

# Ad fontes, результати

## Опис даних

На вхід маємо будь-яке .jpg зображення. У процесі зображення перетворюється в матрицю (розмірності  $[m \times n]$  у випадку чорно-білого зображення і  $[m \times n \times 3]$  у випадку RGB.

Щоправда, автори в статті показують лише обробку чорно-білих та ще й тільки квадратних зображень! Оскільки часу було достатньо - ми зробили узагальнення і реалізували алгоритм і для прямокутних кольорових зображень RGB у два способи:

- на вхід RGB зображення, на вихід чорно-біле
- на вхід і на вихід зображення RGB

## Результати експерименту

Звіримо наші результати з результатами авторів

Імпортуємо функції

```
1 import sys
2 import matplotlib.pyplot as plt
3 sys.path.append('../')
4 from compressor.core import compressor, show_img
```

Функція для читання

```
1 def compute_path(name):
2     return '../img/' + name + '.jpg'
```

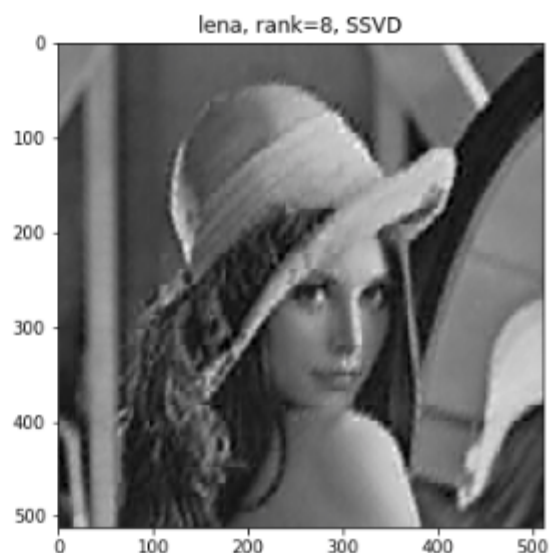
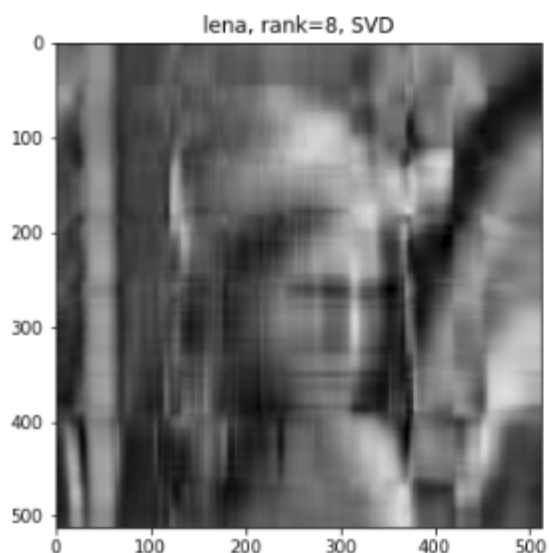
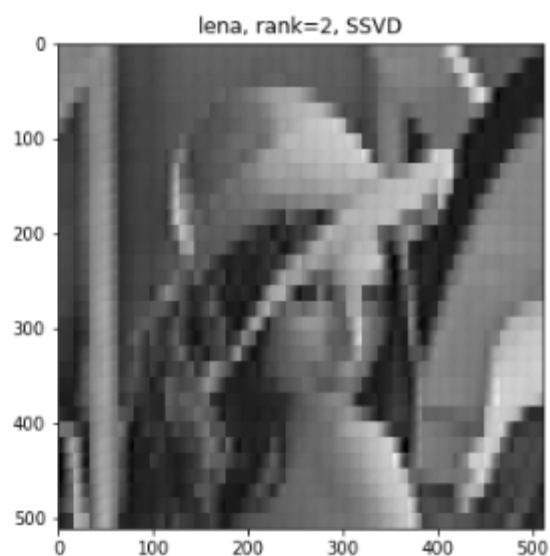
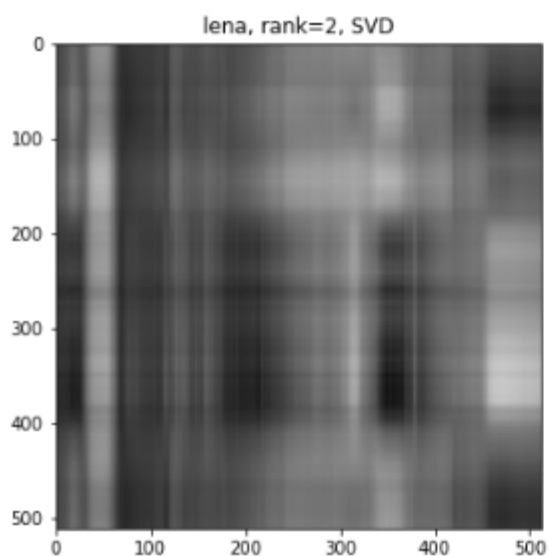


