселя средним значением соседних пикселей. Это простой и быстрый метод для устранения шума.

2. Сглаживание с использованием медианного фильтра (**Median filter**): Применение скользящего окна к изображению и замена каждого пикселя медианным значением соседних пикселей. Медианный фильтр хорошо справляется с шумом “соли и перца”.

3. Гауссовское сглаживание (**Gaussian filtering**): Применение ядра Гаусса к изображению для взвешенного усреднения соседних пикселей.

4. **Non-local Means Denoising**



шум “соли и перца”

шумная картинка

под фильтром Гаусса

Морфологическая обработка изображений

Обычно выполняется для ЧБ изображений (!)

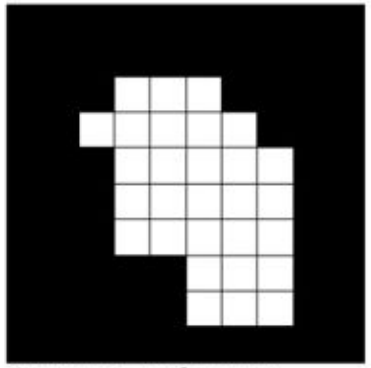
Морфологические операции преобразуют изображение, используя некоторый шаблон или фильтр. Он помещается в различные места на изображении и сравнивает между собой разные области с группами пикселей, преобразуя их в зависимости от задачи.

Морфологическая обработка изображений

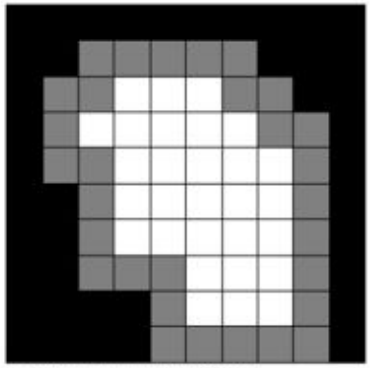
Есть два основных вида морфологических преобразований:

- дилатация (морфологическое расширение) – свертка изображения (здесь элемент пикселя равен «1», если хотя бы один пиксель в “шаблоне” равен «1»). Такая операция вызывает рост светлых областей на изображении;

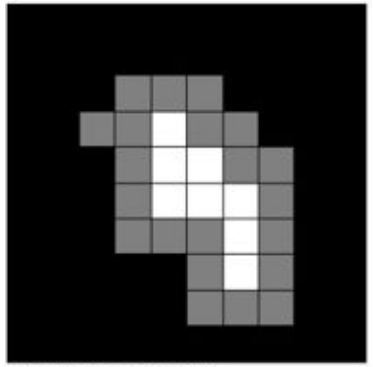
- эрозия (морфологическое сужение) – операция обратная дилатации. Эта операция вызывает уменьшение светлых областей на изображении



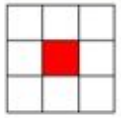
a) исходное изображение



b) результат дилатации



c) результат эрозии



d) шаблон (центр – ведущий элемент)

Расширение	эрозия
Это увеличивает размер объектов.	Это уменьшает размер объектов.
Он заполняет отверстия и поврежденные участки.	Убирает мелкие аномалии.
Он соединяет области, которые разделены промежутком меньше, чем структурирующий элемент.	Уменьшает яркость ярких объектов.
Увеличивает яркость объектов.	Он удаляет объекты меньшего размера, чем элемент структурирования.

Морфологическая обработка VS Размытие

Фокус на модификации геометрических структур и форм на изображении

```
# определение ядра свертки
morph_kernel = np.ones((3, 3))

# применение функций к изображению
# параметр iterations означает, сколько раз будет применена операция
dilate_img = cv2.dilate(img, kernel= morph_kernel, iterations=1)
erode_img = cv2.erode(img, kernel= morph_kernel, iterations=1)
```

направлено на уменьшение шума и детализации путем сглаживания интенсивности пикселей

```
img = cv2.imread('distorted.png')

# параметром ksize=(11, 11) зададим размер ядра фильтра размытия 11x11 пикселей:
# параметры sigmaX/Y 0, 0 отвечают за сдвиг ядра при проходе по осям X, Y
blurred_img = cv2.GaussianBlur(img, ksize=(11, 11), sigmaX =0, sigmaY=0)
```

Оба подхода могут быть использованы совместно для достижения определенных целей и улучшения качества моделей

Определение границ объектов на изображении

1) **0) Приведение к оттенкам серого (!)**
2) **Сглаживание**. Поскольку обнаружение краев чувствительно к шуму на изображении, первым шагом является удаление шума на изображении с помощью фильтра Гаусса 5x5

3) **Расчет градиента**. Применяется фильтр Собеля как в горизонтальном, так и в вертикальном направлении, чтобы получить первую производную в горизонтальном направлении (Gx) и вертикальном направлении (Gy). Из этих двух изображений мы можем найти градиент края и направление для каждого пикселя следующим образом:

$$Edge\_Gradient\ (G) = \sqrt{G_x^2 + G_y^2}$$

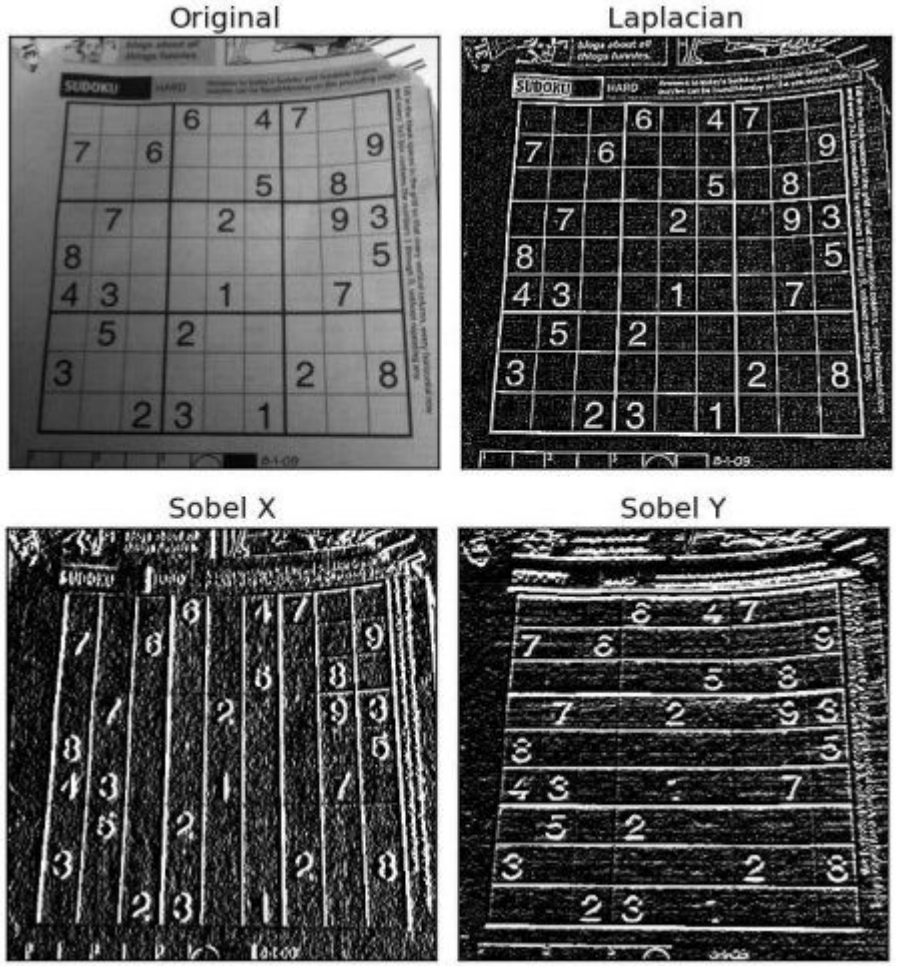
$$Angle\ (\theta) = \tan^{-1}\left(\frac{G_y}{G_x}\right)$$

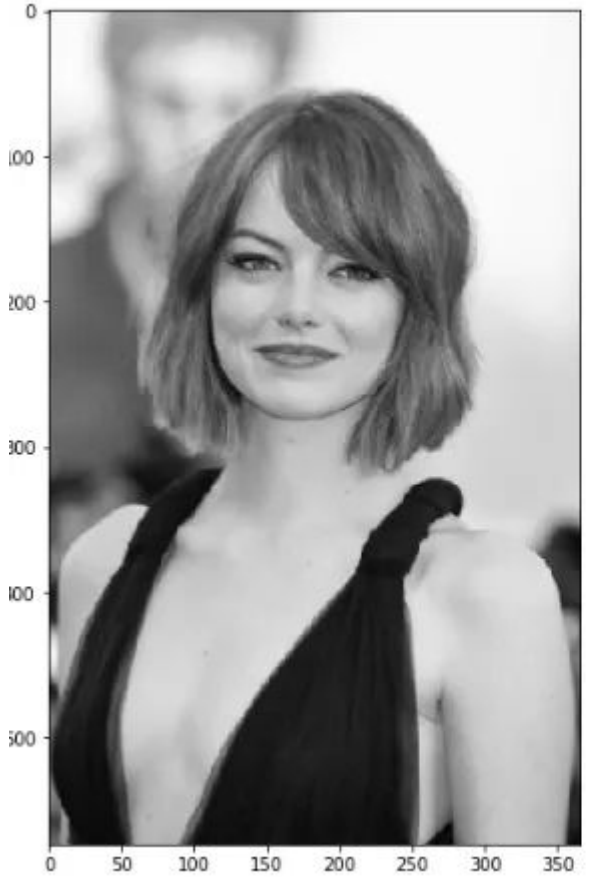
Gradient direction is always perpendicular to edges. It is rounded to one of four angles representing vertical, horizontal and two diagonal directions.

