Применение сингулярного разложения к сжатию изображений

Мильчевская Александра

БЭК-243

Немного теории

Сингулярным разложением (SVD, Singular value decomposition) матрицы $A \in \mathrm{Mat}_{m\times n}(\mathbb{R})$ называется ее представление в виде

$$A = U\Sigma V^T,$$

где $\Sigma \in \mathrm{Mat}_{m\times n}(\mathbb{R}^+)$ — диагональная матрица, элементы главной диагонали которой — сингулярные числа матрицы A; $U \in \mathrm{Mat}_{m\times m}(\mathbb{R})$, $V \in \mathrm{Mat}_{n\times n}(\mathbb{R})$ — ортогональные матрицы, элементы которых — левые и правые сингулярные вектора.

Фробениус

Введём норму Фробениуса матрицы как

$$\|A\|_f = \sqrt{\operatorname{tr}(A^T A)},$$

где $\operatorname{tr}(A^T A)$ — след матрицы $A^T A$. Обозначим через A_r матрицу ранга $r < \operatorname{rang} A$. Возникает вопрос: как найти матрицу A_r наименее отличающуюся от A по норме Фробениуса (т.е. найти такую A_r, что $\|A - A_r\|_f$ будет минимальна). Это можно сделать с помощью сингулярного разложения.

Теорема. Пусть Σ_r — матрица полученная из Σ заменой части диагональных элементов нулями: $\sigma_{ii} = 0$, $i > r$. Тогда $A_r = U\Sigma_r V^T$.

Последнее равенство можно переписать еще в более экономичном виде: $A_r = U_r\hat{\Sigma}_r V_r^T$, где матрицы U_r, V_r и Σ_r получаются из U, V, Σ_r отсечением неиспользуемых элементов:

$$U_r = \begin{pmatrix} u_{11} & \dots & u_{1r} \\ \vdots & \ddots & \vdots \\ u_{m1} & \dots & u_{mr} \end{pmatrix}, \ V_r = \begin{pmatrix} v_{11} & \dots & v_{n1} \\ \vdots & \ddots & \vdots \\ v_{1r} & \dots & v_{nr} \end{pmatrix}, \ \hat{\Sigma}_r = \begin{pmatrix} \sigma_{11} & & \\ & \ddots & \\ & & \sigma_{rr} \end{pmatrix}$$

Если сингулярные значения матрицы убывают достаточно быстро (а, оказывается, в реальных задачах часто это именно так), то норма разности будет малой при небольшом значении r.

Очевидно, что вместо хранения исходной матрицы A (размера $m \times n$) можно хранить матрицы U_r и V_r и диагональные элементы матрицы $\hat{\Sigma}_r$ (т.е. вместо хранения $m \times n$ элементов мы будем хранить $mr + nr + r = r(m + n + 1)$ элементов, где r мало). На этом основано сжатие данных с помощью SVD разложения.

План сжатия

- **Черно-белое изображение**
  1)Конвертируем в оттенки серого
  2)Применяем SVD к матрице интенсивности
  3)Подбираем rank k, сохраняющий 99% информации(или примерно определяем по «локтю»)
  4)Восстанавливаем из усеченных матриц
- **Цветное изображение**
  1)Разделяем на R, G, B каналы
  2)Для каждого канала применяем SVD
  3)Используем одинаковый k для всех каналов (99% информации)
  4)Собираем сжатые каналы в итоговое изображение

Преобразование картинки в серую

```python
def rgb2gray(rgb):
    return np.dot(rgb[...,:3], [0.2989, 0.5870, 0.1140])

image = img.imread(TEST_IMAGE_PATH)

image_grayscale = rgb2gray(image)

print(image.shape, image_grayscale.shape)

plt.imshow(image)
plt.show()

plt.imshow(image_grayscale, cmap=plt.get_cmap('gray'))

plt.show()
```

