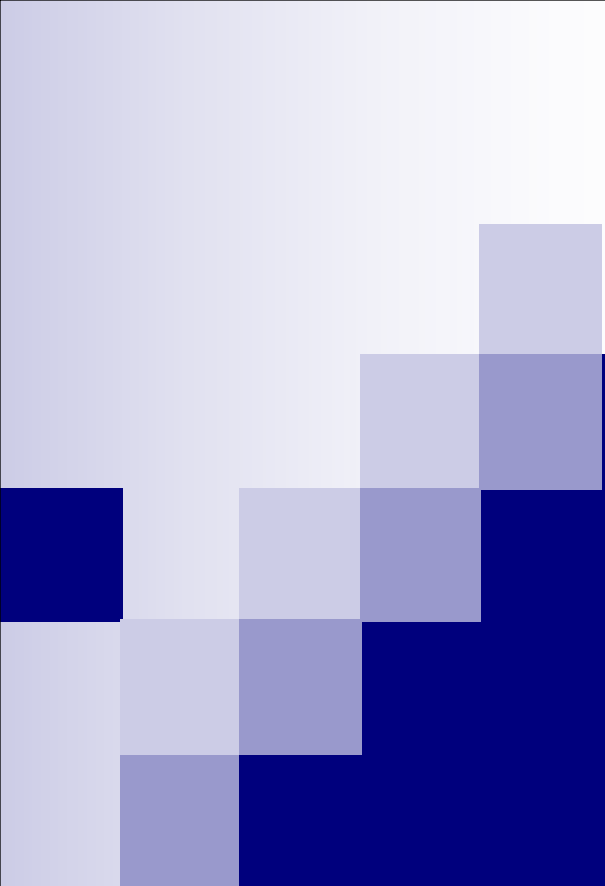




Распознавание образов



Обработка и анализ изображений

Зачем обрабатывать изображения?

- Для эффективного хранения и передачи
 - Компактное представление изображения в камере
 - Передача изображения с Марса на Землю
- Для подготовки к показу или печати
 - Приведение изображения к нужному размеру
 - Полутоновое представление
- Для улучшения и восстановления изображений
 - Удаление царапин на кадрах старых фильмов
 - Повышение видимости новообразований на медицинских снимках
- Для получения информации с изображений
 - Прочитать почтовый индекс на конверте
 - Измерить уровень влажности поверхности по аэроснимку

Цифровые изображения и цифровая обработка изображений

- Изображение – двумерный сигнал, который может восприниматься зрительной системой человека
- Цифровое изображение – представление изображений, дискретизированное по времени и в пространстве
- Цифровая обработка изображений – выполнение операций цифровой обработки сигналов для цифровых изображений

Цифровые изображения и цифровая обработка изображений

■ Изображение = функция

$$f(x, y, z, t, w) = v$$

- x, y, z – пространственные координаты (**где?**)
- t – время (**когда?**)
- w – длина регистрируемой волны, цвет (**что?**)
- v – значение функции в интервале $0 \dots v_{\max}$