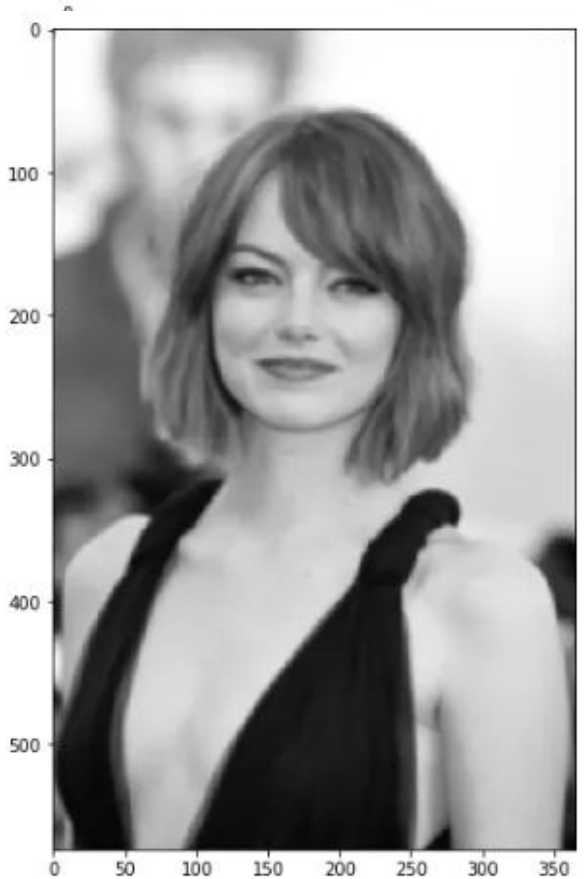
Определение границ объектов на изображении

OpenCV предоставляет три типа градиентных фильтров или фильтров верхних частот:

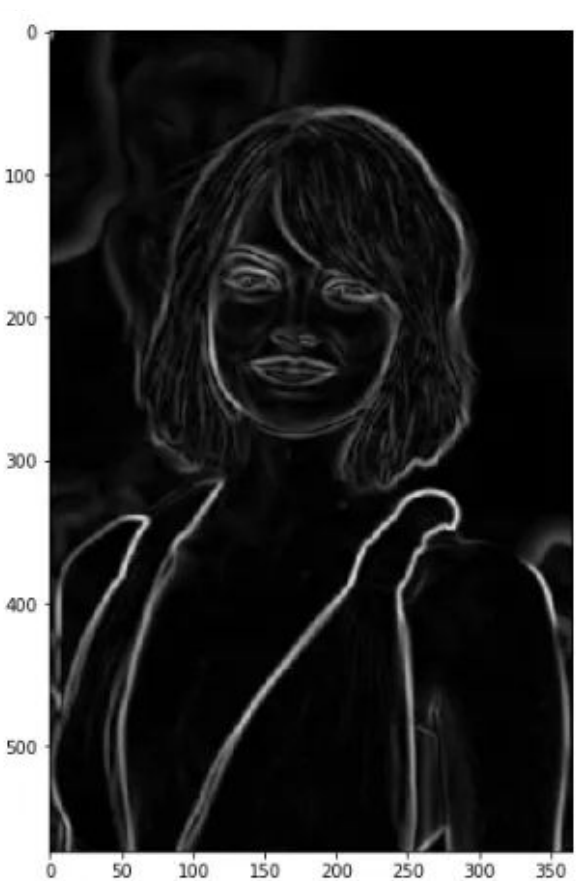
- Sobel
- Scharr
- Laplacian







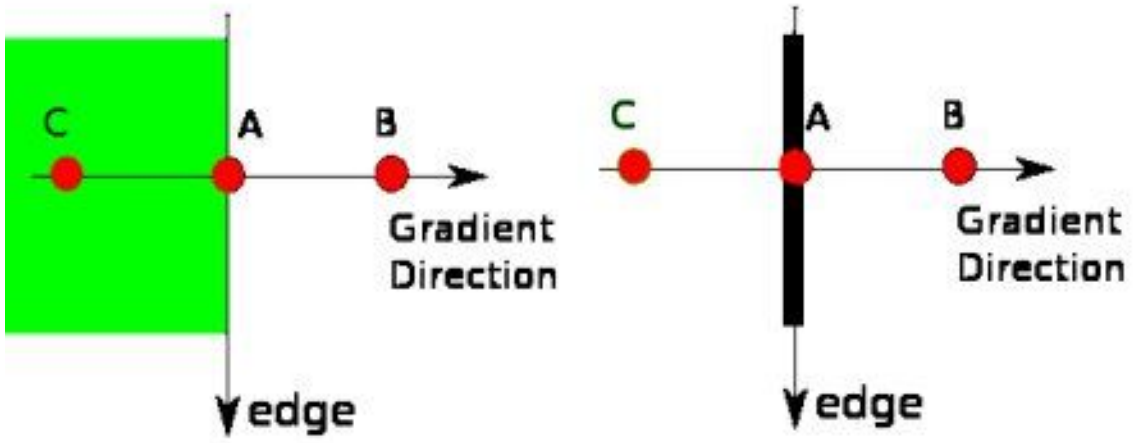
Сглаживание



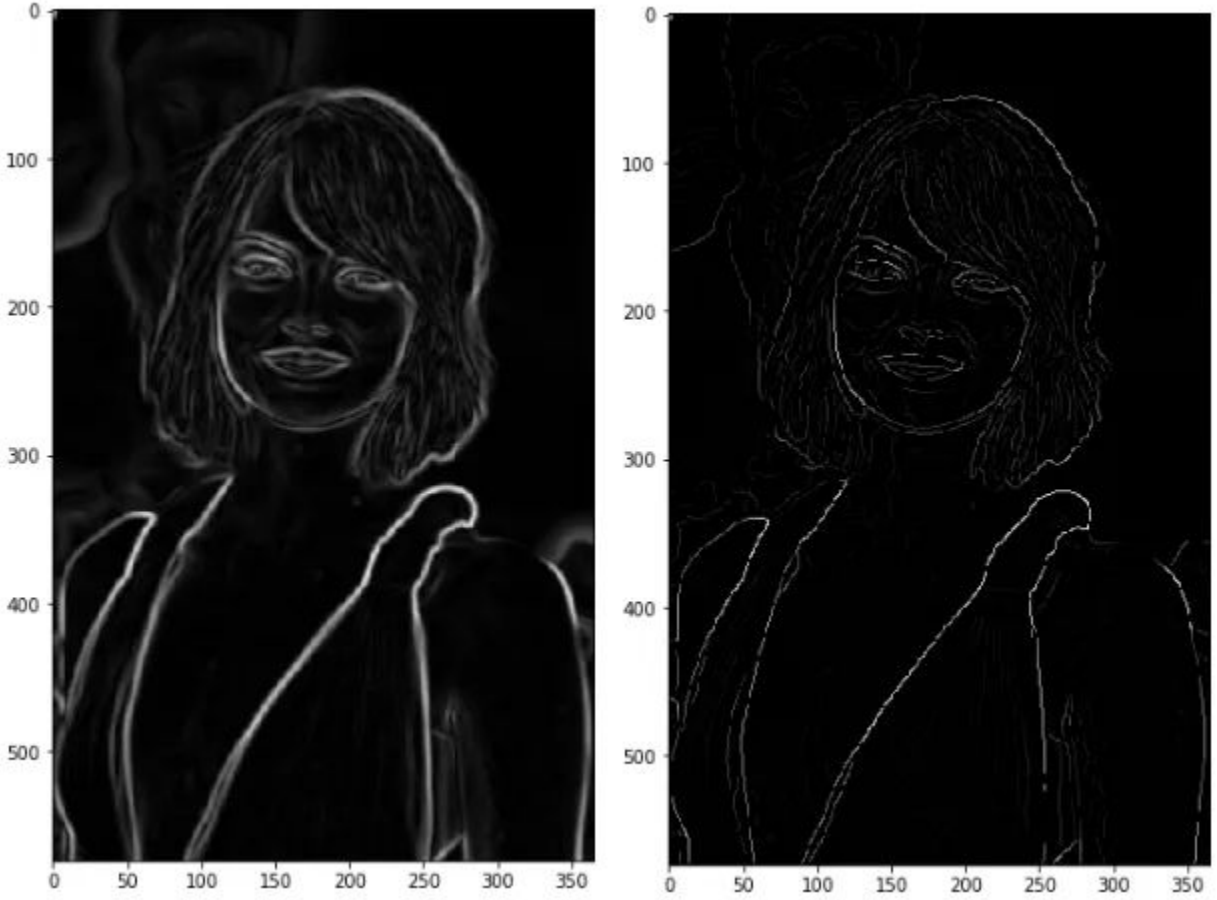
Интенсивность градиента

Определение границ объектов на изображении

3) **Немаксимальное подавление**. После получения величины и направления градиента выполняется полное сканирование изображения, чтобы удалить любые нежелательные пиксели, которые могут не составлять края. Для этого у каждого пикселя проверяется является ли он локальным максимумом в своей окрестности в направлении градиента:



В результате получится бинарное изображение с «тонкими краями»

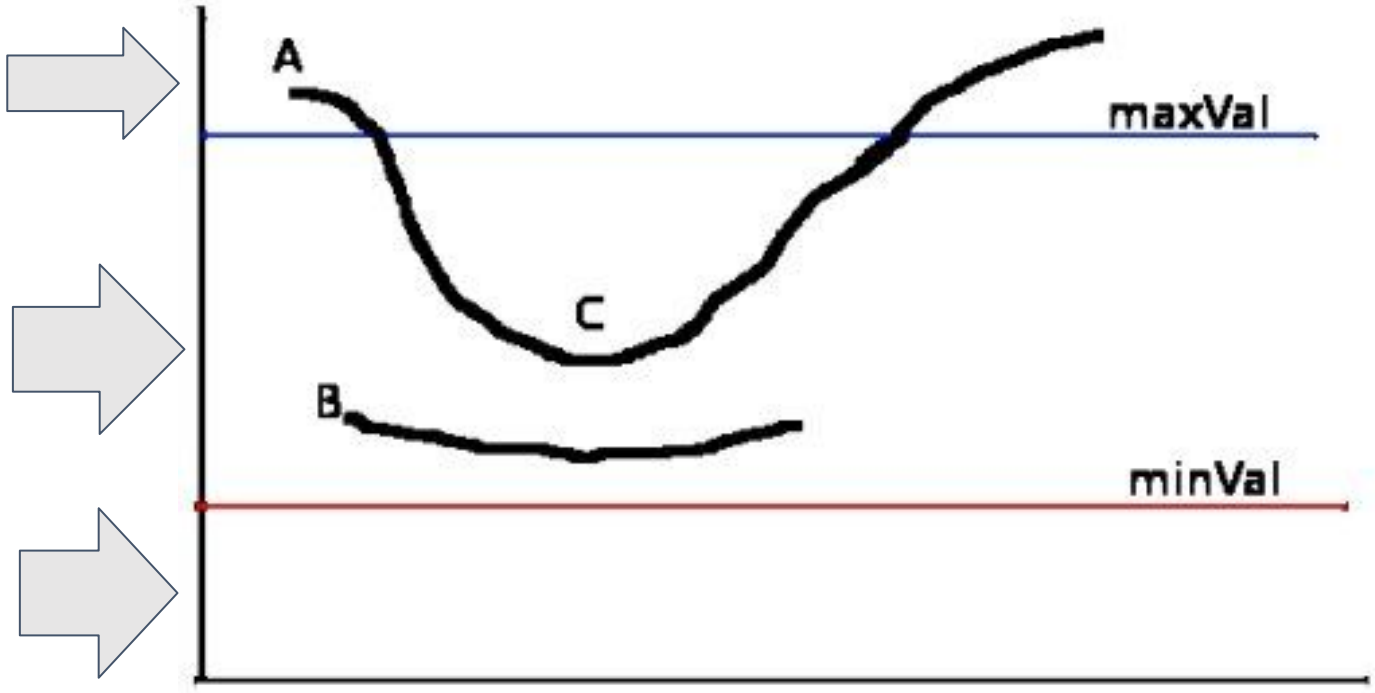


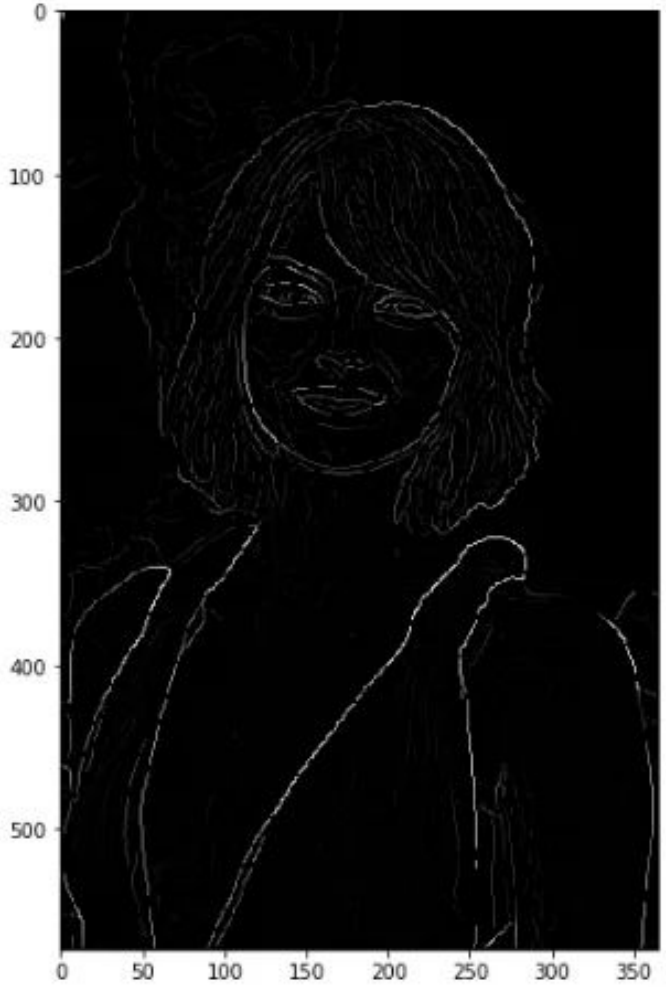
Немаксимальное подавление

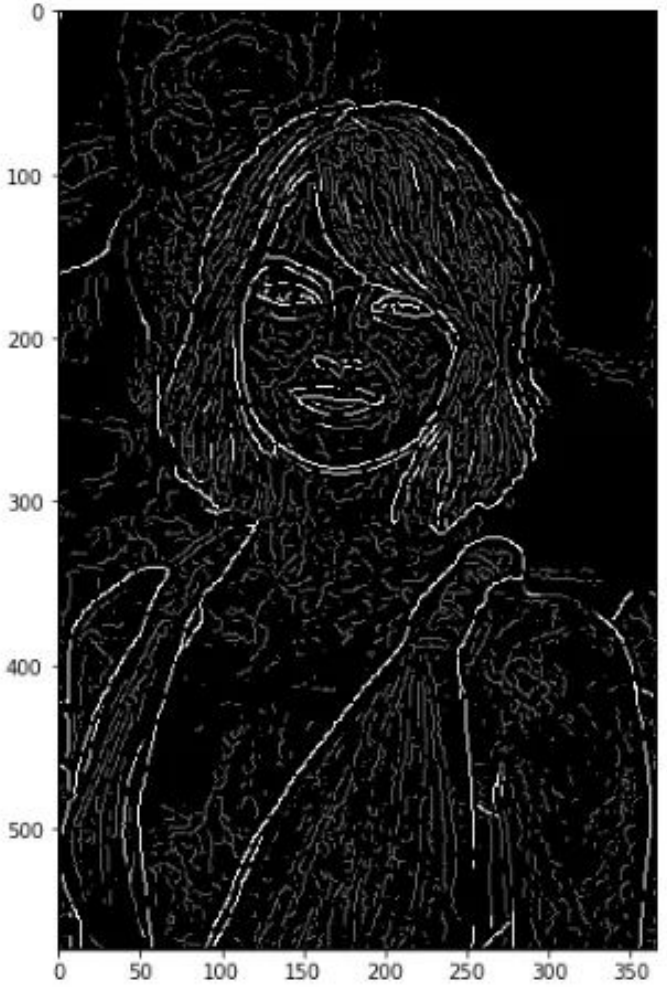
Определение границ объектов на изображении

