Лабораторная работа №6

Задание №1. Как зовут жирафа, который занимается линейной алгеброй? Зильвестр Владимирович Джирафский!

Привет! Время выбрать весёлую картинку, чтобы мощно сжать её... Выбор широкий, но мы то знаем, как получить зачёт автоматом ;)

Рис. 1: Красавчик с двумя банановыми утками, которого мы сжимаем.

И сегодня именно он станет героем нашей лабораторной работы. Чтобы вся магия сжатия работала, нам придётся сделать всё немного грустным с нотками вьетнамских флешбеков — то есть чёрно-белым. Но не беда, мы всё равно сделаем это красиво, с помощью Python и модуля `Pillow`.

```python
from PIL import Image

img = Image.open('banana ducks.jpg')  # открываем изображение
img = img.convert('L')  # преобразуем изображение в черно-белое
img.save('banana ducks_bw.jpg')  # сохраняем изображение

```

И вот мы получаем вот такое невесёлое изображение:

Рис. 2: В этом красавчике с двумя банановыми утками 952560 чисел.

Ну да, выглядит грустненько, но мы же знаем, что это всего лишь промежуточный этап.

Теперь нам нужно представить это изображение в виде матрицы и, конечно же, выводить страшный список из 952560 чисел мы абсолютно точно не будем, поэтому выведем матрицу так, как её представляет `numpy`:

```python
from PIL import Image
import numpy as np

img = Image.open('banana ducks_bw.jpg')  # открываем изображение
img_array = np.array(img)  # преобразуем изображение в массив numpy
img_array = img_array / 255  # нормализуем значения пикселей до диапазона 0-1
print(img_array.shape)  # выводим размерность массива
print(img_array)  # выводим массив
```

```
(882, 1080)
[[255 255 255 ... 228 229 230]
 [255 255 255 ... 229 229 230]
 [255 255 255 ... 229 229 230]
 ...
 [ 91  91  90 ...  38  37  36]
 [ 92  97 100 ...  46  49  48]
 [ 92  97 100 ...  46  49  48]]
```

Действительно, в левом верхнем углу белое пятное, справа сверху немного светло, левый нижний край чуть тёмный, а справа снизу совсем темно. Всё верно, всё работает. Теперь мы можем сжать изображение, применив к нему сингулярное разложение. Для этого нам нужно найти это разложение для его матрицы, а потом взять только первые k компонент. И чтобы это сделать, мы воспользуемся функцией `numpy.linalg.svd`:

```python
from PIL import Image
import numpy as np


def svd_compress(img_array, k):
    U, E, V = np.linalg.svd(img_array)  # применяем SVD к массиву numpy
    U_k = U[:, :k]  # оставляем только первые k столбцов матрицы U
    E_k = np.diag(E[:k])  # оставляем только первые k элементов диагонали матрицы E
    V_k = V[:k, :]  # оставляем только первые k строк матрицы V
    return U_k @ E_k @ V_k  # перемножаем матрицы и возвращаем полученную матрицу


def convert_to_img(img_array, k):
    img = Image.fromarray(img_array)  # создаем изображение из матрицы
    img.convert('RGB').save(f'banana ducks_{k}.jpg')  # сохраняем изображение в файл


img = Image.open('banana ducks_bw.jpg')  # открываем изображение
img_array = np.asarray(img.convert('L'))  # преобразуем изображение в массив numpy

#         sqr  75%  50%  25%  20% 12.5% 10%  5%  1%
for k in [882, 810, 540, 270, 216, 135, 108, 54, 10]:
    img_array_k = svd_compress(img_array, k)
    convert_to_img(img_array_k, k)

```

В строках 6-11 описана функция `svd_compress`, принимающая на вход матрицу изображения и коэффициент k и возвращающая матрицу, полученную из произведения укороченного SVD. В строках 15-18 описана функция `convert_to_img`, которая принимает на вход матрицу изображения, преобразует её обратно в изображение и сохраняет его в файл. Затем мы открываем наше грустное изображение и балуемся с ним, оставляя в нём всё меньше и меньше сингулярных чисел от матрицы 882×882 до 10×10.

Для исходного изображения было необходимо хранить в полном SVD 1945206 чисел, что сильно больше 952560, однако в reduced-SVD (с отрезанными лишними нулями в матрице Σ) хранится уже 1731366 чисел. А можно ли ещё сжать? Конечно! Под каждым из изображений я оставлю подпись, содержащую процент сжатия, выбранное k и количество чисел, необходимое для хранения сжатого изображения. Ну что, поехали!

Рис. 3: Сжатие на 18.33% ($k = 882$), 1731366 чисел.

Рис. 4: Сжатие на 25% ($k = 810$), 1590030 чисел.

Рис. 5: Сжатие на 50% ($k = 540$), 1060020 чисел.

Рис. 6: Сжатие на 75% ($k = 270$), **530010** чисел.

Рис. 7: Сжатие на 80% ($k = 216$), 424008 чисел.

Рис. 8: Сжатие на 87.5% ($k = 135$), 265005 чисел.

Рис. 9: Сжатие на 90% ($k = 108$), 212004 чисел.

Рис. 10: Сжатие на 95% ($k = 54$), 106002 чисел.

Рис. 11: Сжатие на 99.07% ($k = 10$), 19630 чисел.

И что же это получается, декомпозиция Перегудина? Оставляю этот вопрос на размышление читателю... А я пойду поплачу, потому что мне жалко этого красавчика с двумя банановыми утками.

Подводим выводы, делаем итоги

- Чем меньше k, тем меньше сингулярных чисел формируют матрицу и тем меньше она похоже на изначальную, но из-за этого больше сжатие, которое ухудшает качество изображения.
- Фактически вес файла не изменяется, т.к. JPG и так использует оптимальные технологии сжатия, но если бы мы строили свою технологию сжатия, основанную на сингулярных разложениях, то вес в некоторых случаях можно заметно уменьшить без заметной глазу потери качества.
- Эффективными оказались сжатия на $75 - 87.5\%$. Они требуют для хранения меньше памяти, чем исходная чёрно-белая картинка, но при этом качество изображения остаётся достаточно приемлемым.
- Картинка всё ещё различима на сжатиях на $90 - 95\%$, но качество здесь гораздо хуже.
- Картинка полностью неразличима на сжатии на 99.07%, но при этом требуется всего 19630 чисел для хранения, что в 48.5 раз меньше, чем в исходной картинке.