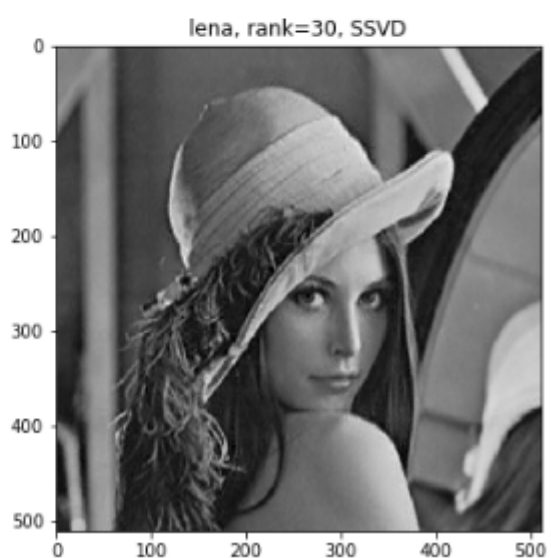
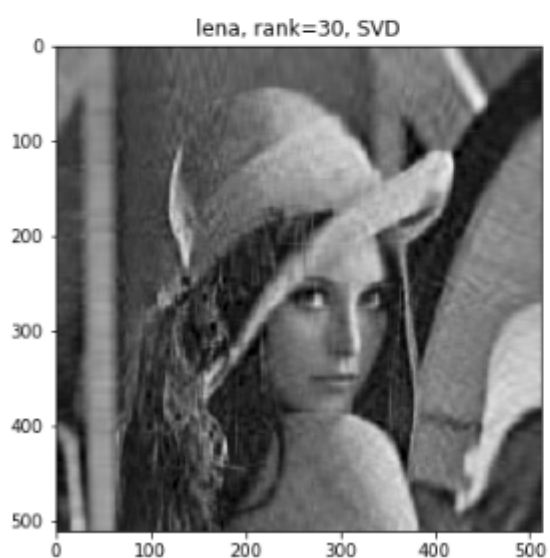
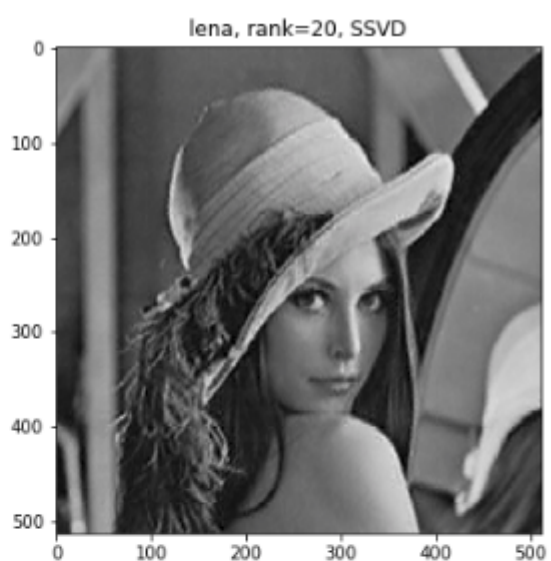
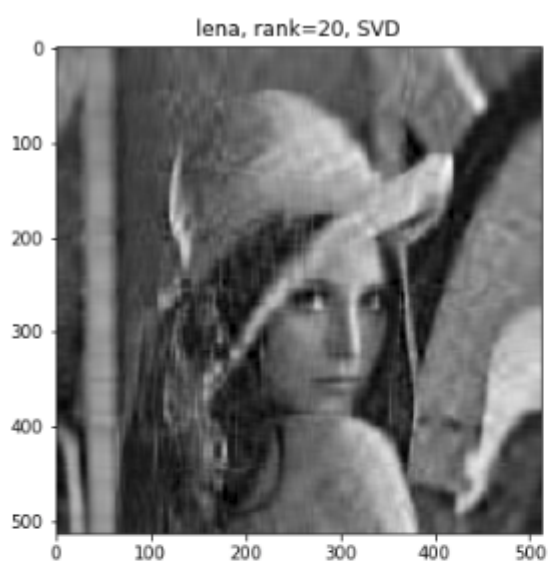
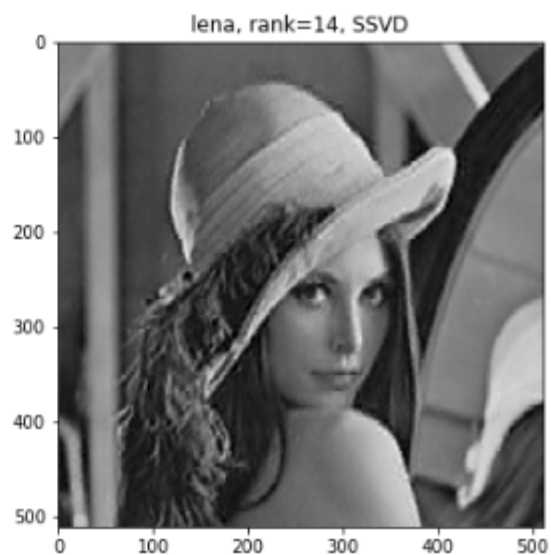
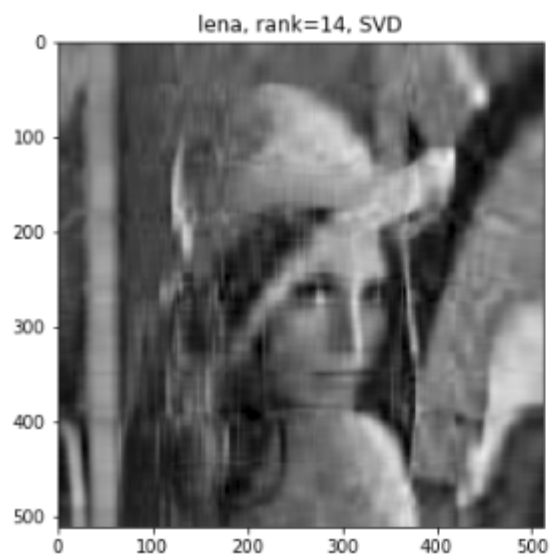
*lena.jpg* оригінальне зображення  
(512x512)