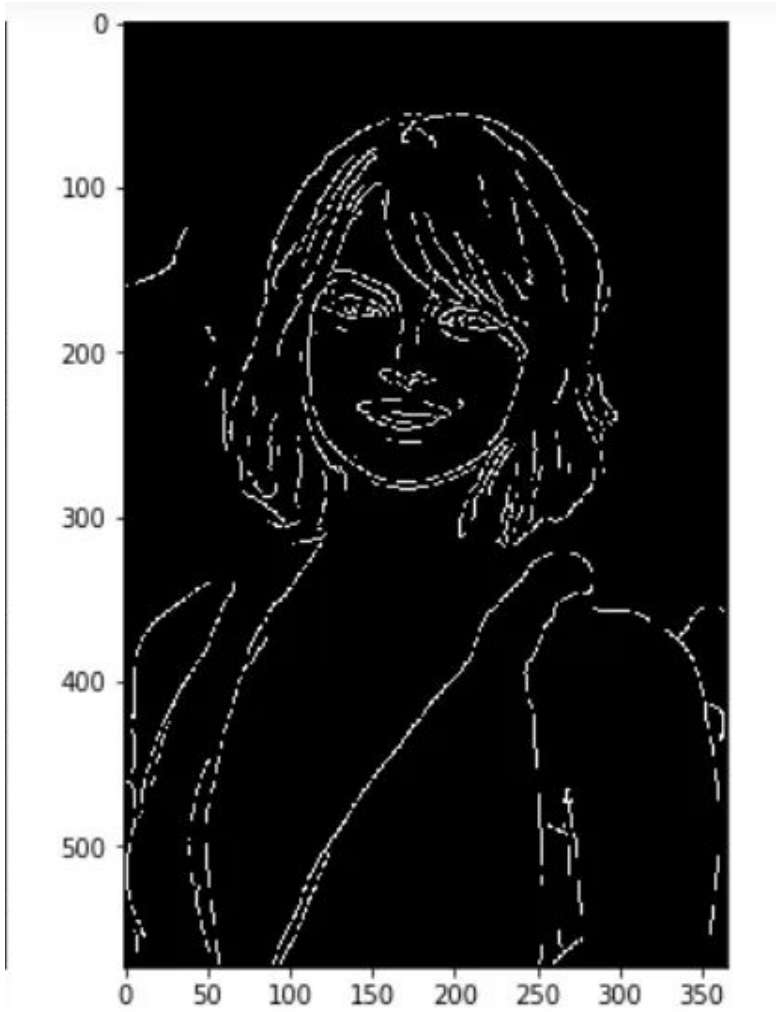
4) Шаг двойного порога и гистерезис:

- **Сильные пиксели** — это пиксели, интенсивность которых настолько высока, что мы уверены, что они вносят свой вклад в окончательный край.
- **Слабые пиксели** — это пиксели, значение интенсивности которых недостаточно, чтобы считаться сильными, но все же недостаточно мало, чтобы считаться нерелевантными для обнаружения границ.
- Остальные пиксели считаются нерелевантными для края.

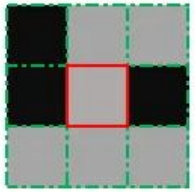




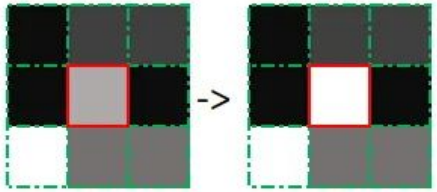




Сильные и слабые пиксели



No strong pixels around



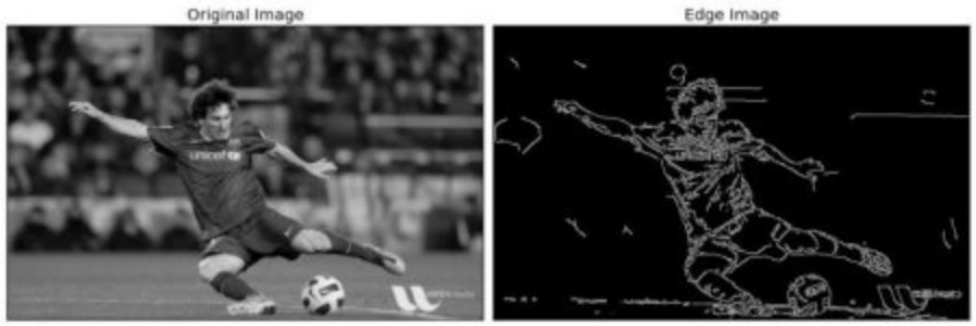
One strong pixel around

Результаты процесса гистерезиса

Canny в Open CV

```
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt

img = cv.imread('messi5.jpg', cv.IMREAD_GRAYSCALE)
assert img is not None, "file could not be read, check with os.path.exists()"
edges = cv.Canny(img,100,200)

plt.subplot(121),plt.imshow(img,cmap = 'gray')
plt.title('Original Image'), plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(edges,cmap = 'gray')
plt.title('Edge Image'), plt.xticks([]), plt.yticks([])

plt.show()
```

See the result below:



image

Выделение ключевых точек

Ключевые точки - это информативные и уникальные области изображения, которые могут быть использованы для различных задач, таких как сопоставление изображений и object detection

Алгоритмы для выделения ключевых точек, которые есть в Open CV:

1. **SIFT** (Scale-Invariant Feature Transform) - https://docs.opencv.org/master/da/df5/tutorial_py_sift_intro.html

2. **SURF** (Speeded-Up Robust Features): - https://docs.opencv.org/master/df/dd2/tutorial_py_surf_intro.html

3. **ORB** (Oriented FAST and Rotated BRIEF) - https://docs.opencv.org/master/d1/d89/tutorial_py_orb.html

4. **KAZE** (KAZE Features) - https://docs.opencv.org/master/d3/d61/tutorial_py_kaze_start.html

5. **AKAZE** (Accelerated-KAZE) - https://docs.opencv.org/master/d8/d30/tutorial_py_akaze_start.html

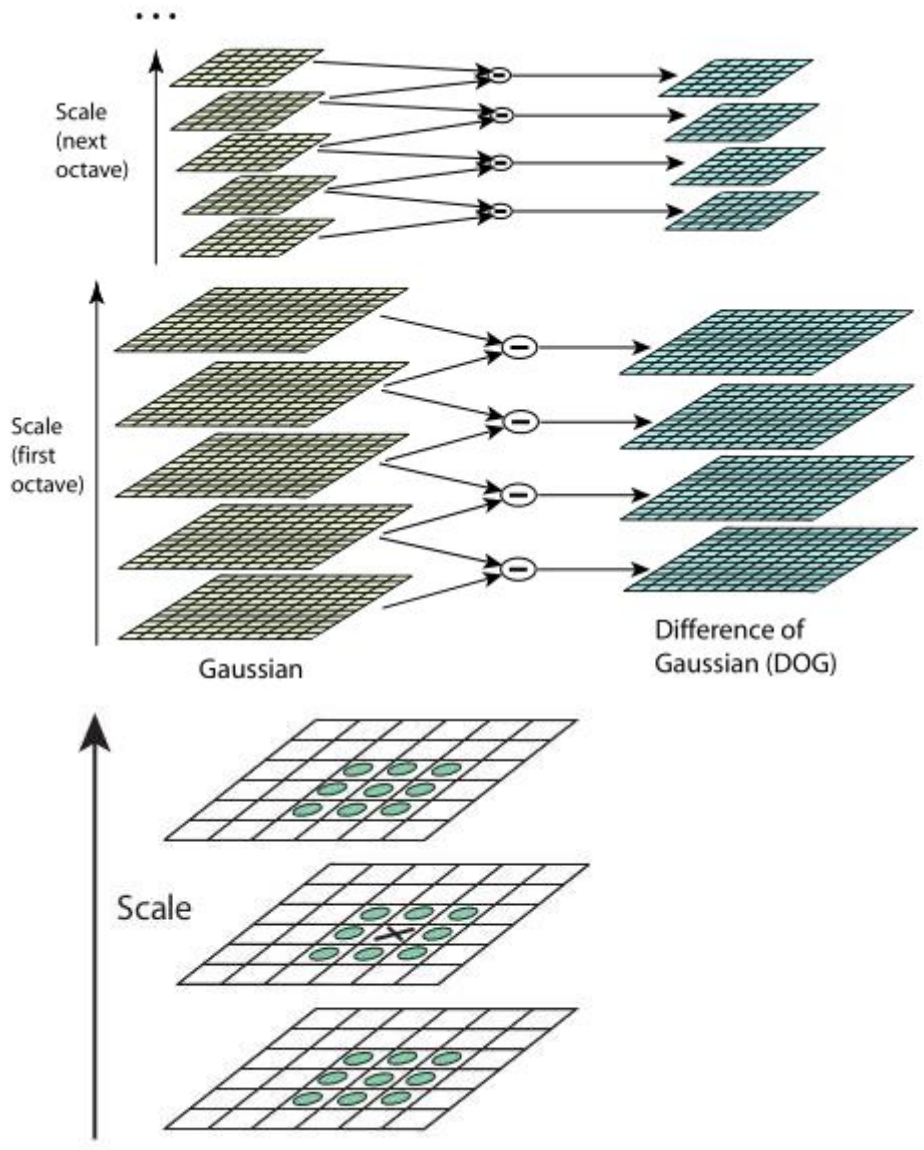
6. **BRISK** (Binary Robust Invariant Scalable Keypoints) - https://docs.opencv.org/master/dc/d29/tutorial_py_brisk_start.html

Выделение ключевых точек - SIFT (2004)

1) Нахождение особых точек - построение пирамиды гауссианов (Gaussian) и разностей гауссианов (Difference of Gaussian, DoG). Разностью гауссианов называют изображение, полученное путем попиксельного вычитания одного гауссиана исходного изображения из гауссиана с другим радиусом размытия

$$\begin{aligned} D(x,y,\sigma) &= (G(x,y,k\sigma) - G(x,y,\sigma)) * I(x,y) \\ &= L(x,y,k\sigma) - L(x,y,\sigma). \end{aligned}$$

Как только эта DoG найдена, изображения ищутся на предмет локальных экстремумов по масштабу и пространству.