Было - стало


0
200
400
600
800
1000
1200
1400
0
250
500
750
1000
1250
1500
1750

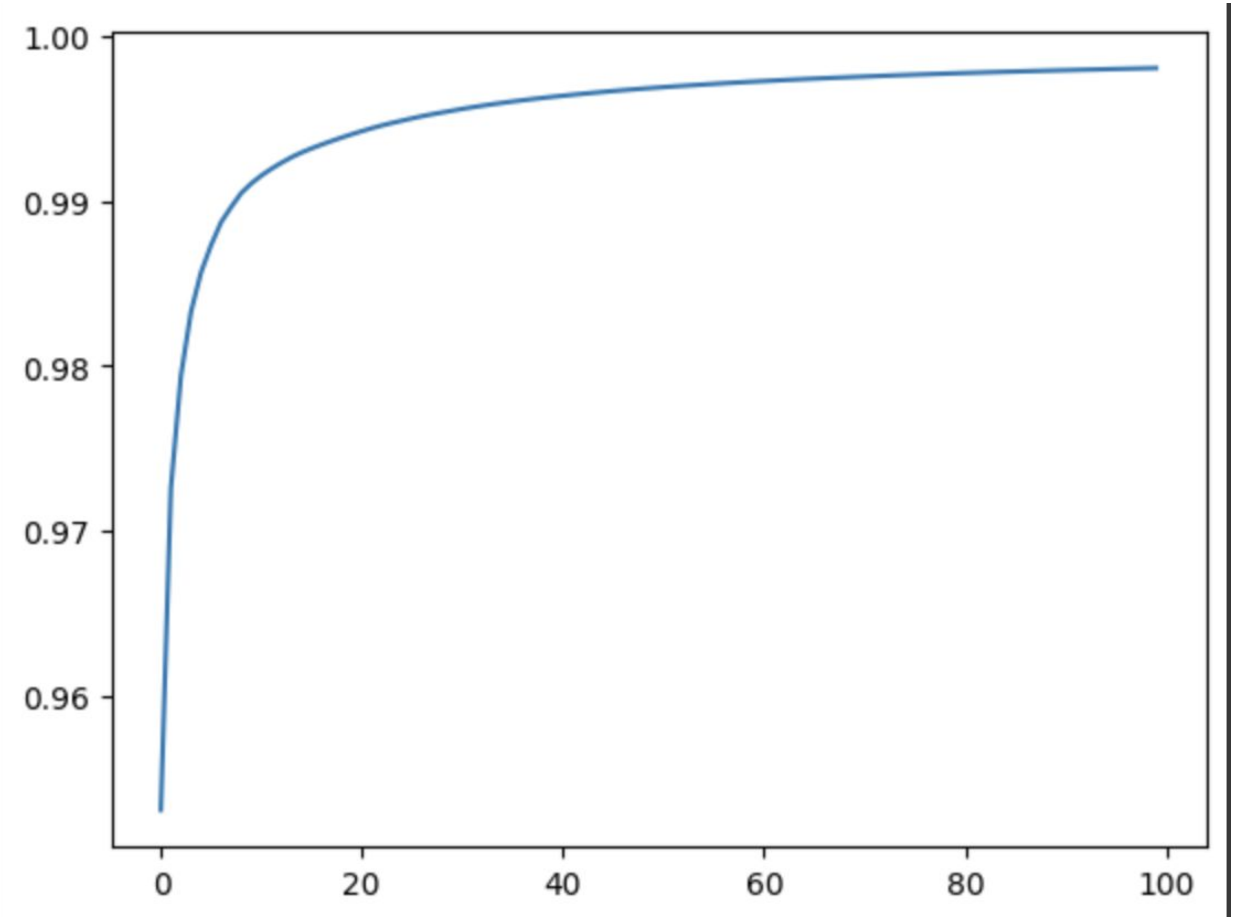

0
200
400
600
800
1000
1200
1400
0
250
500
750
1000
1250
1500
1750

Ищем SVD

```
def svd_reconstruct(U, S, Vh, num_eigen_values=None):
    rank = S.size
    S_partial = np.copy(S)
    if num_eigen_values is not None:
        S_partial[num_eigen_values:] = 0


    return U[:, :rank] @ np.diag(S_partial) @ Vh[:rank, :]


U, S, Vh = np.linalg.svd(image_grayscale, full_matrices=True)

print(U.shape, S.shape, Vh.shape)

image_grayscale_reconstructed = svd_reconstruct(U, S, Vh)

print(np.allclose(image_grayscale_reconstructed, image_grayscale))

plt.imshow(image_grayscale_reconstructed, cmap=plt.get_cmap('gray'))
```