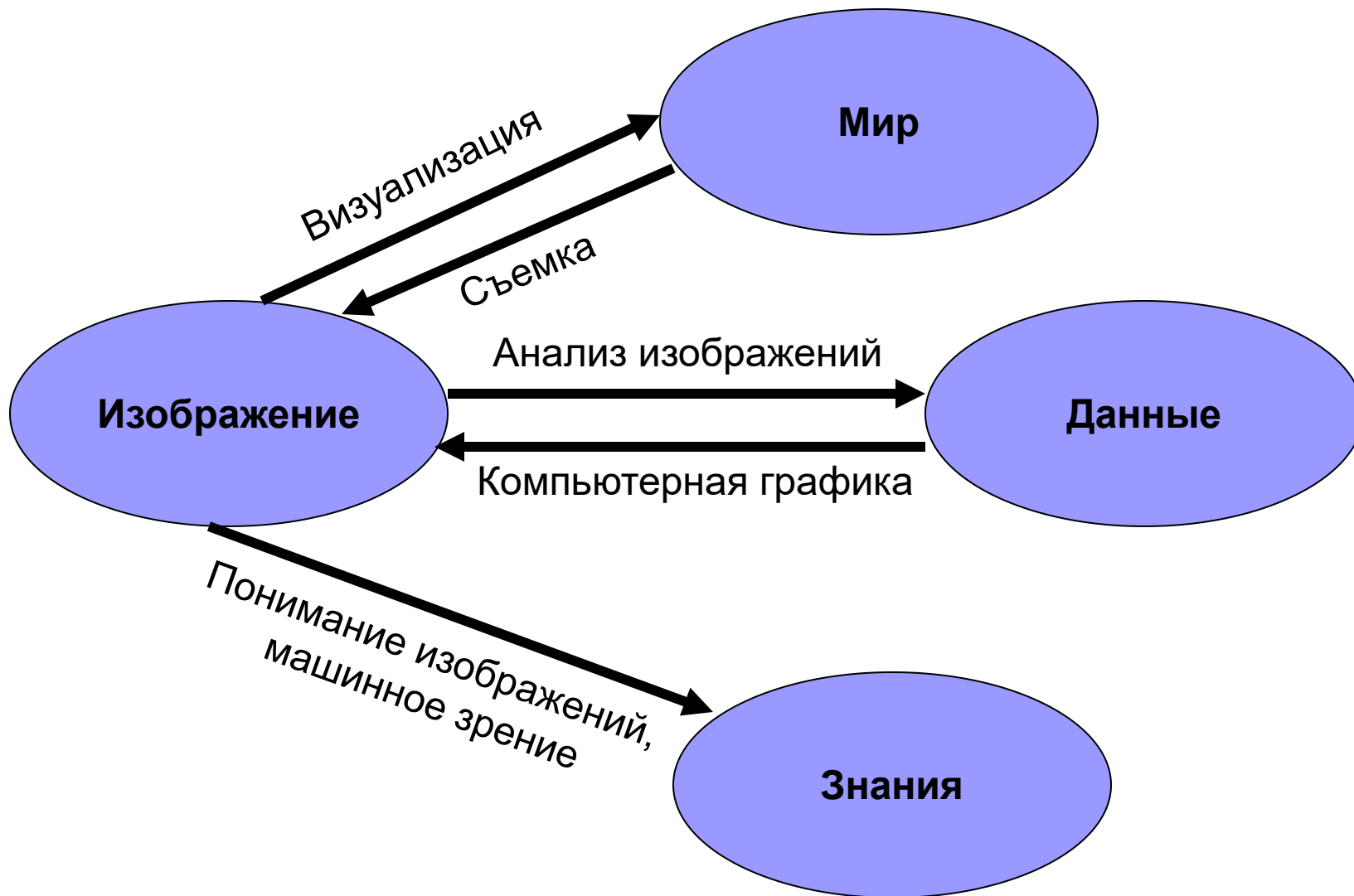
• Примеры

- $f(x, y)$ = черное/белое изображение
- $f(x, y, t)$ = видеоизображение
- $f(x, y, z)$ = томографическое изображение
- $f(x, y, w)$ = спутниковое изображение

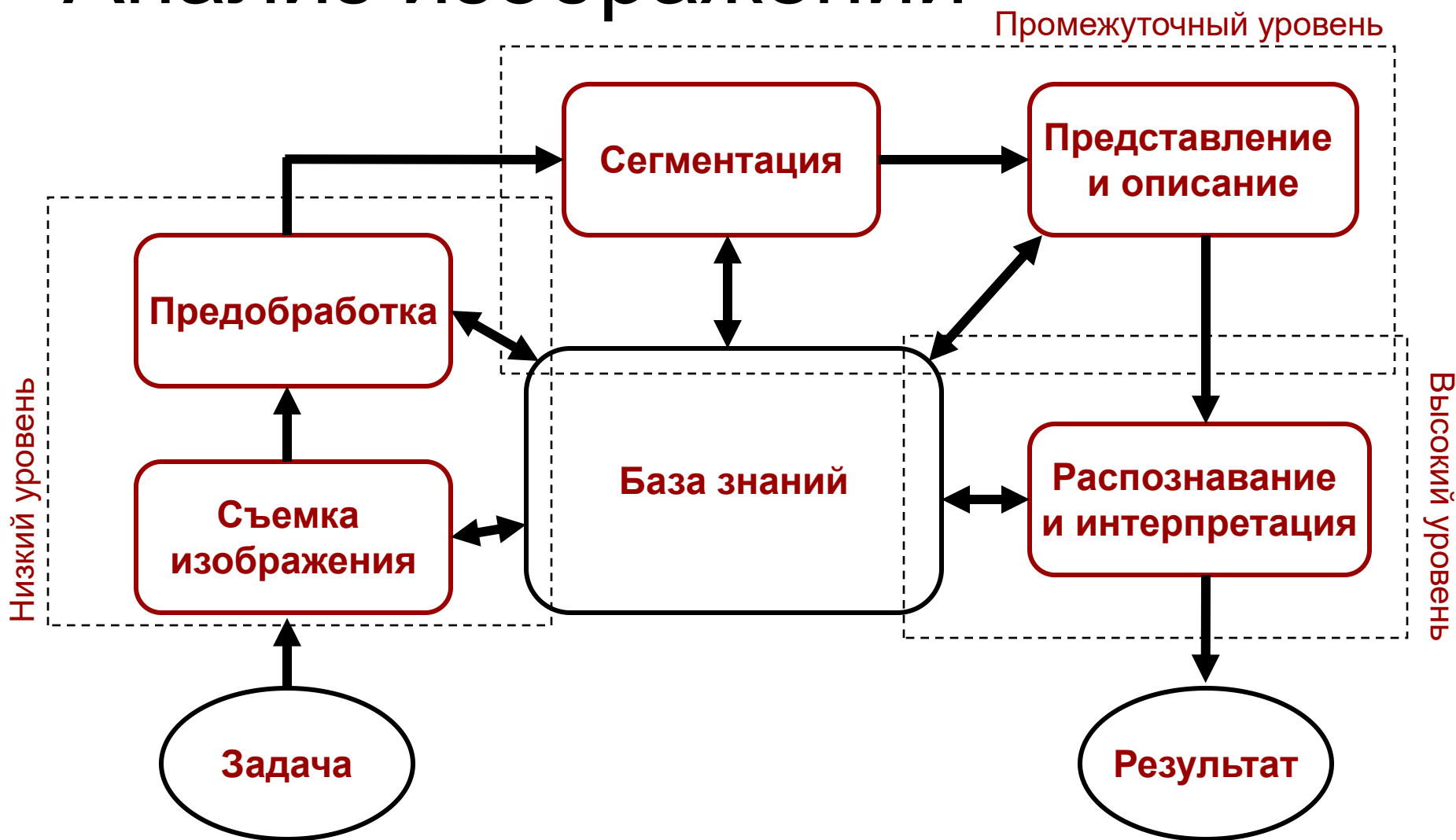
Обработка изображений

- Улучшение изображений (image enhancement)
 - Улучшение визуального качества изображений для восприятия человеком (сглаживание, контрастирование, эквализация гистограмм и т.п.)
- Восстановление изображений (image restoration/reconstruction)
 - Устранение смазов, размытия и т.п.
 - Воссоздание изображения – результата косвенных измерений (интерферометрия, радары, томография)
- Анализ изображений (image analysis)
 - Автоматическое извлечение информации из изображений (машинное зрение, медицинские приложения)
- Сжатие изображений (image compression)
 - Эффективное хранение и передача изображений с потерями или без в зависимости от приложения

