

بسمه تعالى

تحقيق در مورد الگوریتم SVD و فشرده سازی تصویر توسط آن

Singular Value Decomposition

سید وحیدرضا ابن‌الرضا

Email: Ebnoreza@outlook.com

فشرده سازی فرآیندی وابسته به موضوع (oriented subject) می باشد. هیچ الگوریتم فشرده سازی نمی تواند به ادعای بهینه بودن در کلیه شرایط را داشته باشد عملیات فشرده سازی کامال سفارشی می باشد.

فشرده سازی تصویر کاربردی از فشرده سازی اطلاعات در تصاویر دیجیتال است. هدف از آن کاهش اطلاعات بلا استفاده برای ذخیره کردن یا انتقال اطلاعات به شکل بهینه می باشد.

فشرده سازی تصویر می تواند به صورت بی اتلاف (Lossless) و با اتلاف (Lossy) صورت گیرد. فشرده سازی با اتلاف خصوصاً وقتی برای نرخ بیت های پایین استفاده شود به کیفیت تصویر لطمه می زند. روش های فشرده سازی بی اتلاف همچنین ممکن است برای محتويات پر ارزش مثل تصاویر پزشکی یا تصاویر اسکن شده برای باقیانی شدن نیز ترجیح داده شوند.

یکی از راه هایی که میتوان با استفاده از آن موجب کاهش اطلاعات و فشرده سازی تصاویر شد، استفاده از الگوریتم SVD (singular value decomposition) میباشد.

SVD چیست؟

روش تجزیه مقادیر منفرد یا SVD تاریخچه طولانی و البته جالبی دارد. این روش، در علوم اجتماعی و با تست هوش شروع شد. در گذشته، پژوهشگران آزمون های را برای اندازه گیری جنبه های مختلف هوش مانند هوش فضایی و هوش کلامی ارائه کردند که همبستگی زیادی با هم داشتند. آنها بر این باور بودند که یک معیار عمومی مشترک برای هوش وجود دارد و آن را برای هوش کلی، g نامیدند که امروزه بیشتر به عنوان آی کیو (IQ) شناخته می شود. بنابراین، پژوهشگران تصمیم گرفتند عوامل مختلف هوش را تجزیه و مهم ترین آنها را انتخاب کنند. در ساده ترین حالت، تجزیه مقادیر منفرد به نوعی همین تجزیه و انتخاب مهم ترین عوامل یک ماتریس را انجام می دهد.

امروزه، تجزیه مقادیر منفرد در بسیاری از شاخه های علوم و نجوم کاربرد دارد. این روش، در یادگیری ماشین و آمار توصیفی و مدل سازی آماری نیز بسیار مورد استفاده قرار می گیرد.

همان طور که می توانیم یک عدد را به اعداد اول تجزیه کنیم، همین کار را هم می توانیم برای ماتریس ها انجام دهیم. یعنی یک ماتریس را به عوامل سازنده ای آن تجزیه کنیم.

$$A = USV^T$$

که در آن:

- A یک ماتریس $m \times n$
- U یک ماتریس متعامد $m \times m$
- S یک ماتریس قطری $m \times n$
- V یک ماتریس متعامد $n \times n$ است.

شکل بصری ابعاد ماتریس ها به صورت زیر است:

$$m[n] = m[m] \cdot m[m] \cdot m[n] \cdot n[n]$$

در واقع ماتریس اصلی (Factorization) ماتریس A تشکیل شده از ضرب این سه ماتریس U و S و ترانهادهی V است. به این کار در اصلاح تقسیم‌بندی می‌گویند و به این معنی است که ماتریس A می‌تواند از سه ماتریس U و S و V تشکیل شود.