## Приклад виклику функцій

```
1  lena_svd2 = compressor(  
2      compute_path('lena'), # назва зображення  
3      rank=2, # ранг  
4      im_type='gray', # тип зображення, ще може бути 'rgb'  
5      compressor_type='SVD') # тип компресингу  
6  
7  lena_ssvd2 = compressor(  
8      compute_path('lena'),  
9      rank=2,  
10     im_type='gray',  
11     compressor_type='SSVD')  
12  
13  show_img(lena_svd2)  
14  show_img(lena_ssvd2)
```

Порівнюємо результати для різних рангів:



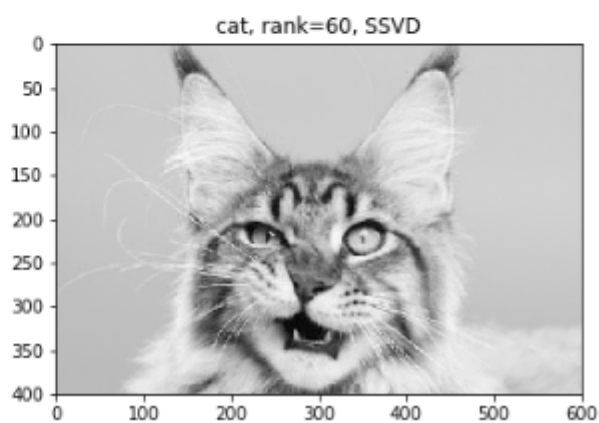
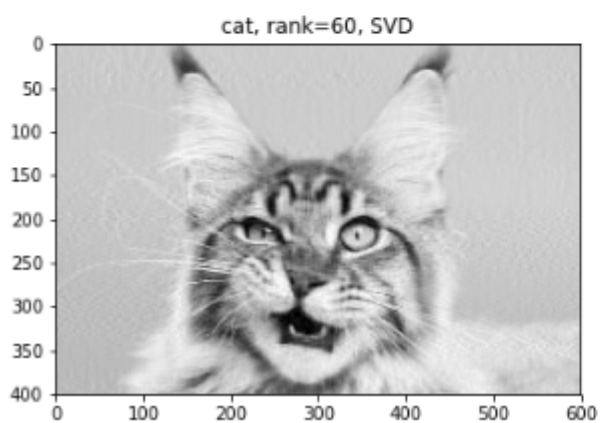
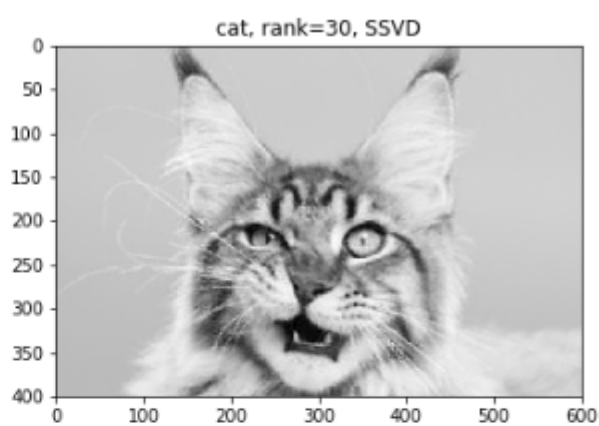
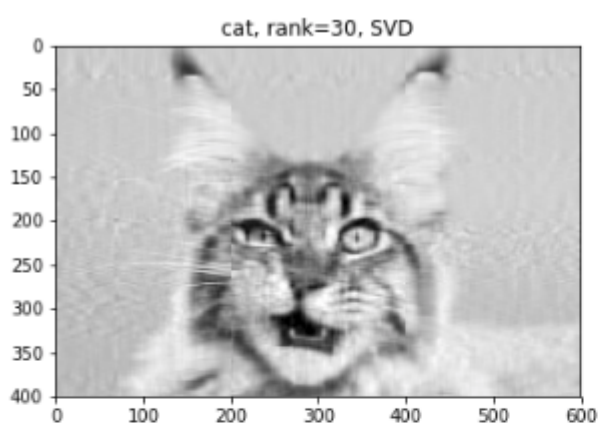


Результати нашого дослідження ідентичні результатам, які отримали автори.

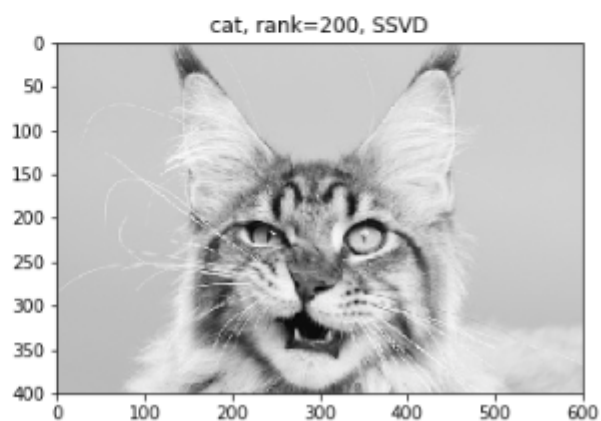
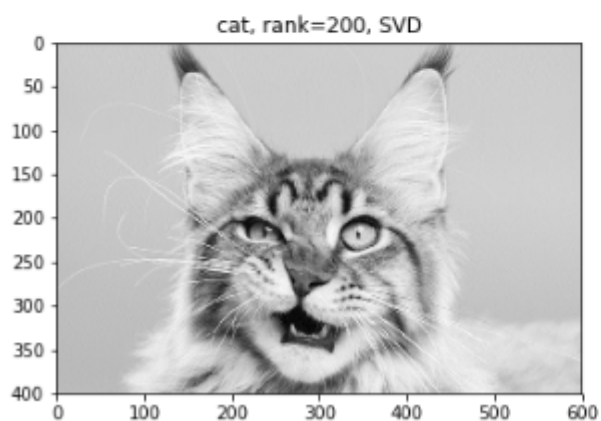
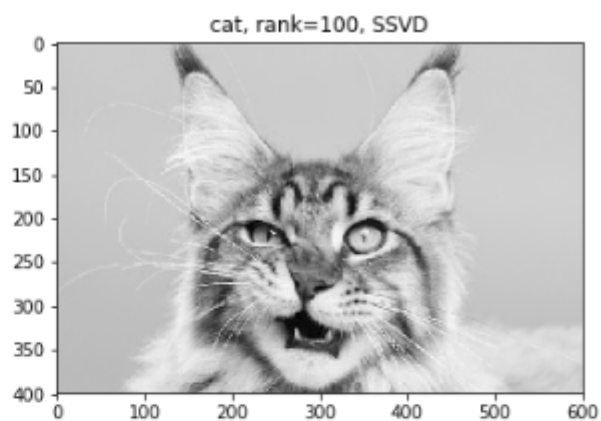
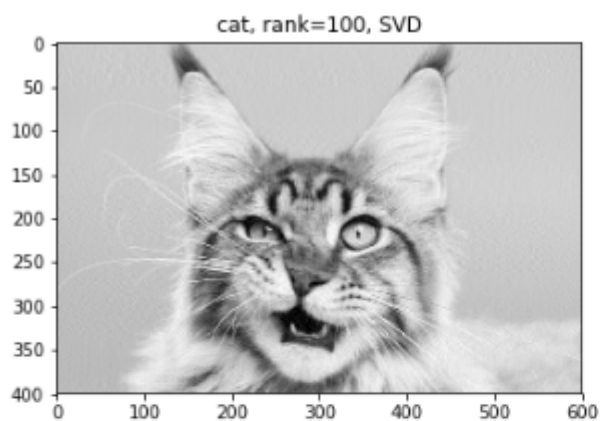
Для кольорових зображень