Анализ изображений

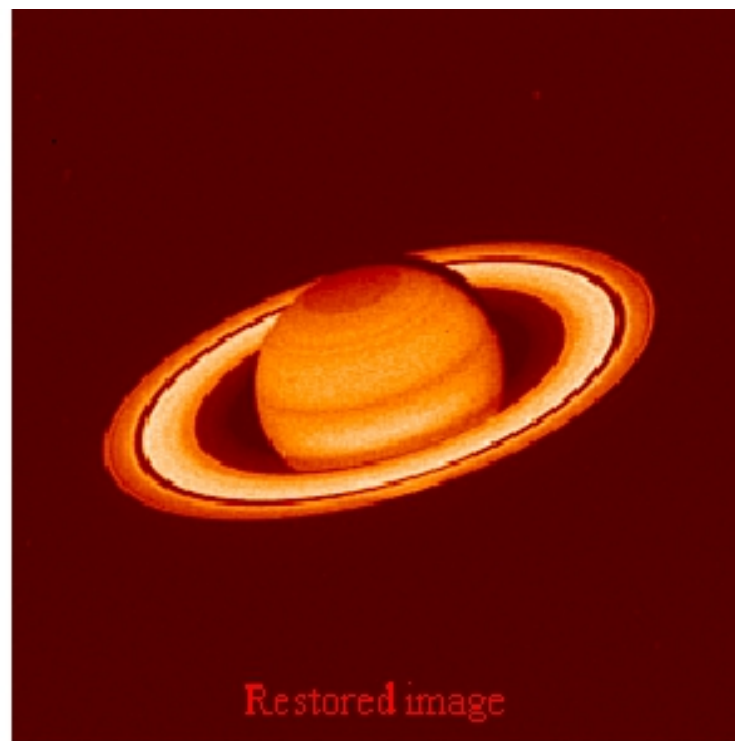


Анализ изображений



Астрономические изображения

- Восстановление изображения, полученного орбитальным телескопом Hubble



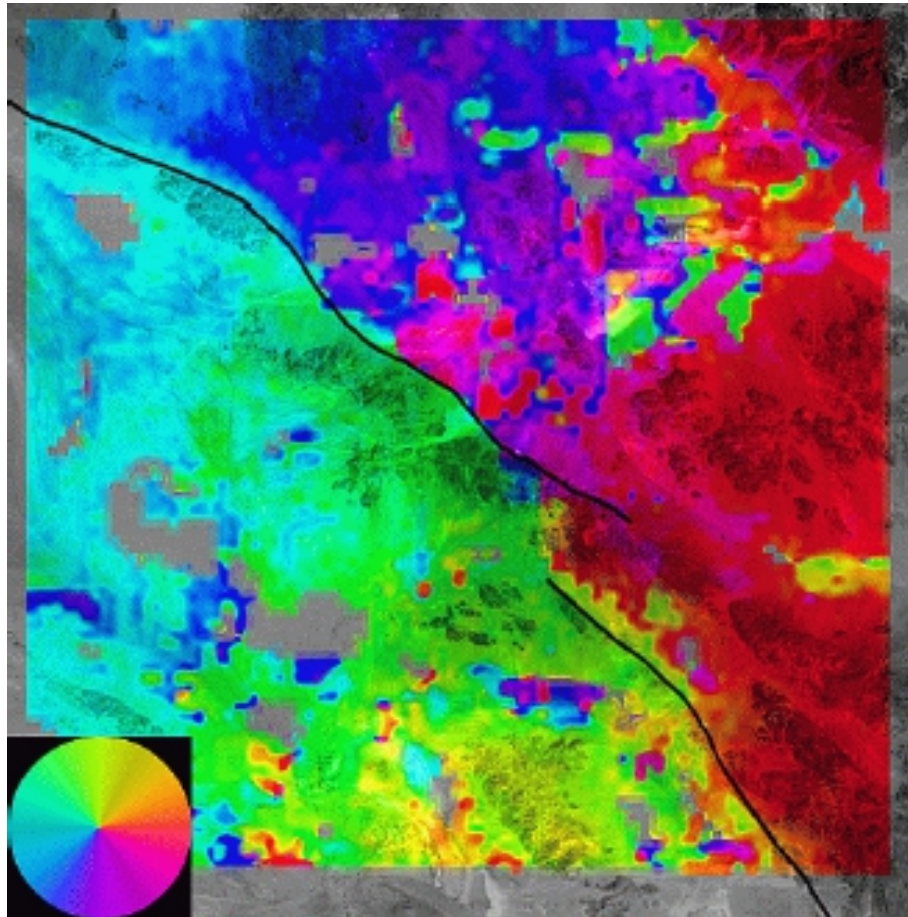
Анализ данных дистанционного зондирования

- Выделение территории поселения на аэроснимке



Анализ данных дистанционного зондирования

- Анализ последствий землетрясения по спутниковым изображениям



Реставрация изображений

- Реставрация фотографий



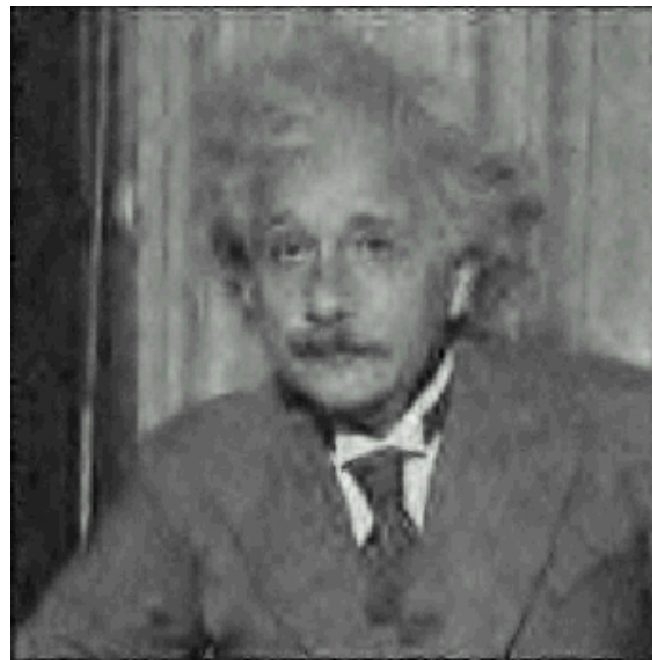
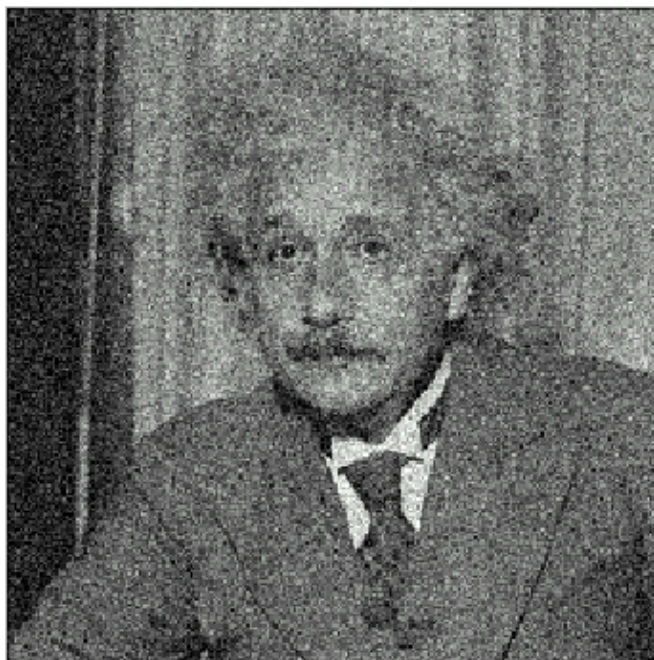
Реставрация изображений

