

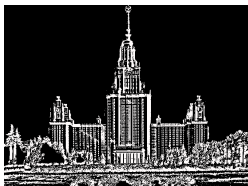
# Занятие 4: Основы обработки изображений

Практикум на ЭВМ 2021/2022

Елистратов Семен Юрьевич

МГУ имени М. В. Ломоносова, факультет ВМК, кафедра ММП

# Типы изображений



Бинарное



Серое



Цветное

## Пример: устранение шума

noisy



non-local means



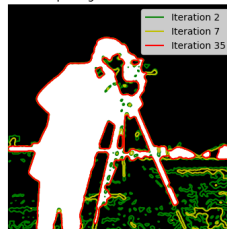
[http://scikit-image.org/docs/dev/auto\\_examples/filters/...](http://scikit-image.org/docs/dev/auto_examples/filters/...)

# Пример: выделение краёв

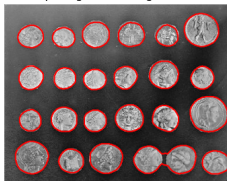
Morphological ACWE segmentation



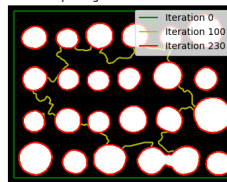
Morphological ACWE evolution



Morphological GAC segmentation



Morphological GAC evolution



[http://scikit-image.org/docs/dev/auto\\_examples/segmentation/...](http://scikit-image.org/docs/dev/auto_examples/segmentation/...)

## Пример: изменение баланса цвета



<http://studioeszkozok.hu/uploads/images/...>

## Зачем нужна обработка изображений?

1. Улучшение изображения для восприятия человеком (изображение должно стать «лучше» с субъективной точки зрения человека)
2. **Улучшение изображения для восприятия компьютером**  
(улучшение качества работы алгоритмов)
3. Технические нужды (например, уменьшение размера изображений для пересылки по почте)
4. Построение спецэффектов

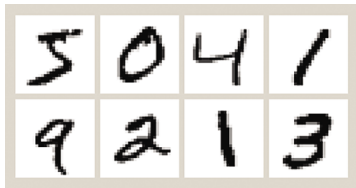
Обо всём этом подробнее в курсах «Обработка и распознавание изображений» и «Компьютерная графика»

# Улучшение качества работы алгоритма на изображениях

- ▶ Предобработка входных изображений
  - ▶ Удаление шума, преобразование цветов
- ▶ Постобработка изображений
  - ▶ Удаление шумовых объектов, отверстий
- ▶ Выделение дополнительных признаков
  - ▶ Выделение важных объектов, областей
- ▶ Аугментация объектов
  - ▶ Преобразование объектов в ходе обучения/применения модели

## Пример: классификация цифр

Первое практическое задание: классификация датасета MNIST



Класс изображения не меняется при:

- ▶ сдвигах на 1-10 пикселей
- ▶ поворотах на 10-15 градусов в каждую из сторон
- ▶ размытии, удалении шумов

Добавив преобразованные объекты в исходную выборку, можно существенно повысить качество



# Аугментация

Часто используемые методы аугментации

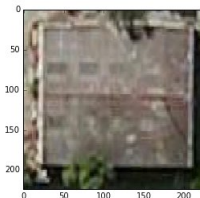
- ▶ отражение
- ▶ сдвиг
- ▶ поворот
- ▶ масштабирование (увеличение)
- ▶ добавление шума, размытие
- ▶ изменение яркости, контрастности, цветовой гаммы
- ▶ удаление части изображения

[Хорошая статья по аугментации изображений](#)

## Пример: классификация типов крыш

Задача: определение типа крыши, один из четырёх классов:

1. North-South orientation
2. East-West orientation
3. Flat roof
4. Other



1. При повороте на  $90^\circ$  объект 1 и 2 класса меняет класс.
2. При повороте на  $180^\circ$  объект 1 и 2 класса не меняет класс.
3. При повороте на  $90^\circ$  объект 3 класса не меняет класс.