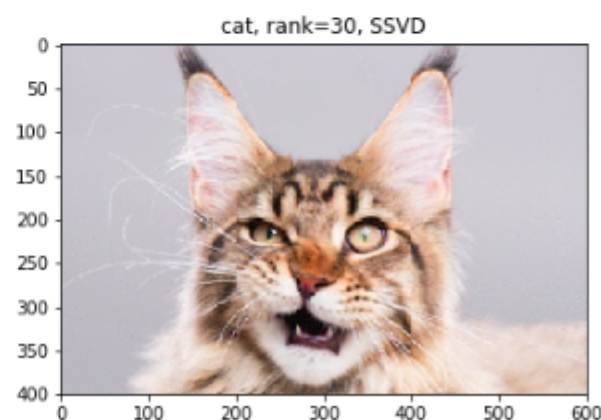
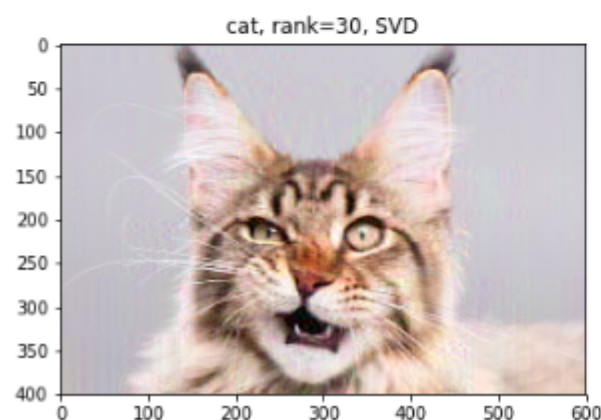
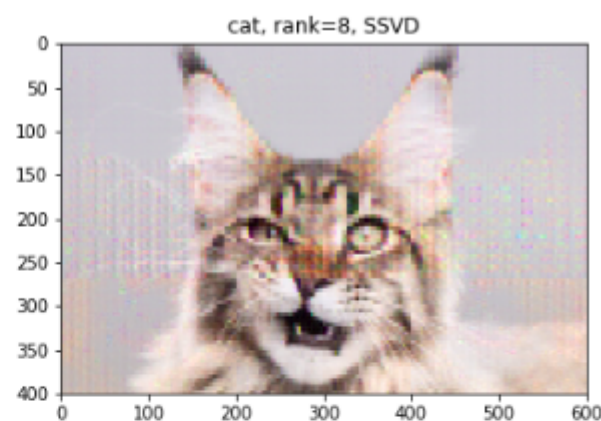
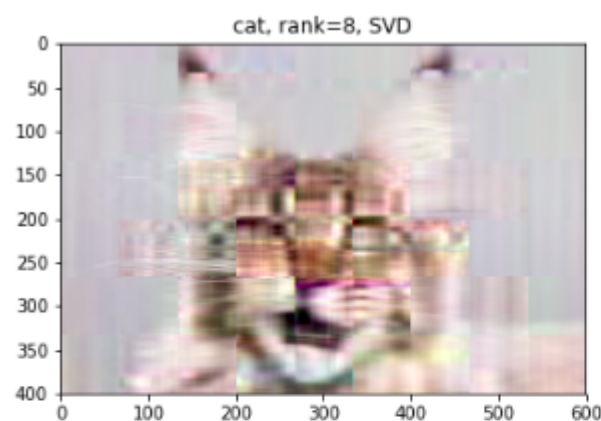
Оригінальне зображення (400x600)