Это мы применим к серой картинке и по отдельности к каждому из RGB в цветной

График «Локтя»

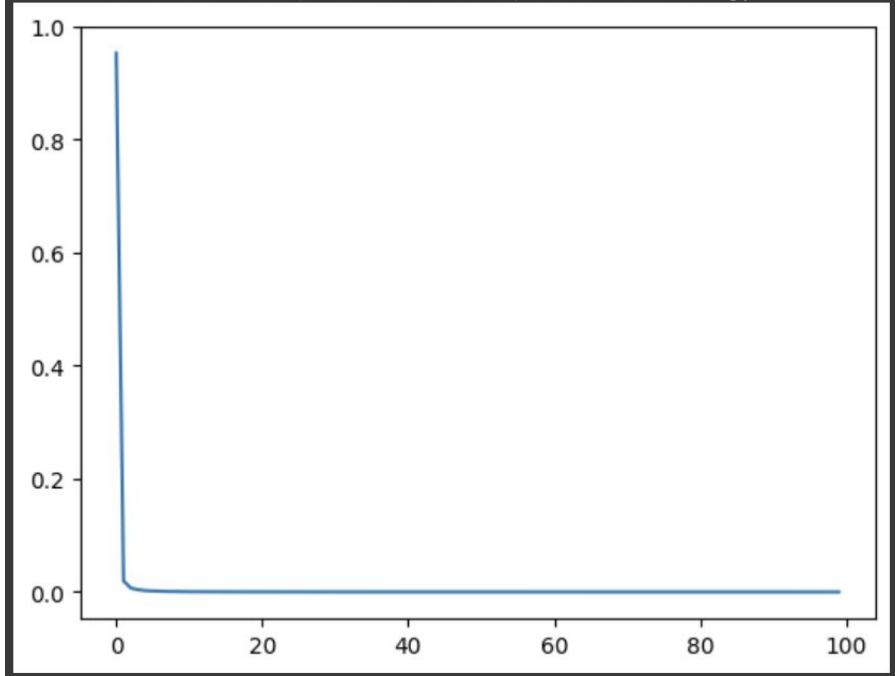


Все, что после 60-80 не сильно меняет «точность», но мы определим не на глаз, а с учетом необходимой для конкретной задачи точности. В нашем проекте предположим, что 99% «с головой» хватит (в сереньком мы, ради интереса, посмотрим, что будет если взять разный ранг)

Кол - с помощью Фробениуса построил

```
S_normalized = np.square(S) / np.sum(np.square(S))

plt.plot(S_normalized[:100])

S_normalized
```