Аугментация объектов улучшила точность с 80% до 84%

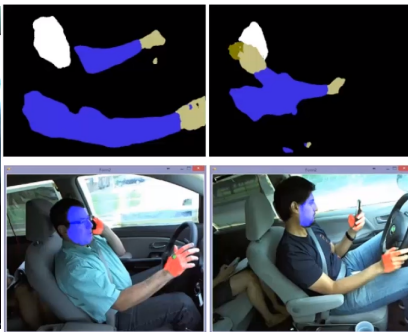
## Пример: детектирование действий водителя

Детектировать запрещённые действия водителей по видео:

Исходное изображение:



Выделение областей:



Алгоритм можно настраивать отдельно на выделенные области

# Библиотека scipy

- ▶ Scipy — библиотека для научных вычислений  
<https://scipy.org/>
- ▶ В том числе, есть несколько модулей для работы с изображениями (misc, ndimage)
- ▶ Только самые базовые алгоритмы

# Библиотека scikit-image

- ▶ Scikit-image — библиотека для работы с изображениями  
<http://scikit-image.org/>
- ▶ Большое количество реализованных алгоритмов для работы с изображениями
- ▶ Много выложенных примеров использования библиотеки, но плохая документация

# Библиотека OpenCV

- ▶ OpenCV — продвинутая библиотека для компьютерного зрения, есть хороший интерфейс для Python 2.7  
<https://opencv-python-tutroals.readthedocs.io/en/latest/index.html>
- ▶ Огромное количество реализованных алгоритмов для работы с изображениями, видео, 3D моделями
- ▶ Хорошая документация

# Представление изображения в памяти (scipy)

```
>>> import matplotlib.pyplot as plt
>>> %matplotlib inline
>>> import scipy.misc as misc
...
>>> img = misc.imread('msu.png') # загрузка изображения
>>> print(type(img)) # изображение - трёхмерный numpy array
<class numpy.ndarray>
>>> print(img.shape)
(595, 800, 3)
>>> plt.imshow(img)
```



# Представление изображения в памяти (skimage)

То же самое, но с scikit-image:

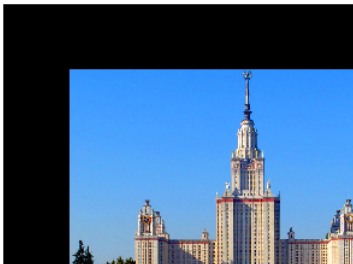
```
>>> import skimage.io as io
...
>>> img = io.imread('msu.png') # загрузка изображения
>>> print(type(img)) # изображение - трёхмерный numpy array
<class numpy.ndarray>
>>> print(img.shape)
(595, 800, 3)
>>> plt.imshow(img)
```





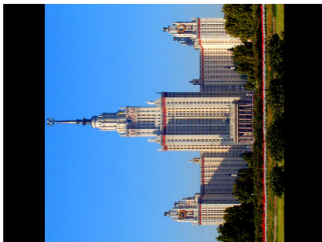
# Сдвиги изображений

```
>>> import scipy.ndimage as ndimage  
...  
>>> new_img = ndimage.shift(img, [150, 150, 0])  
>>> plt.imshow(new_img)
```



## Повороты изображений (scipy)

```
>>> new_img = misc.imrotate(img, 90)  
>>> plt.imshow(new_img)
```



Изображение обрезается, нельзя сохранить исходные пропорции, нельзя обрезать границы.

## Повороты изображений (skimage)

```
>>> import skimage.transform as transform  
>>> new_img = transform.rotate(img, 90, resize=True)  
>>> plt.imshow(new_img)
```



Много параметров:

```
transform.rotate(image, angle, resize=False, center=None,  
order=1, mode='constant', cval=0, clip=True,  
reserve_range=False)
```

## Оттенки серого

```
>>> import skimage.color as color
>>> grey_img = color.rgb2grey(img)
>>> plt.imshow(grey_img, cmap=plt.cm.Greys)
```



## Оттенки серого

```
>>> import skimage.color as color
>>> grey_img = color.rgb2grey(img)
>>> plt.imshow(grey_img, cmap=plt.cm.Greys)
```



Формула для преобразования:

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

# Бинаризация изображений

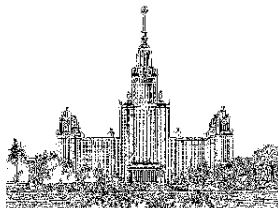
Бинаризация — перевод изображения в черно-белое

Бинаризация бывает:

- ▶ Глобальная (один порог для всех пикселей)
- ▶ Локальная (порог зависит от положения пикселя)
- ▶ Адаптивная (порог зависит от положения и от яркости пикселя)