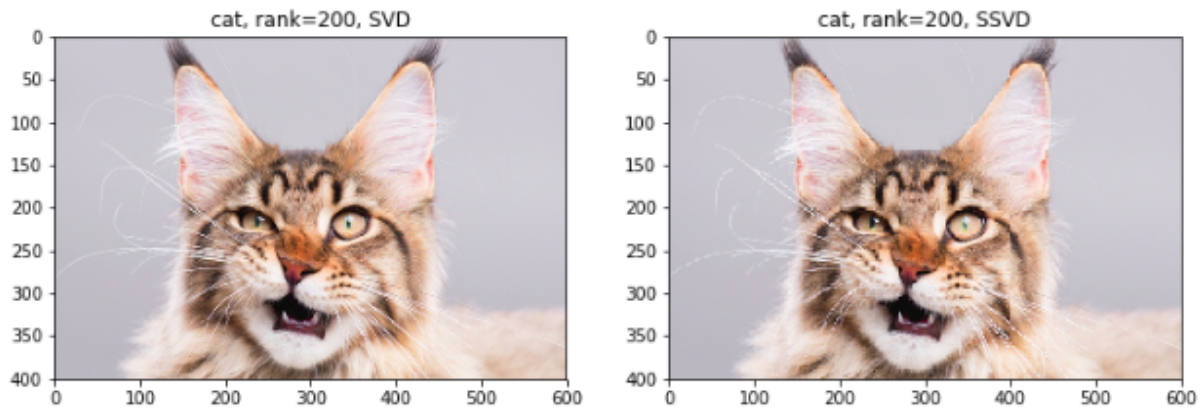






## RGB:





Останній приклад показує твердження авторів, що алгоритм SSVD гірше працює з точними (рівними, геометричними) фігурами. Вусики в кота на останньому зображенні у випадку використання SSVD мають гіршу якість, порівняно з SVD.

Подивитися детальніше на результати можна у файлах, які знаходяться в *experiment/results/*. Тест алгоритму знаходиться в файлі *experiment/experiment.ipynb* , приклад виклику функції:



```
visualize_results('lena', [2, 8, 14, 20, 30], im_type='gray', save=True, verbose=False)
```

```
1 # 1й аргумент - назва зображення
2 # 2. - тестові випадки (для рангу=2, рангу=8 ...)
3 # 3. тип зображення, яке повертати
4 # 4. чи зберігати зображення у файлі в папці experiment/results/
5 # 5. чи виводити в консоль повідомлення про прогрес
```

### Обрахунок економії пам'яті

Також, можна обчислити скільки пам'яті ми економимо, використовуючи цей алгоритм. Для порівняння ми використали зображення *lena.jpg* та метод бібліотеки NumPy, який зберігає файл у розширенні *.npz*; для рангу = 30, економія в пам'яті вийшла більше ніж у три рази.

$1025\text{KB} / 301\text{KB} = 3.4$

Ім'я	Дата змінення	Тип	Розмір
 original_image.npz	10.12.2017 18:00	Файл NPZ	1 025 КБ
 ssvd_image.npz	10.12.2017 18:06	Файл NPZ	301 КБ



SSVD, ранг = 30



Оригінальне зображення

Масиви можна знайти у *arrays* і за допомогою функції `np.load()` завантажити їх

## Висновки

---

### Чому навчились при розв'язанні цієї задачі?

- Ми зрозуміли основу алгоритму SVD, змогли покращити його результати для зображень за допомогою його варіації - SSVD-алгоритму.
- Розібралися як працювати з такими Python бібліотеками, як NumPy, Matplotlib, PIL і змогли все поєднати для того щоб реалізувати ці алгоритми на практиці.
- Навчились правильно зчитування зображення, представляти зображення як матрицю і матрицю як зображення для візуалізації результатів.
- Написали конвертацію кольорового зображення в чорно-біле, а також реалізувати алгоритм для кольорових зображень, розкладаючи їх на спектри RGB.

### Що викликало найбільшу складність?

- Зрозуміти як працює SVD і чому це працює.
- Автори наводили як приклад квадратну чорно-білу картинку. Формули і пояснення складали відповідно до цього, що викликало труднощі при реалізації алгоритму перетасовки для прямокутного зображення.
- Найбільшу складність викликало узагальнення для кольорових зображень.

### Що лишилося незрозумілим?

- "A bit allocation strategy" пункт в оригінальній статті

**Виконали Роман Вей та Забульський Володимир**