■ Реставрация фильмов



Реставрация изображений

- Подавление шумов



Выделение лиц на изображении

- Выделение лиц на портретных фото



Выделение лиц на изображении

- Выделение лиц на групповых фото



Выделение лиц на изображении

- Отслеживание лиц



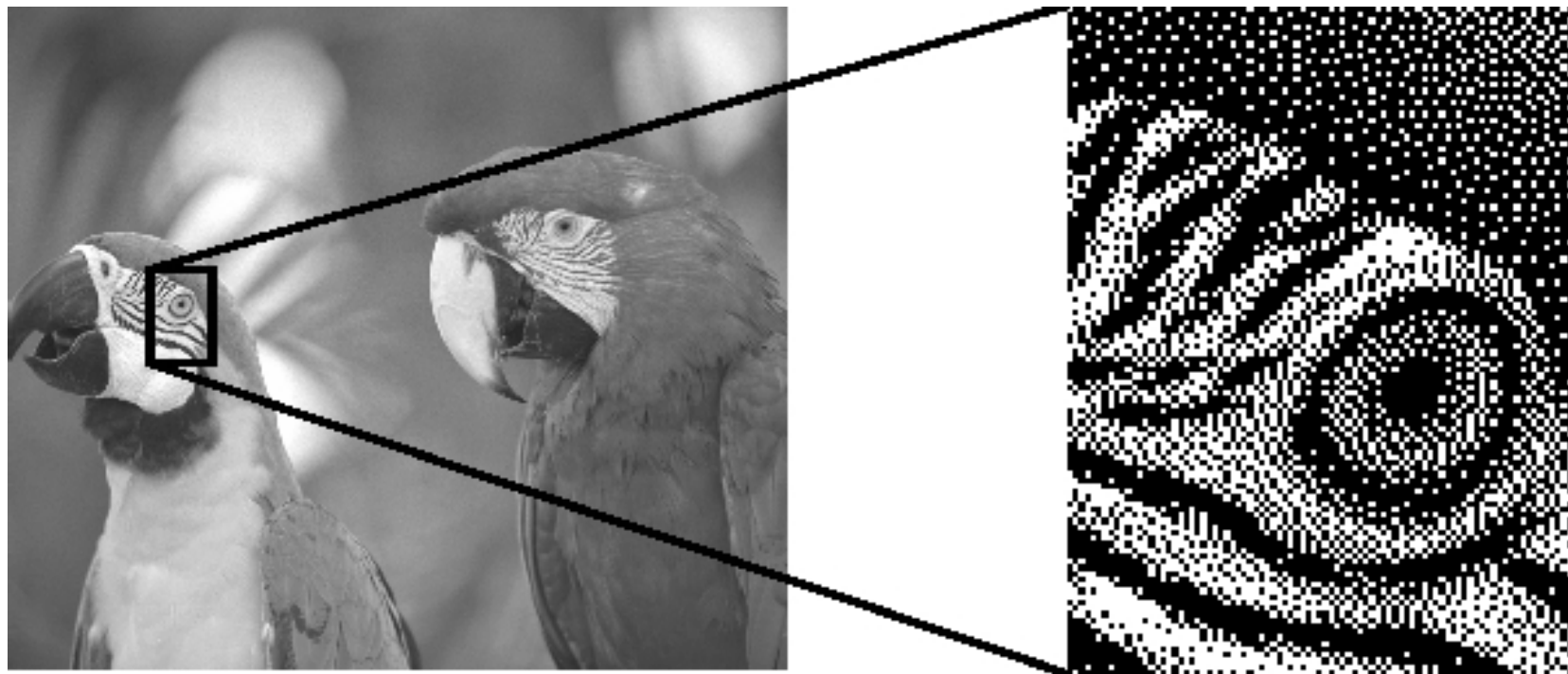
Выделение лиц на изображении

■ Поиск лиц в БД



Подготовка к показу или печати

- Полутонное представление

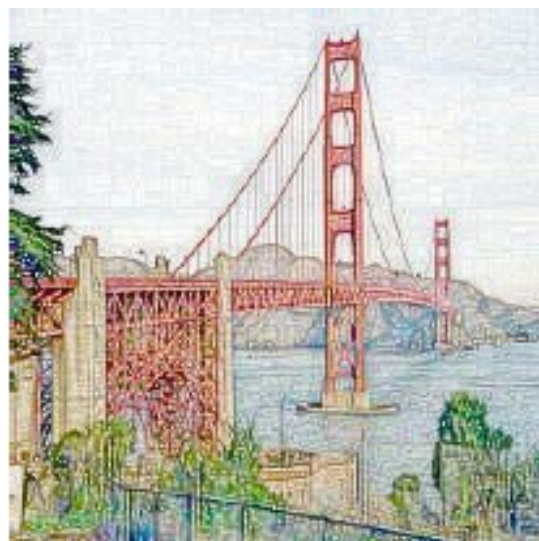


Подготовка к показу или печати

■ Спецэффекты



Фото



Имитация
карандашного
рисунка



Имитация
рисунка
маслом

Подготовка к показу или печати

■ Морфинг



Распознавание изображений

■ Распознавание рукописного текста

0 1 2 3 4 5 6 7 8 9

0 1 2 3 4

0 1 2 3 4 5 6 7 8 9

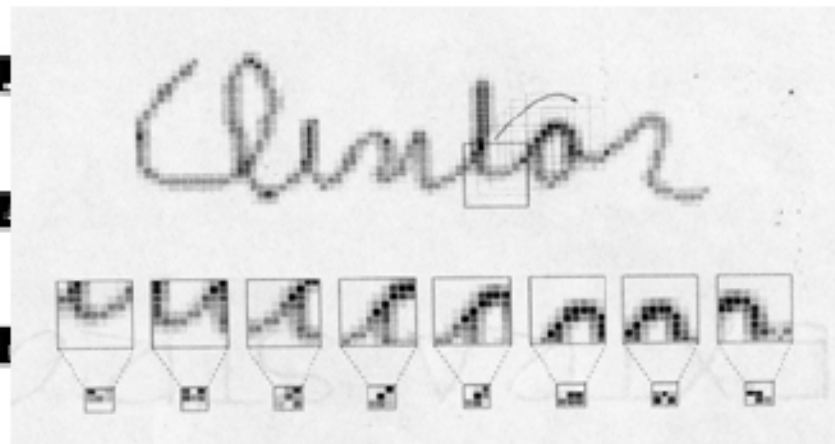
0 1 2 3 4

0 1 2 3 4 5 6 7 8 9

0 1 2 3 4

(a)

(b)