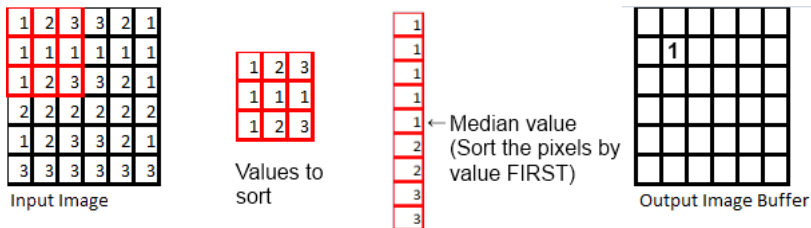
## Бинаризация в opencv

```
>>> import cv2 as cv
>>> th1, glob_bin_img = cv.threshold(
...     grey_img, 0.5, 1, cv.THRESH_BINARY
... )
>>> adaptive_bin_img = cv.adaptiveThreshold(
...     img_tmp, 255, cv.ADAPTIVE_THRESH_MEAN_C,
...     cv.THRESH_BINARY, 5, 5
... )
```



# Медианный фильтр

Медианный фильтр — возвращает медианное значение по окну



Является нелинейным фильтром (нельзя выразить с помощью линейных операций от входа).



## Удаление шумов с помощью медианного фильтра

```
>>> noise_coords_i = np.random.randint(0, img.shape[0], 50000)
>>> noise_coords_j = np.random.randint(0, img.shape[1], 50000)
>>> img_copy = np.copy(grey_img)
>>> img_copy[noise_coords_i, noise_coords_j] = 0
...
>>> blur_img = filters.median(img_copy)
```



## Гауссовский фильтр (gaussian blur)

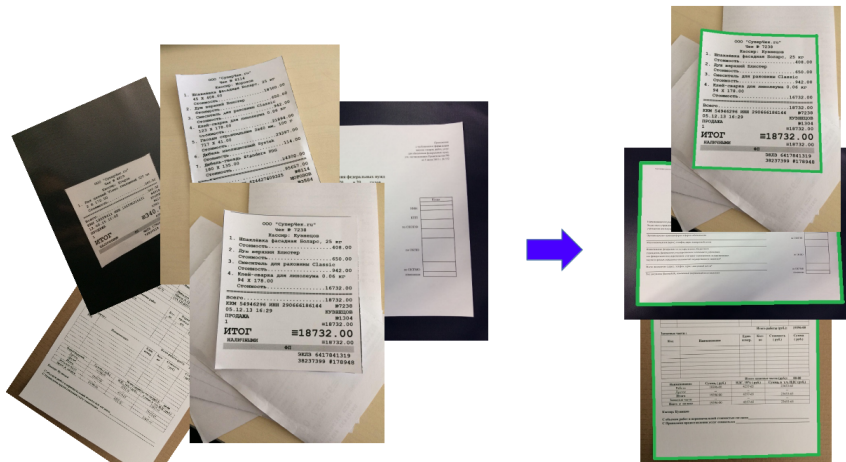
Можно применять гауссовский фильтр, чтобы сгладить/размыть изображение (является линейным):

$$Y[i, j] = \sum_{l=-k}^k \sum_{m=-k}^k X[i+l, j+m] \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{\sqrt{l^2 + m^2}}{2\sigma^2}\right)$$

```
>>> blur_image = filters.gaussian_filter(img, sigma=5)
```



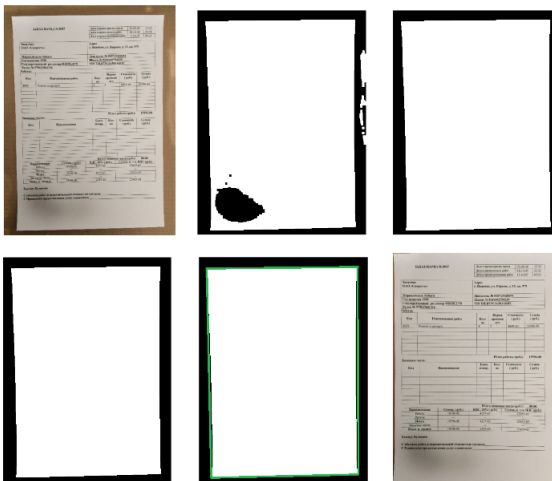
# Пример: бейзлайн для выделения границ документа



## Пример: бейзлайн для выделения границ документа

1. Сглаживание
2. Бинаризация
3. Удаление небольших черных областей
4. Удаление небольших белых областей
5. Аппроксимация четырехугольником
6. Восстановления порядка вершин
7. Вырезание документа

# Пример: бейзлайн для выделения границ документа



# Заключение

- ▶ Используя методы обработки изображений, можно повысить качество работы алгоритма
- ▶ Используя классические подходы, можно получить бейзлайн решения даже для сложных задач
- ▶ В Python есть несколько библиотек с уже реализованными алгоритмами обработки
- ▶ У многих библиотек есть свои встроенные средства обработки