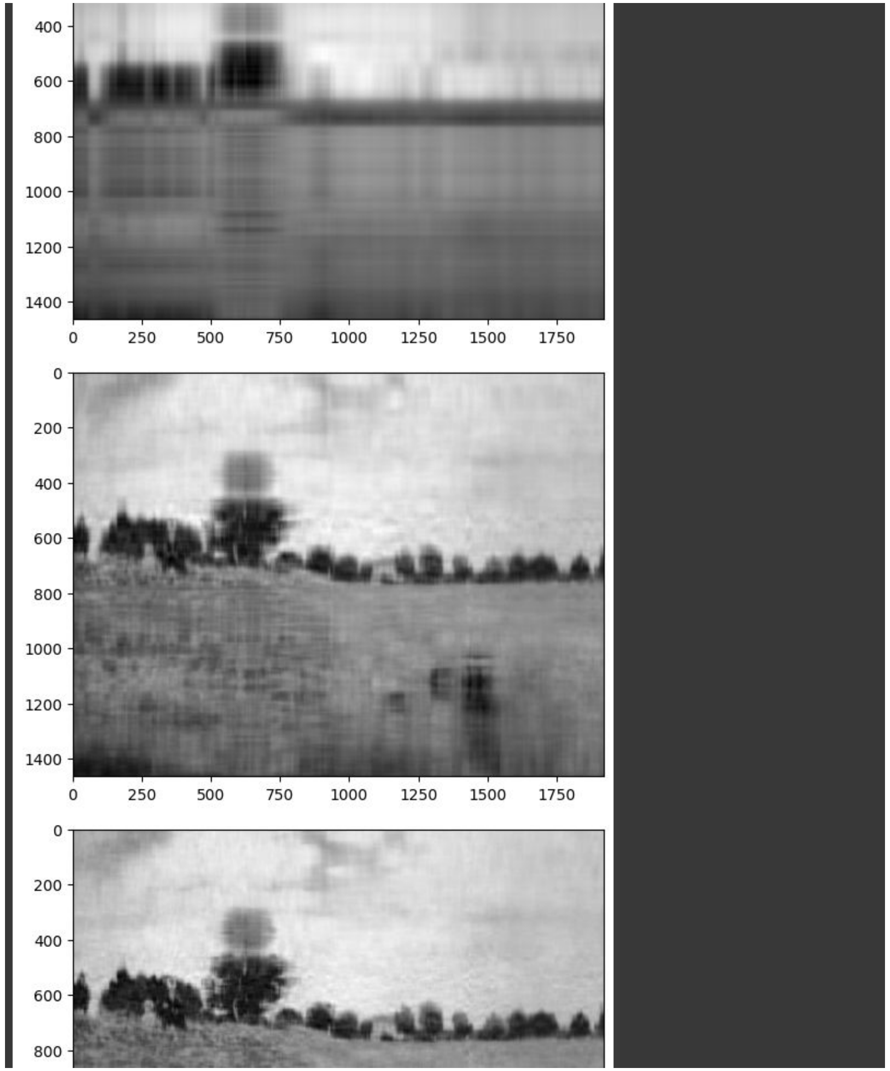
```
array([9.53118545e-01, 1.94907098e-02, 6.80418233e-03, ...,
       6.08410005e-09, 5.78937479e-09, 5.54187886e-09])
```




```
plt.plot(np.cumsum(S_normalized[:100]))
```