0 1 2 3 4 5 6 7 8 9

0 1 2 3 4 5 6 7 8 9

0 1 2 3 4 5 6 7 8 9

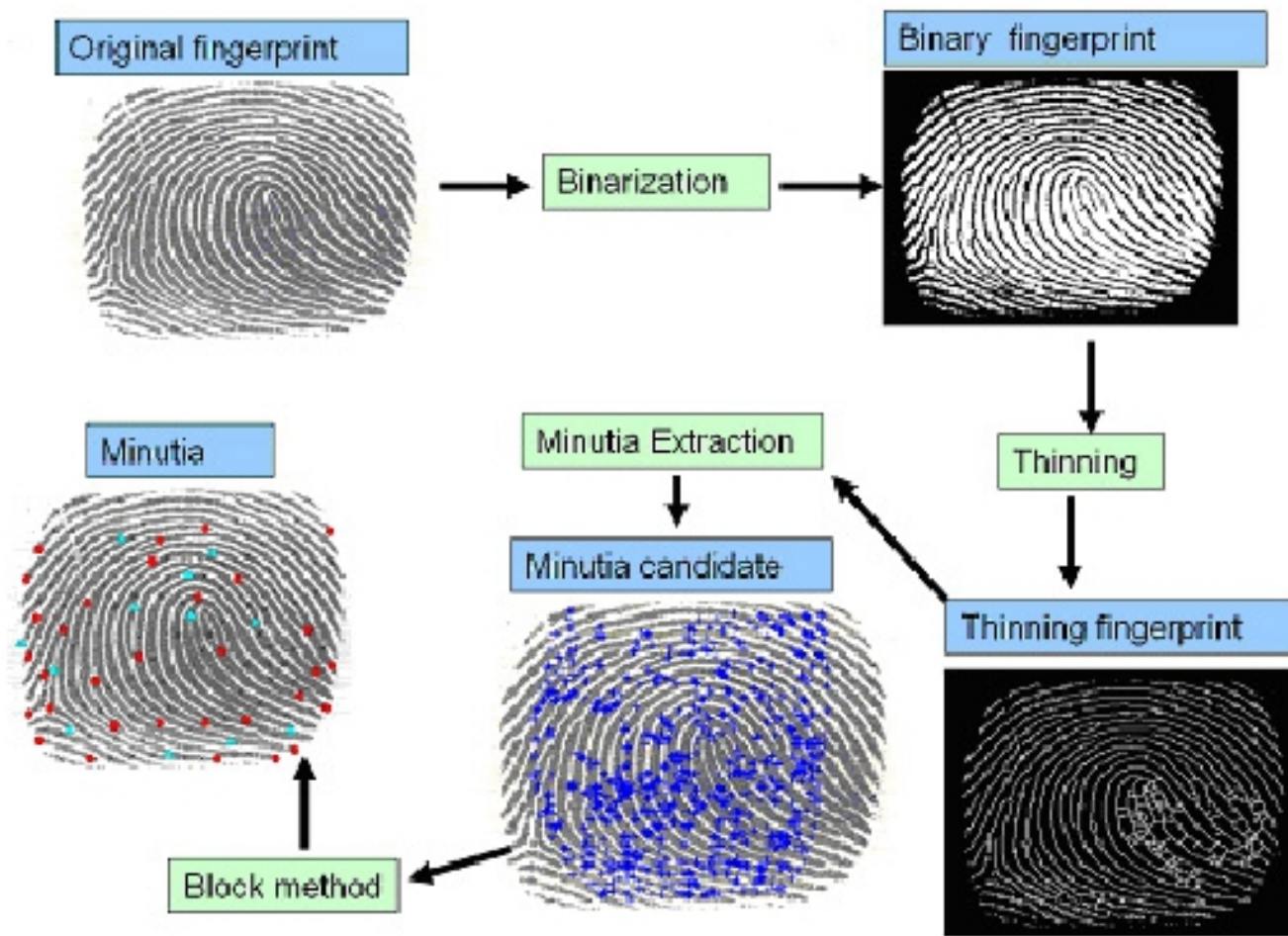
0 1 2 3 4 5 6 7 8 9

0 1 2 3 4 5 6 7 8 9

0 1 2 3 4 5 6 7 8 9

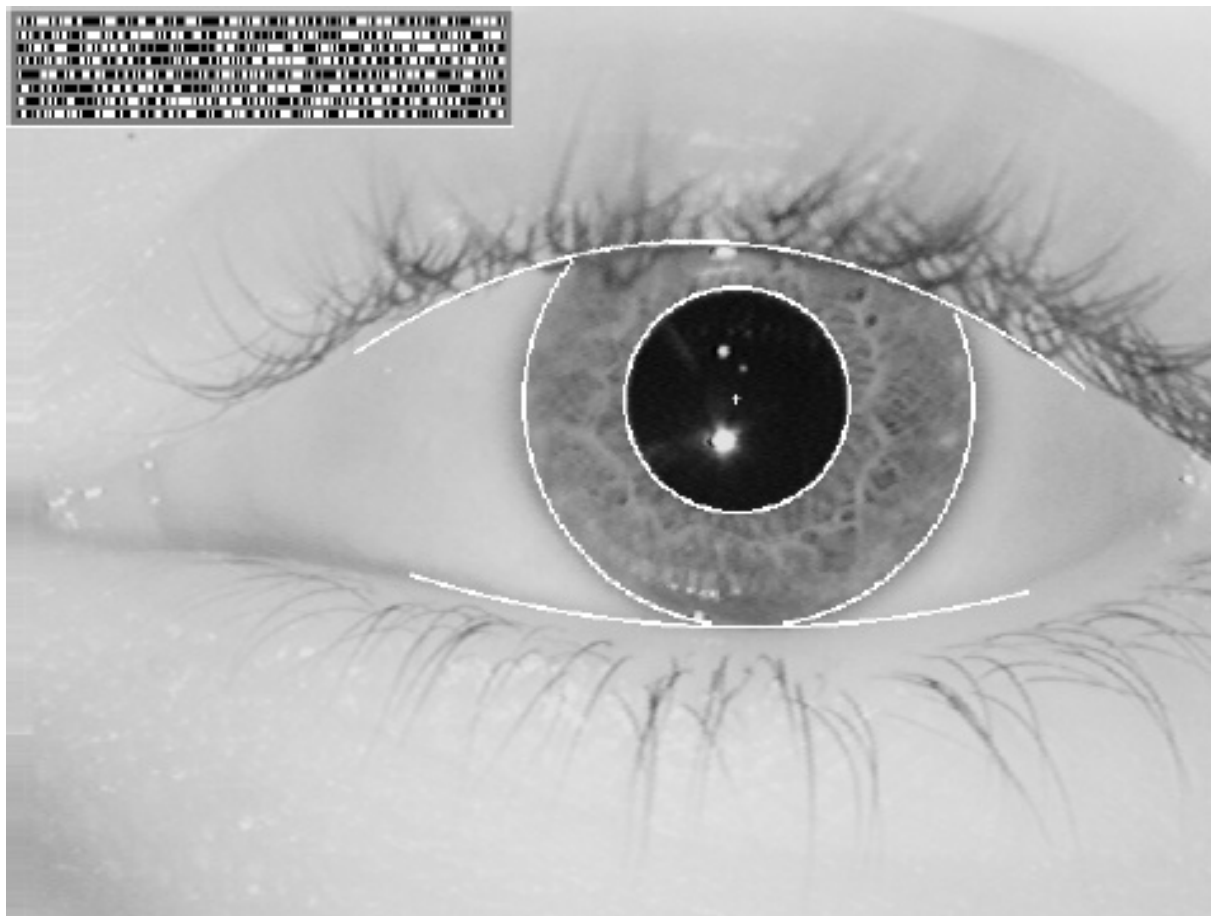
Распознавание изображений

- Биометрия: распознавание отпечатков пальцев



Распознавание изображений

- Биометрия: распознавание радужки



Другие приложения

- Системы визуальной связи
- Проверка и диагностика сложных систем
 - Тело человека
 - Промышленные системы
- Системы безопасности
- Управление мультимедиа-информацией
- Кино, игры
- ...и т.п., и т.д.



Различные способы преобразования изображений с помощью Python и PIL

❑ Пример №1

```
import random
```

```
from PIL import Image, ImageDraw #Подключим  
необходимые библиотеки
```

```
image = Image.open("image.jpg") #Открываем изображение
```

```
draw = ImageDraw.Draw(image) #Создаем инструмент для  
рисования
```

```
width = image.size[0] #Определяем ширину
```

```
height = image.size[1] #Определяем высоту
```

```
pix = image.load() #Выгружаем значения пикселей
```

Оттенки серого

Для получения этого преобразования проще всего «усреднить» каждый пиксел:

$$S = (r + g + b) // 3$$

Хотя в реальности используется более сложное преобразование