در زیر نمونه‌ای از تقسیم‌بندی (Factorization) ماتریس A به سه ماتریس را نشان داده‌ایم:

$$\begin{bmatrix} 2 & -2 & 1 \\ 5 & -2 & 1 \end{bmatrix} = \begin{bmatrix} -13.9 & -9.510 \\ -19.51 & -3.910 \end{bmatrix} \begin{bmatrix} 7.77 & 0 & 0 \\ 0 & 2.287 & 0 \\ 0 & 0 & 0.178 \end{bmatrix} \begin{bmatrix} -7.93 & -1.57 & -1.588 \\ -1.57 & 1.79 & -0.97 \\ -1.588 & -0.97 & 1.78 \end{bmatrix}$$

ماتریس اصلی A
 ماتریس مقابله‌دار U
 ماتریس S
 ماتریس V

پروژه

در این پروژه سعی کردیم تا با استفاده از الگوریتم SVD ابتدا یک تصویر سیاه و سفید (ماتریس دارای یک بعد) را فشرده سازی کنیم و در ادامه با تعمیم این الگوریتم برای یک تصویر رنگی (3 بعد ماتریس RGB) این فشرده سازی را تکرار کنیم.

هدف

هدف در این پروژه پیاده سازی ماتریس تصویر A به 3×3 بردار USV^T است و باز سازی تصویر با رنگ های متفاوت با استفاده از این ماتریس ها

پیاده سازی

```
[1]: from matplotlib.image import imread
import matplotlib.pyplot as plt
import numpy as np
import os
import pandas as pd

A = imread('shepherd.jpg')
X = np.mean(A, -1); #RGB to Gray/ Change 3 channel to 2 channel

dpi, height, width = 50, A.shape[0], A.shape[1]
plt.figure(figsize=(width/dpi,height/dpi))

img = plt.imshow(X)
# print(str(X.shape))
img.set_cmap('gray')
plt.axis('off')
plt.savefig('Gray_shepherd.jpg', bbox_inches='tight', pad_inches = 0)
plt.show()

# w = pd.DataFrame(X)
# print (w)
```

بعد از اضافه کردن کتابخانه ها عکس مورد نظر را لود کرده و با استفاده از آن را تبدیل به یک عکس سیاه و سفید می‌کنیم تا عکسی که دارای ماتریس $[::, ::, 3]$ به ماتریس $[::, ::, 1]$ بتوانیم تبدیل کنیم.

ابعاد پلات را مشخص کرده و عکس سیاه و سفید را به عنوان خروجی نمایش میدهیم.

خروجی: (بدلیل واضح بودن تفاوت تصاویر عکس خروجی ها کوچک نشده)



```
U, S, VT = np.linalg.svd(X)
S = np.diag(S)

print('Gray shape is: ' + str(X.shape))
X_sum = np.count_nonzero(X)
print('total array in orginal image: ' + str(f'{X_sum:,}')))

Gray shape is: (2417, 3753)
total array in orginal image: 9,070,326
```