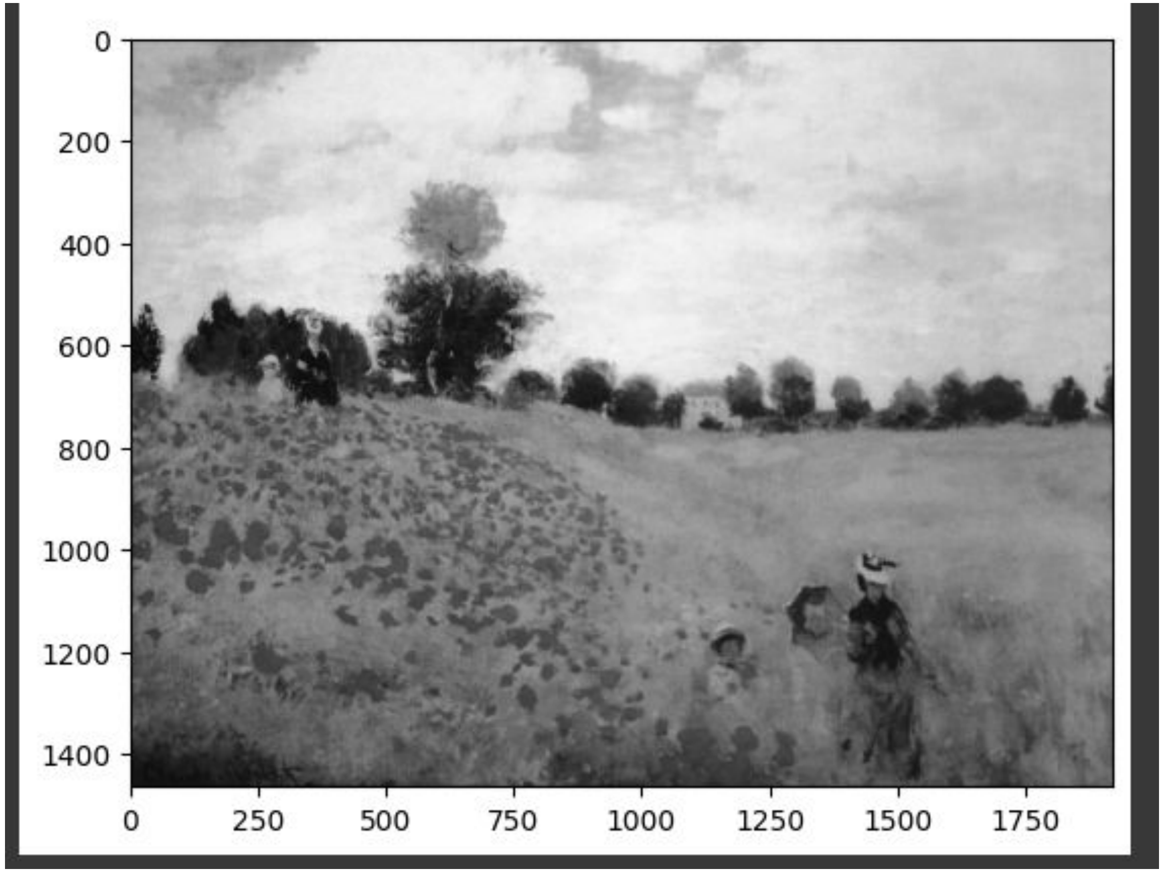
```
[<matplotlib.lines.Line2D at 0x7b53721d6180>]
```

Разные ранги – разная «размытость»



Итог

- В случае с ЧБ картинкой я решила взять просто достаточно большой ранг (200 из 1464), чтобы картинка получилась достаточно точной
- В цветном поступим строже – сделаем точность 99%



Цветные

- С цветными сделаем разложения для RGB(разных цветов) и сожмем с «точностью» 99%

```
im0=image[:, :,0]
im1=image[:, :,1]
im2=image[:, :,2]


U0, S0, Vh0 = np.linalg.svd(im0, full_matrices=True)
U1, S1, Vh1 = np.linalg.svd(im1, full_matrices=True)
U2, S2, Vh2 = np.linalg.svd(im2, full_matrices=True)
```

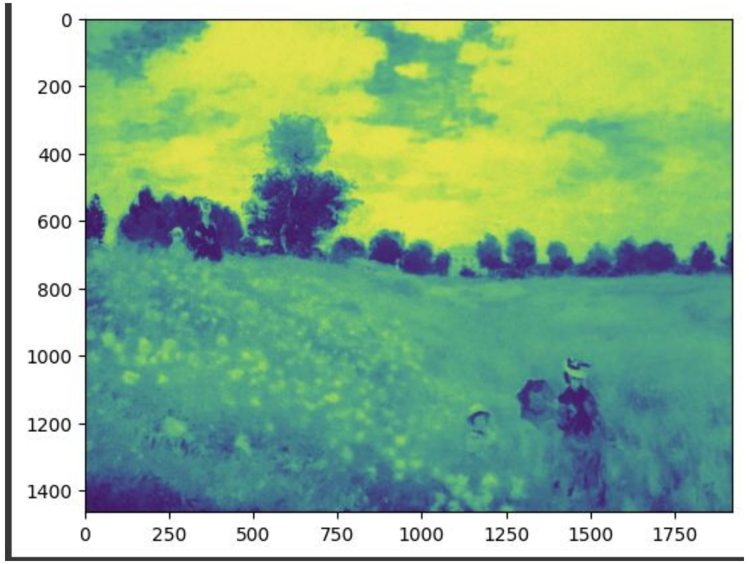
```
S1_normalized = np.square(S1) / np.sum(np.square(S1))
for i in range(0, len(S1_normalized)):
    if np.cumsum(S1_normalized)[i] > 0.999:
        print("Index:", i)
        break
```

```
Index: 260
```