□ Пример №2

```
for i in range(width):  
    for j in range(height):  
        r = pix[i, j][0]  
        g = pix[i, j][1]  
        b = pix[i, j][2]  
        S = (r + g + b) // 3  
        draw.point((i, j), (S, S, S))
```

```
image.save("ans.jpg", "JPEG")
```

```
del draw#Сохраняем результат и удаляем кисть
```

Черно-белое изображение с порогом

Разбьем все пикселы на 2 группы: черные и белые по введенному пользователем порогу **factor**.

Для проверки принадлежности к определенной группе мы будем смотреть к чему ближе значение пиксела: к белому цвету или к чёрному.

```
S = (r + g + b)
if (S > (((255 + factor) // 2) * 3)):
    r, g, b = 255, 255, 255
else:
    r, g, b = 0, 0, 0
```

Черно-белое изображение с порогом

❑ Пример №3

```
factor = int(input('factor:'))  
for i in range(width):  
    for j in range(height):  
        r = pix[i, j][0]  
        g = pix[i, j][1]  
        b = pix[i, j][2]  
        S = (r + g + b)
```

Черно-белое изображение с порогом

□ Пример №3 (продолжение)

```
if (S > (((255 + factor) // 2) * 3)):
```

```
    r, g, b = 255, 255, 255
```

```
else:
```

```
    r, g, b = 0, 0, 0
```

```
draw.point((i, j), (r, g, b))
```


Цвет, присущий старым чёрно-белым фотографиям.