Выделение ключевых точек - SIFT

2) Локализация ключевых точек

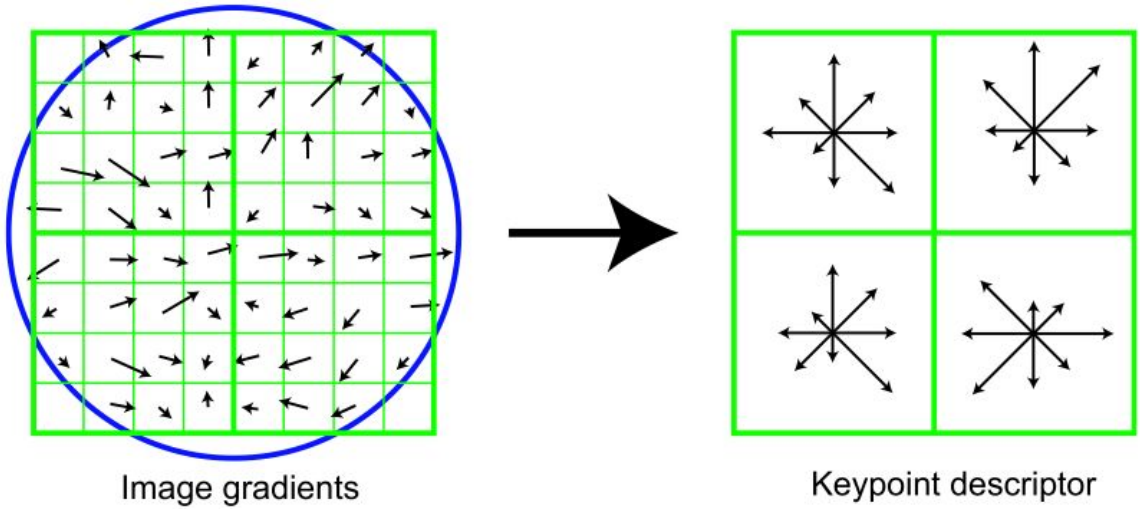
Как только потенциальные ключевые точки найдены, их необходимо уточнить, чтобы получить более точные результаты. Они использовали разложение масштабного пространства в ряд Тейлора, чтобы получить более точное местоположение экстремумов, и если интенсивность в этих экстремумах меньше порогового значения (0,03 согласно статье), оно отбрасывается. Этот порог называется counterThreshold в OpenCV. Устраняют ключевые точки и краевые ключевые точки, и остаются только точки сильного интереса.

3) Назначение ориентации

Теперь каждой ключевой точке назначается ориентация (Направление ключевой точки вычисляется исходя из направлений градиентов точек, соседних с особой) для достижения инвариантности к вращению изображения. Окрестность берется вокруг местоположения ключевой точки в зависимости от масштаба, и в этой области вычисляются величина и направление градиента.

4) Дескриптор ключевой точки

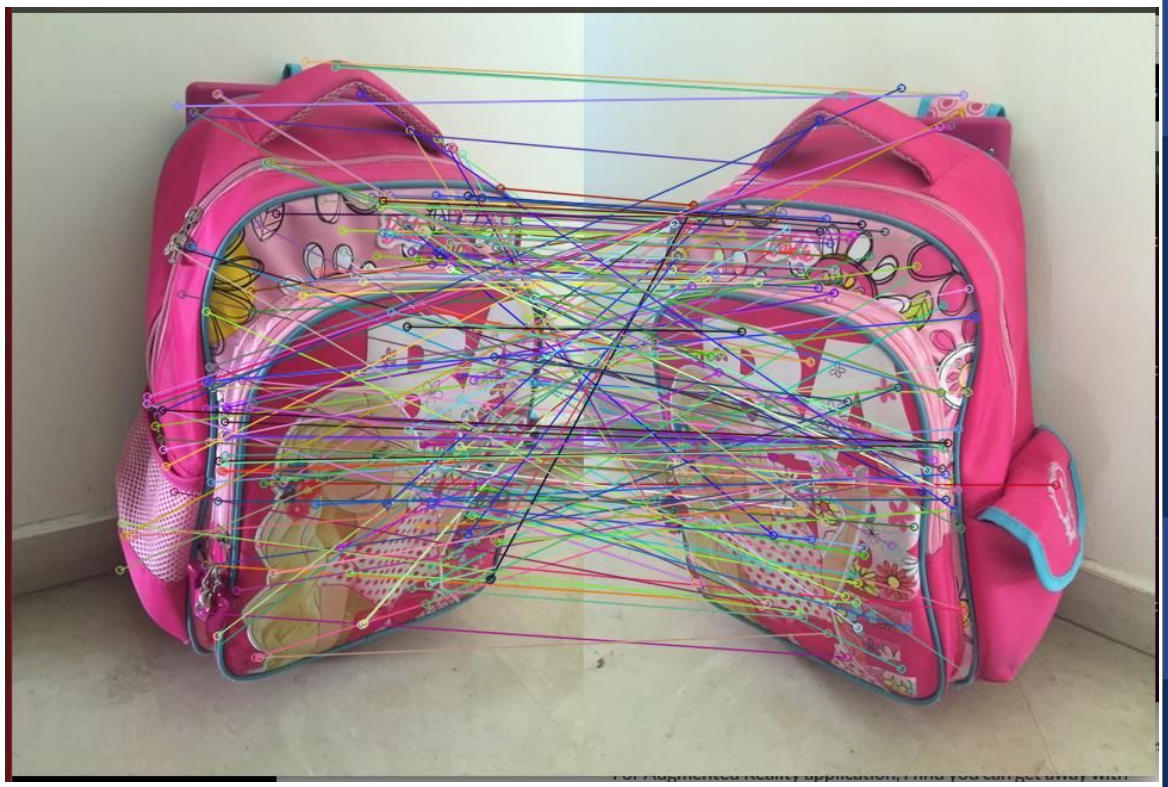
Теперь создан дескриптор ключевой точки. Берется окрестность 16x16 вокруг ключевой точки. Он разделен на 16 подблоков размером 4x4. Для каждого подблока создается гистограмма ориентации с 8 бинами. Таким образом, всего доступно 128 значений бинов. Он представлен в виде вектора для формирования дескриптора ключевой точки. В дополнение к этому, предпринимаются некоторые меры для обеспечения устойчивости к изменениям освещения, вращению и т. д.



Выделение ключевых точек - SIFT

5) Сопоставление ключевых точек

Ключевые точки между двумя изображениями сопоставляются путем определения их ближайших соседей.



Но в некоторых случаях второе ближайшее совпадение может быть очень близко к первому. Это может произойти из-за шума или по другим причинам. В этом случае берется отношение ближайшего расстояния ко второму ближайшему расстоянию. Если он больше 0,8, они отбраковываются. Согласно документу, он устраняет около 90% ложных совпадений и отбрасывает только 5% правильных совпадений.

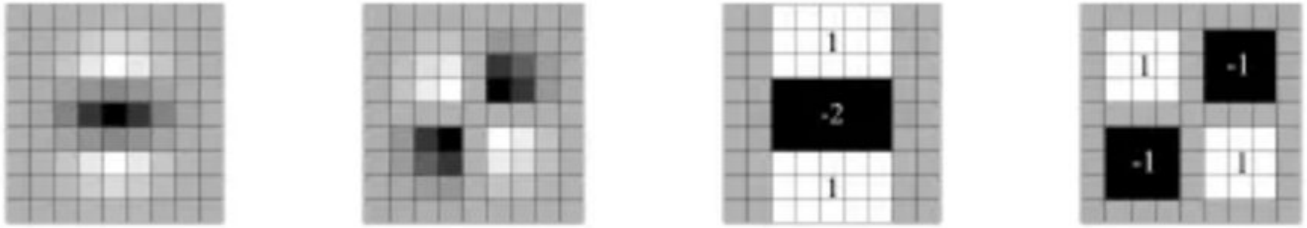
SIFT в Open CV

```
import numpy as np
import cv2 as cv

img = cv.imread('home.jpg')
gray= cv.cvtColor(img,cv.COLOR_BGR2GRAY)

sift = cv.SIFT_create()
kp = sift.detect(gray,None)

img=cv.drawKeypoints(gray,kp,img)

cv.imwrite('sift_keypoints.jpg',img)
```

Выделение ключевых точек - SURF (2006)

SURF был создан как усовершенствование SIFT в 2006 году, направленное на увеличение скорости работы алгоритма.

Вместо того, чтобы использовать разность гауссова для аппроксимации LoG, SURF использует **Box Filters** . Преимущество этого заключается в том, что блочные фильтры можно легко рассчитать , а расчеты для разных масштабов можно выполнять одновременно.