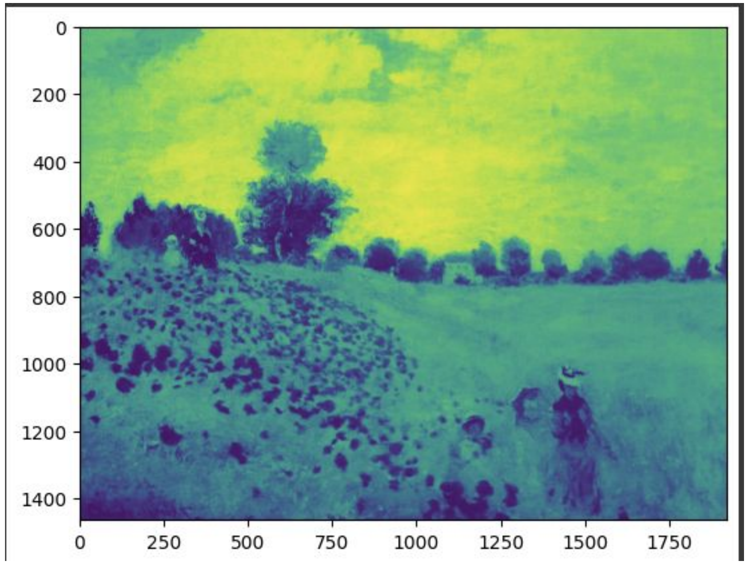
```
S2_normalized = np.square(S2) / np.sum(np.square(S2))
for i in range(0, len(S2_normalized)):
    if np.cumsum(S2_normalized)[i] > 0.999:
        print("Index:", i)
        break
```

В каждом случае мы получили «индекс» - ранг, с ним мы и сожмем каждую из картинок

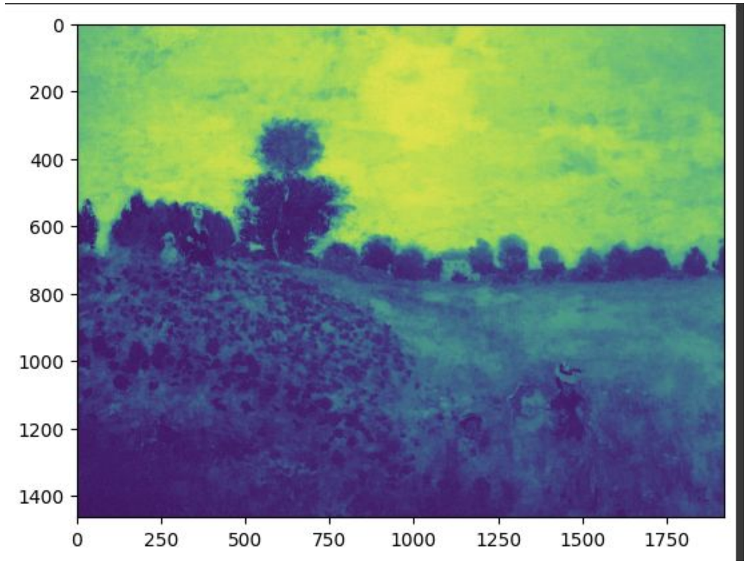
Картинки после сжатия



R



G



B

Соединяем и ито

```
imr= np.zeros_like(image)
imr[:, :, 0] = im0r
imr[:, :, 1] = im1r
imr[:, :, 2] = im2r

plt.imshow(imr)
plt.show()
```