Вычислим среднее и используем коэффициент:

$$\text{mean} = (R + G + B) / 3$$

$$R = \text{mean} + 2 * \text{depth}$$

$$G = \text{mean} + \text{depth}$$

$$B = \text{mean}$$

□ Пример №4

```
depth = int(input('depth:'))  
for i in range(width):  
    for j in range(height):  
        r = pix[i, j][0]  
        g = pix[i, j][1]  
        b = pix[i, j][2]  
        S = (r + g + b) // 3  
        r = S + depth * 2  
        g = S + depth  
        b = S
```

□ Пример №4 (продолжение)

$r = \min(r, 255)$):# Есть опасность, что сумма
превысит предел 255

$g = \min(g, 255)$

$b = \min(b, 255)$

Инверсия цветов (негатив)

В нашем случае это очень просто, достаточно лишь каждое значение пиксела вычесть из 255:

$$255 - r, 255 - g, 255 - b$$

Некоторые детали на таком изображении заметней.

Инверсия цветов (негатив)

❑ Пример №5

```
for i in range(width):
```

```
    for j in range(height):
```

```
        r = pix[i, j][0]
```

```
        g = pix[i, j][1]
```

```
        b = pix[i, j][2]
```

```
        draw.point((i, j), (255 - r, 255 - g, 255 - b))
```

Добавление шумов

Шум очень часто применяют для трансформации исходного изображения. Мы будем всегда добавлять к пикселу какое-нибудь случайное значение. Чем больше разброс этих значений, тем больше шумов:

```
rand = random.randint(-factor, factor)
```

```
r = pix[i, j][0] + rand
```

```
g = pix[i, j][1] + rand
```

```
b = pix[i, j][2] + rand
```

□ Пример №6

```
factor = int(input('factor:'))  
for i in range(width):  
    for j in range(height):  
        rand = random.randint(-factor, factor)  
        r = pix[i, j][0] + rand  
        g = pix[i, j][1] + rand  
        b = pix[i, j][2] + rand
```

□ Пример №6 (продолжение)

`r = max(r, 0)`

`g = max(g, 0)`

`b = max(b, 0)`

`r = min(r, 255)`

`g = min(g, 255)`

`b = min(b, 255)`

`draw.point((i, j), (r, g, b))`

Управление яркостью изображения

Для регулирования яркости к каждому пикселу мы будем добавлять определенное значение. Если оно > 0 , то картинка становится ярче, иначе – темнее.

❑ Пример №7

```
factor = int(input('factor:'))  
for i in range(width):  
    for j in range(height):  
        r = pix[i, j][0] + factor  
        g = pix[i, j][1] + factor  
        b = pix[i, j][2] + factor
```

□ Пример №7 (продолжение)

`r = max(r, 0)`

`g = max(g, 0)`

`b = max(b, 0)`

`r = min(r, 255)`

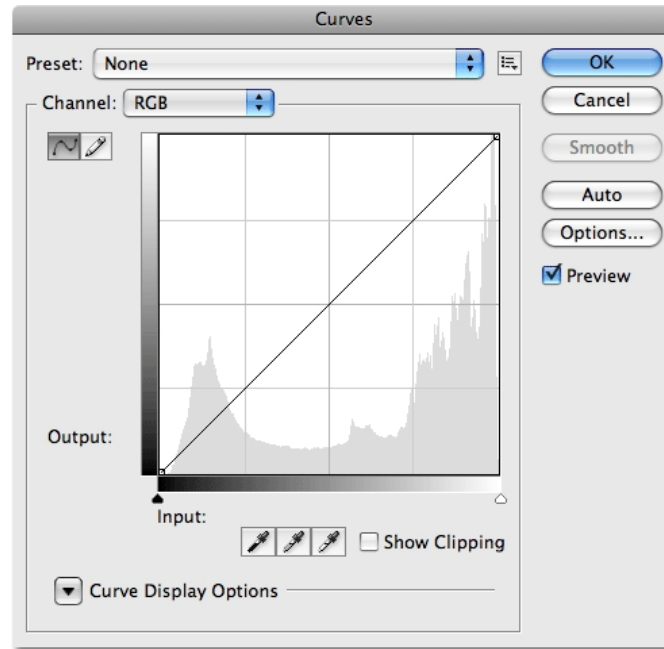
`g = min(g, 255)`

`b = min(b, 255)`

`draw.point((i, j), (r, g, b))`

Высветление изображения

В графических редакторах есть инструмент «Кривые» (Curves). Он позволяет задать функцию, меняющую яркость всего пикселя или отдельной компоненты в зависимости от исходной яркости. Изначально эта функция представляет собой прямую $y = x$.



Высветление изображения

В Python можно написать функцию, которая работает как инструмент Curves. Например, мы можем высветлить темные участки в изображении, не трогая светлые. Это очень частая операция: например, когда на снимке светлое небо и очень темное здание, потому что фотоаппарат подстроился под яркость неба.

«Высветлить» означает увеличить значения всех цветовых компонентов, умножив на какой-то коэффициент. Важно помнить, что эти значения не могут быть больше 255.

□ Пример №8

```
factor = int(input('factor:'))
```

```
for i in range(width):
```

```
    for j in range(height):
```

```
        draw.point((i, j), (curve(pix[i, j], factor)))
```

□ Пример №8 (функция)

```
def curve(pixel, factor):
```

```
    r, g, b = pixel
```

```
    brightness = r + g + b
```

```
    if 0 < brightness < factor:
```

```
        k = factor / brightness
```

```
        return min(255, int(r * k ** 2)), \
```

```
                min(255, int(g * k ** 2)), \
```

```
                min(255, int(b * k ** 2))
```

```
    else:
```

```
        return r, g, b
```

Кадрирование изображения

В PIL есть большое число встроенных инструментов для изменения изображений.