Два примера гауссовых частных производных второго порядка (слева) и фильтров соответствующего поля (справа)

SURF в Open CV

```
>>> img = cv.imread('fly.png', cv.IMREAD_GRAYSCALE)

# Create SURF object. You can specify params here or later.
# Here I set Hessian Threshold to 400
>>> surf = cv.xfeatures2d.SURF_create(400)

# Find keypoints and descriptors directly
>>> kp, des = surf.detectAndCompute(img,None)

>>> len(kp)
 699
```

1199 keypoints is too much to show in a picture. We reduce it to some 50 to draw it on an image. While matching, we may need all those features, but not now. So we increase the Hessian Threshold.

```
# Check present Hessian threshold
>>> print( surf.getHessianThreshold() )
400.0

# We set it to some 50000. Remember, it is just for representing in picture.
# In actual cases, it is better to have a value 300-500
>>> surf.setHessianThreshold(50000)

# Again compute keypoints and check its number.
>>> kp, des = surf.detectAndCompute(img,None)

>>> print( len(kp) )
47
```

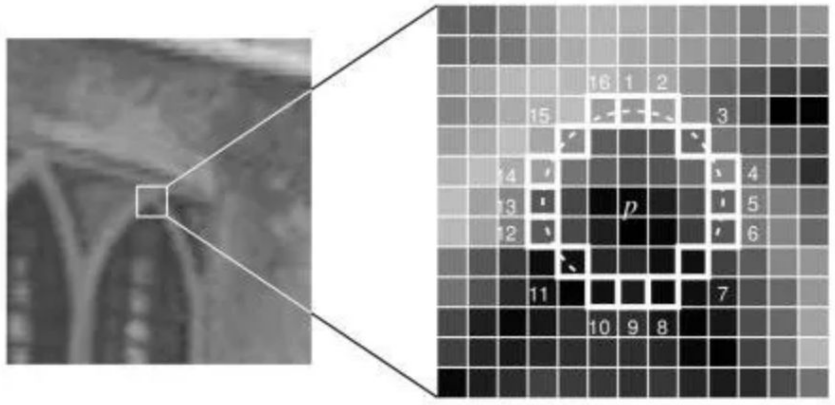
It is less than 50. Let's draw it on the image.

```
>>> img2 = cv.drawKeypoints(img,kp,None,(255,0,0),4)

>>> plt.imshow(img2),plt.show()
```

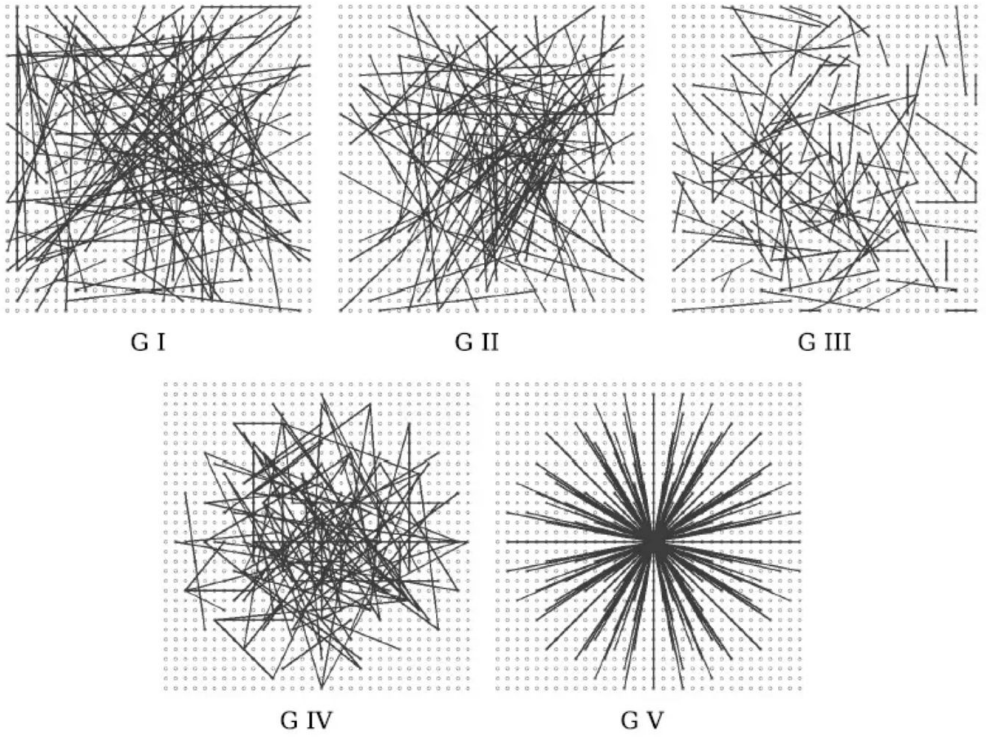
image

Выделение ключевых точек - ORB (2011)

ORB представляет собой комбинацию двух алгоритмов FAST и BRIEF и был создан как альтернатива SIFT и SURF в 2011 году.



FAST - для поиска особых точек

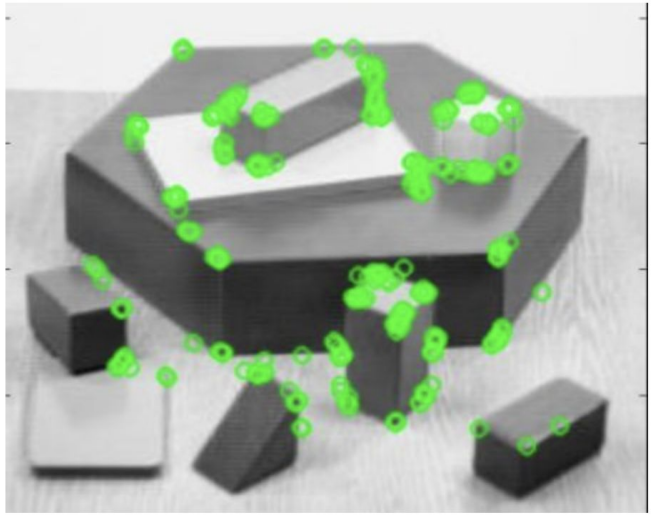


BRIEF - дескриптор

ORB в Open CV

```
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt

img = cv.imread('simple.jpg', cv.IMREAD_GRAYSCALE)

# Initiate ORB detector
orb = cv.ORB_create()

# find the keypoints with ORB
kp = orb.detect(img,None)

# compute the descriptors with ORB
kp, des = orb.compute(img, kp)

# draw only keypoints location,not size and orientation
img2 = cv.drawKeypoints(img, kp, None, color=(0,255,0), flags=0)
plt.imshow(img2), plt.show()
```

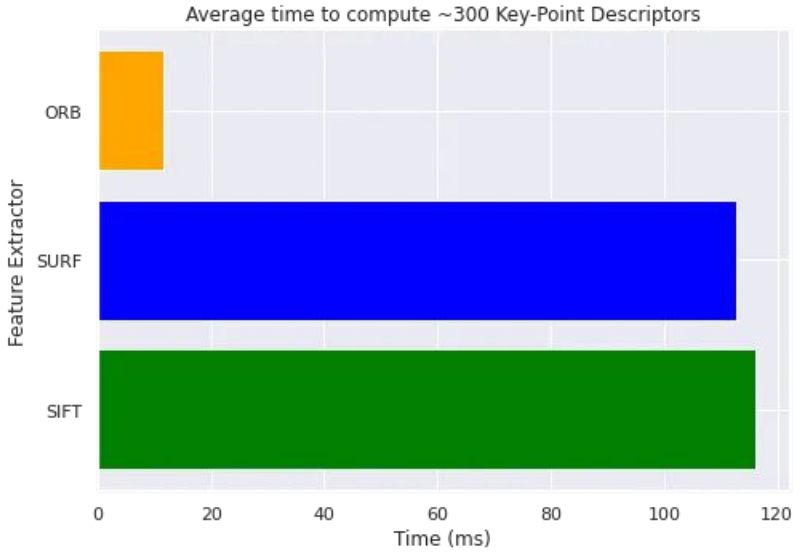
See the result below:

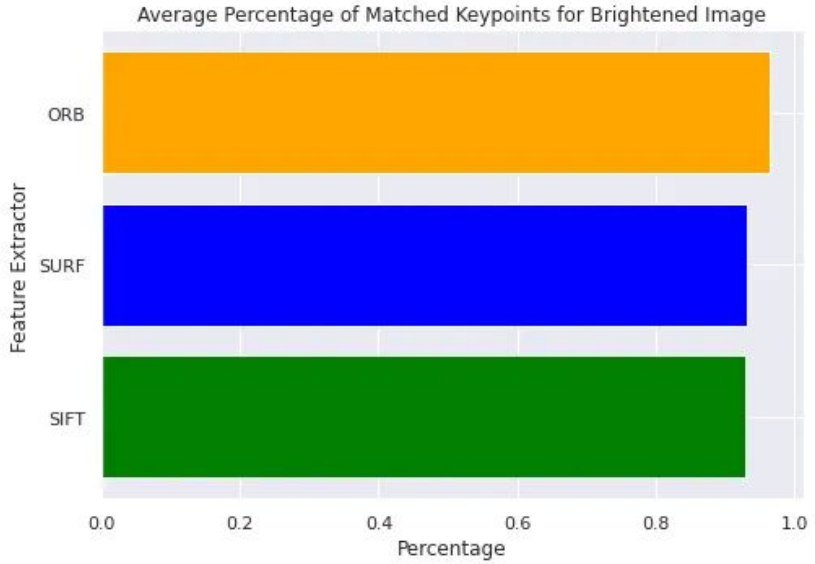


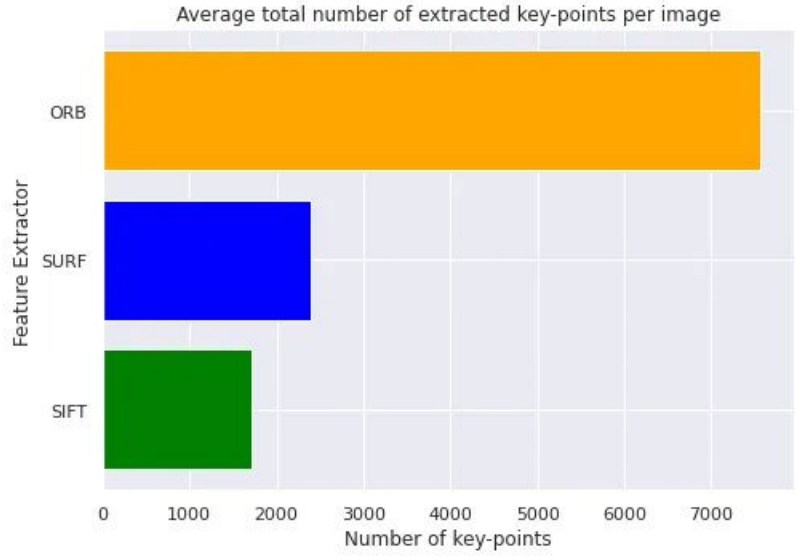
image

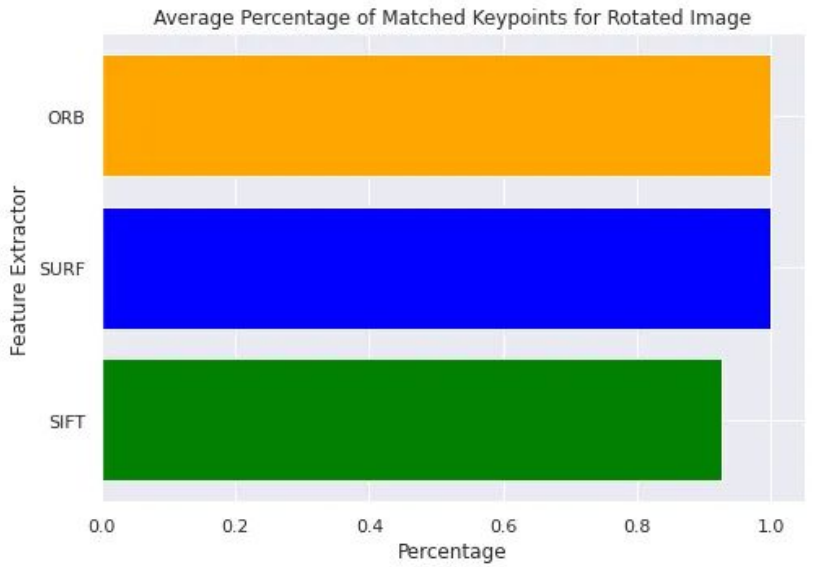
SIFT vs SURF vs ORB

- Здесь мы видим, что ORB может извлекать наибольшее количество ключевых точек на изображение, почти в три раза больше, чем SURF. Вероятно, это связано с тем, что ORB не требует, чтобы ключевая точка была абсолютным локальным максимумом, а была максимумом/минимумом n-непрерывного ряда.
-



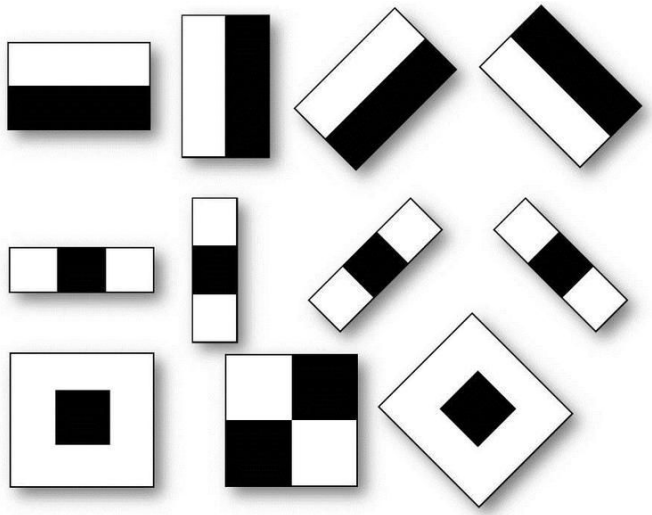






Выделение признаков

Признаки Хаара - Простой прямоугольный признак Хаара можно определить как разность суммы пикселей областей внутри прямоугольника, которая может находиться в любом положении и масштабе исходного изображения

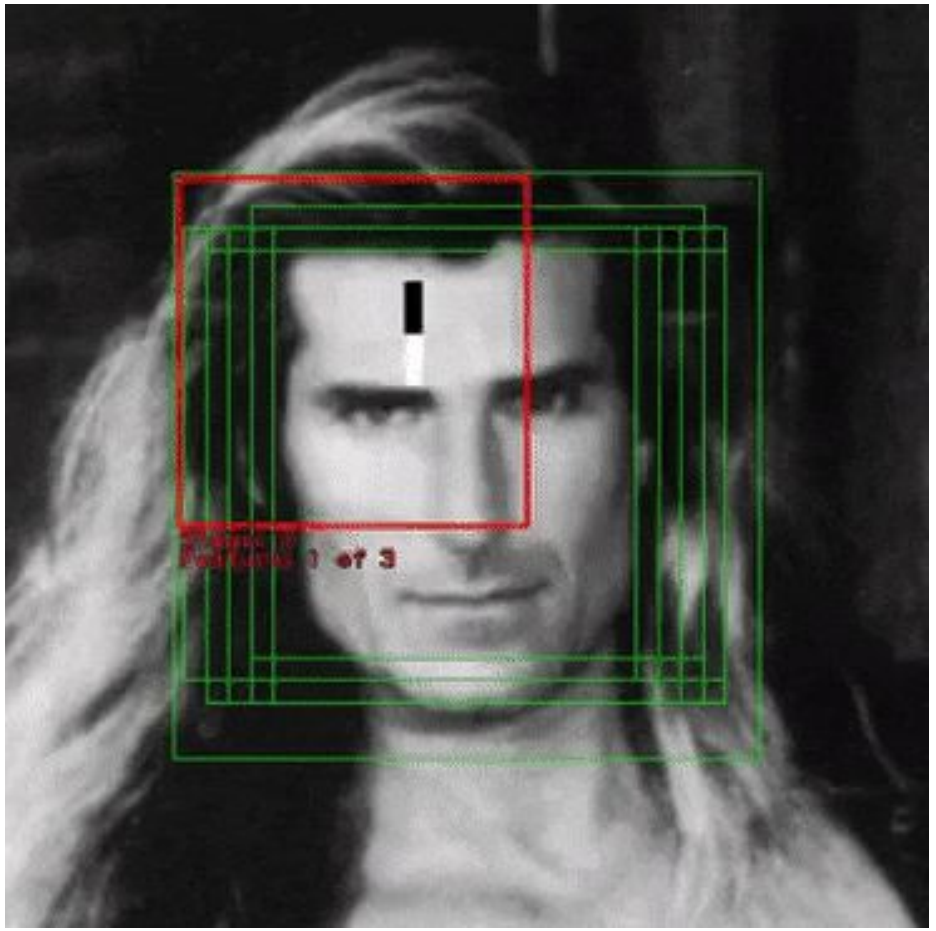


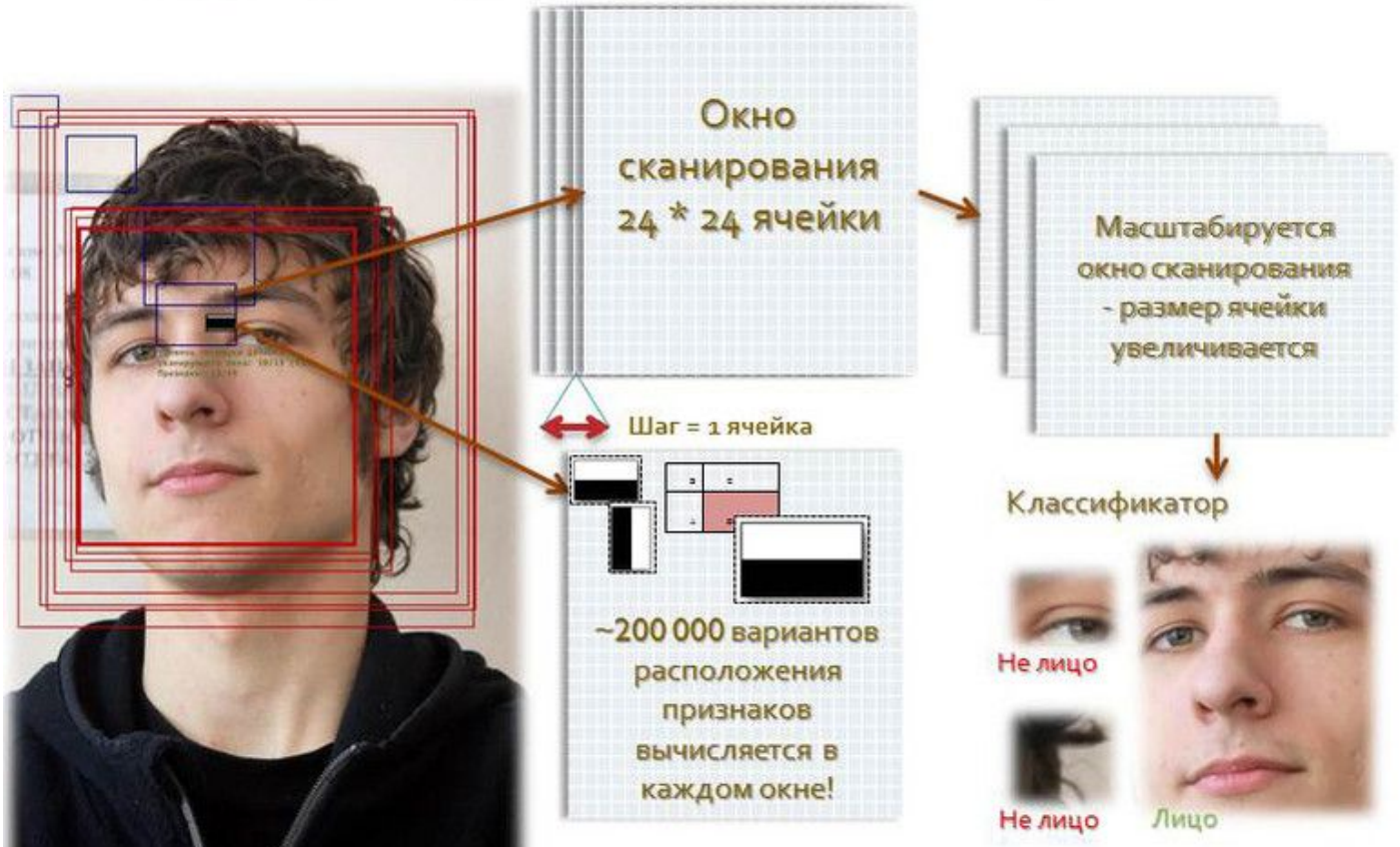
Такие свёртки подчёркивают структурную информацию объекта

Чем больше используется различных примитивов, тем точнее можно потом классифицировать объект.

Для центра лица человека будет всегда отрицательна следующая свёртка:

Глаза будут темнее, чем область между ними, так же как область рта будет темнее чем лоб.

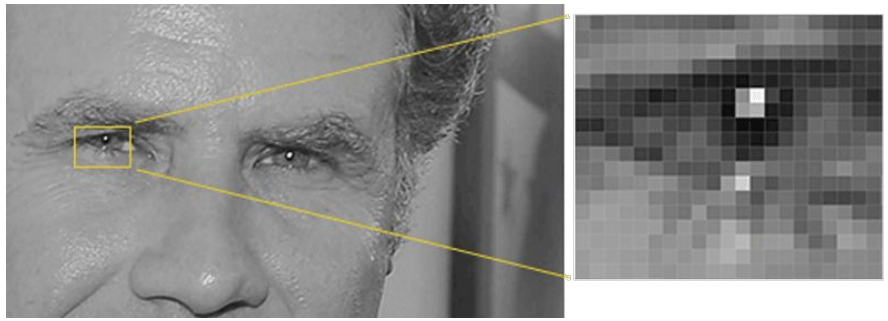


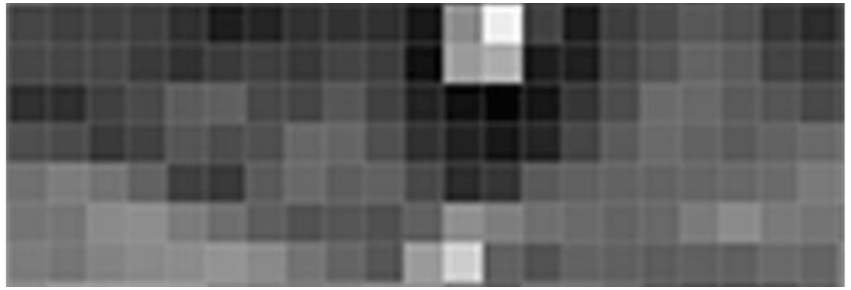

Окно сканирования 24 * 24 ячейки
Масштабируется окно сканирования - размер ячейки увеличивается
Шаг = 1 ячейка
~200 000 вариантов расположения признаков вычисляется в каждом окне!
Классификатор
Не лицо
Не лицо
Лицо

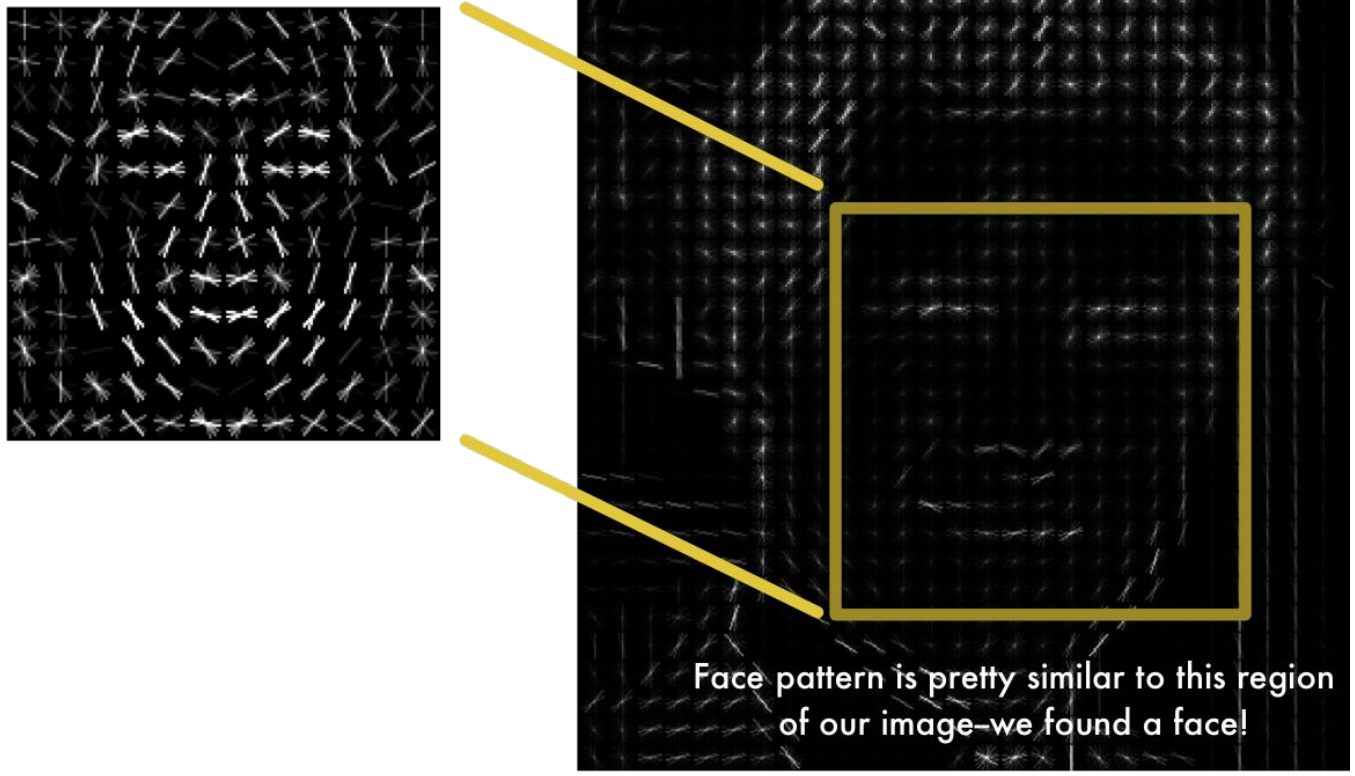
Выделение признаков

Признаки Хога - Histograms of Oriented Gradients (HOG)

Основная идея HOG заключается в вычислении гистограмм ориентированных градиентов в локальных областях изображения, и использовании этой информации для представления формы, текстуры и структуры объектов.







Спасибо за внимание!