Например, мы можем с помощью функции **crop** вырезать прямоугольный кусочек из изображения. В функцию передаются координаты верхнего левого и правого нижнего угла вырезаемого прямоугольника одним кортежем. Обратите внимание: все подобные функции не изменяют исходное изображение, а возвращают его измененную копию.

❑ Пример №9

```
im = Image.open("image.jpg")  
im2 = im.crop((200, 200, 500, 500))  
im2.save('res.jpg')
```

Изменение размера изображения

Мы можем изменить размер изображения с помощью функции `resize`, в которую кортежем передается новый размер изображения.

□ Пример №10

```
im = Image.open("image.jpg")  
im2 = im.resize((200, 200))  
im2.save('res.jpg')
```

Сокращение цветов в палитре изображения

Функция **quantize** используется для сокращения цветов в палитре изображения и используется для создания миниатюр для предпросмотра. Принимает на вход число меньшее 256 — количество цветов.

*Обратите внимание: эта функция преобразовывает изображение в формат **bmp**.*

Пример №11

```
im = Image.open("image.jpg")  
im2 = im.quantize(16)  
im2.save('res.bmp', 'BMP')
```

Вращение и отражение изображения

PIL содержит готовые реализации вращения и отражения изображения. Повороты и отражения можно выполнить с помощью функции `transpose`, в которую передается тип преобразования. Это может быть отражение слева направо, или сверху вниз, или повороты на 90, 180 или 270 градусов.

Вращение и отражение изображения

□ Пример №12

```
im = Image.open("image.jpg")
im2 = im.transpose(Image.FLIP_LEFT_RIGHT).
    transpose(Image.ROTATE_90)
# Image.FLIP_LEFT_RIGHT, - отражение слева на право
# Image.FLIP_TOP_BOTTOM, - отражение сверху вниз
# Image.ROTATE_90, - поворот на 90 градусов против
    часовой стрелки
# Image.ROTATE_180,
# Image.ROTATE_270
im2.save('res.jpg')
```

Избранные функции модуля ImageOps

В модуле **ImageOps** библиотеки **PIL** есть встроенные реализации превращения изображения в негатив (функция **invert**) и в черно-белое изображение (функция **grayscale**).

Можно отразить изображение слева на право функцией **mirror**).

Управляя параметром **cutoff** функции **autocontrast**, можно менять контрастность изображения.

Функция **colorize** поможет раскрасить черно-белое изображение в выбранную пару цветов.

□ Пример №13

```
from PIL import ImageOps
```

```
im = Image.open("image.jpg")
```

```
im2 = ImageOps.invert(im)
```

```
im2.save('res.jpg')
```

❑ Пример №14

```
from PIL import ImageOps
```

```
im = Image.open("image.jpg")
```

```
im2 = ImageOps.grayscale(im)
```

```
im2.save('res.jpg')
```


□ Пример №15

```
from PIL import ImageOps
```

```
im = Image.open("image.jpg")
```

```
im2 = ImageOps.autocontrast(im, 10)
```

```
im2.save('res.jpg')
```

Замена цветов в черно-белом изображении

□ Пример №16

```
from PIL import ImageOps
```

```
im = Image.open("image.jpg")
```

```
im2 = ImageOps.colorize(im, 'black', 'white', mid = None,  
                        blackpoint = 0, whitepoint = 255,  
                        midpoint = 127)
```

```
im2.save('res.jpg')
```

Pillow позволяет использовать множество различных фильтров для обработки изображения. Они являются частью модуля **ImageFilter**. Давайте рассмотрим несколько примеров использования метода **filter()**

❑ Пример №17

```
from PIL import ImageFilter
```

```
im = Image.open("image.jpg")
```

```
im2 = im.filter(ImageFilter.BLUR)
```

```
#Варианты: SMOOTH, SMOOTH_MORE
```

```
im2.save('res.jpg')
```

□ Пример №18

```
from PIL import ImageFilter
```

```
im = Image.open("image.jpg")
```

```
im2 = im.filter(ImageFilter.GaussianBlur(radius=5))
```

```
im2.save('res.jpg')
```

□ Пример №19

```
from PIL import ImageFilter
```

```
im = Image.open("image.jpg")
```

```
im2 = im.filter(ImageFilter.CONTOUR)
```

```
# FIND_EDGES
```

```
im2.save('res.jpg')
```

❑ *Пример №20*

```
from PIL import ImageFilter
```

```
im = Image.open("image.jpg")
```

```
im2 = im.filter(ImageFilter.EMBOSS)
```

```
im2.save('res.jpg')
```

❑ Пример №21

```
from PIL import ImageFilter
```

```
im = Image.open("image.jpg")
```

```
im2 = im.filter(ImageFilter.SHARPEN)
```

```
# В а р и а н т ы :  D E T A I L ,  E D G E _ E N H A N C E ,  
    EDGE_ENHANCE_MORE
```

```
im2.save('res.jpg')
```


Улучшение изображения через **ImageFilter** в **Pillow**:

▣ *Пример №22*

```
from PIL import ImageEnhance
```

```
im = Image.open("image.jpg")
```

```
enh = ImageEnhance.Contrast(im)
```

```
im2 = enh.enhance(1.9)
```

```
im2.save('res.jpg')
```

□ Пример №23

```
def transparency(filename1, filename2):  
    im1 = Image.open(filename1)  
    koef1 = 0.5  
    koef2 = 1 - koef1  
    pixels1 = im1.load()  
    im2 = Image.open(filename2)  
    pixels2 = im2.load()  
    x, y = im1.size
```

Наложение изображений с прозрачностью

□ Пример №23 (продолжение)

```
for i in range(x):  
    for j in range(y):  
        r1, g1, b1 = pixels1[i, j]  
        r2, g2, b2 = pixels2[i, j]  
        r = int(koef1 * r1 + koef2 * r2)  
        g = int(koef1 * g1 + koef2 * g2)  
        b = int(koef1 * b1 + koef2 * b2)  
        pixels1[i, j] = r, g, b  
im1.save('res.jpg')  
  
transparency("image1.jpg", "image2.jpg")
```