سپس با استفاده از `np.linalg.svd` ماتریس عکس را به 3 ماتریس متعامد U , S , VT تبدیل میکنیم. (میدانیم که ضرب این 3 ماتریس به ما همان ماتریس اولیه X را خواهد داد. سری S را به یک ماتریس قطری تبدیل میکنیم. ماتریس X یک ماتریس با تعداد آرایه های 9,070,326 خواهد بود. یعنی برای بدست آمدن تصویر اصلی سیاه و سفید از این مقدار ارایه استفاده شده است.

```

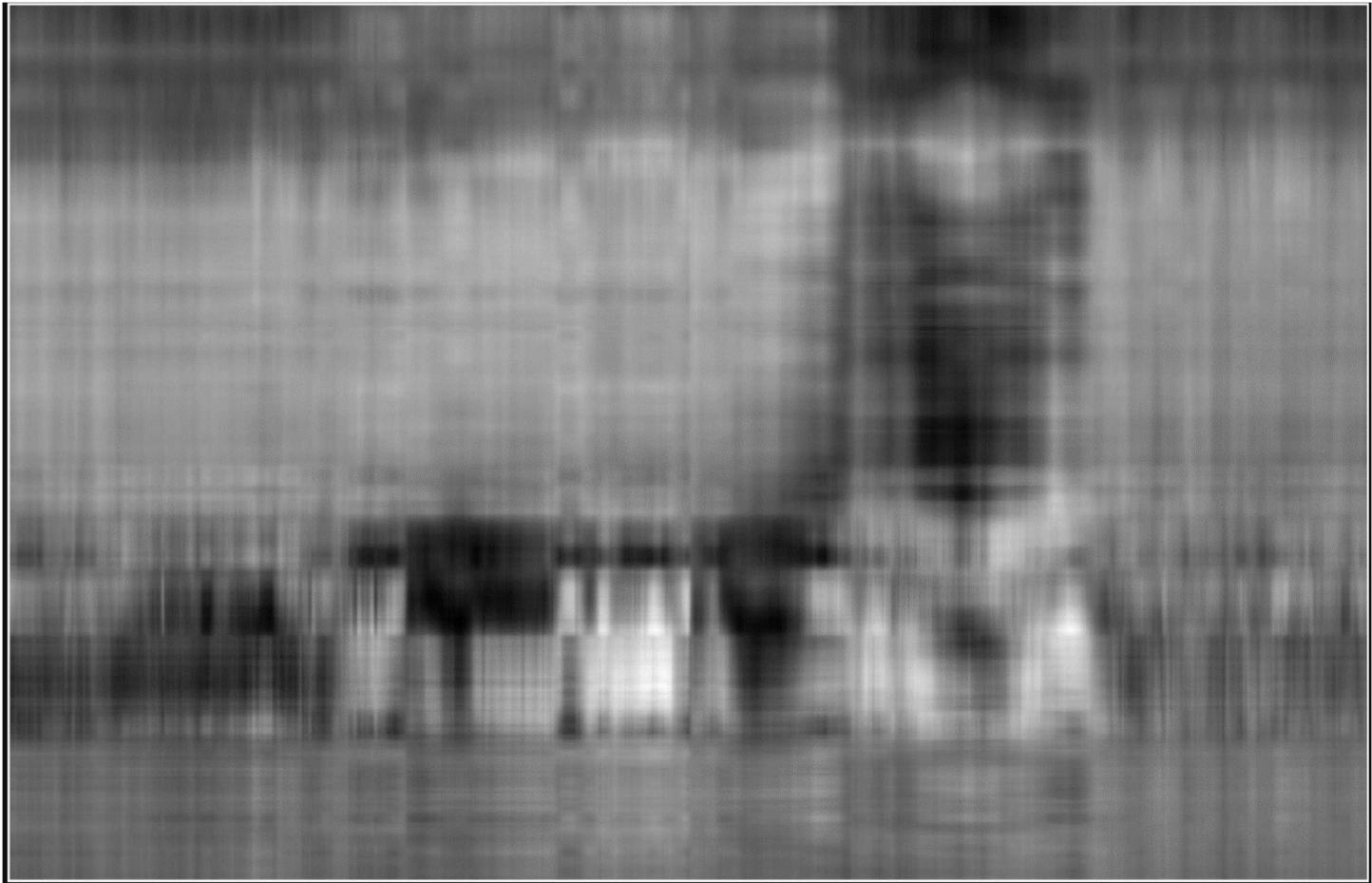
for r in (5, 25, 100, 400):
    # construct approximate aimage
    Xapprox = U[:,r:] @ S[:,r:] @ VT[:,r:]
    plt.figure(figsize=(width/dpi,height/dpi))
    img = plt.imshow(Xapprox)
    img.set_cmap('gray')
    plt.axis('off')
    # plt.title('r = ' + str(r))
    plt.savefig(str(r)+'.jpg', bbox_inches='tight', pad_inches = 0)
    plt.show()

print('image shape is: ' + str([np.shape(x) for x in [U[:,r:], S[:,r:], VT[:,r:]]]))
U_sum = np.count_nonzero(U[:,r:])
S_sum = np.count_nonzero(S[:,r:])
VT_sum = np.count_nonzero(VT[:,r:])
total_gray_arr = U_sum+S_sum+VT_sum
print('total array in image: ' + str(f'{total_gray_arr:,}'))
print('Compression Ration is: ' + str(f'{total_gray_arr/X_sum:.2%}'))

```

سپس برای 4 رنک متفاوت 5 و 25 و 100 و 400 سعی میکنیم ماتریس X را با استفاده از 3 ماتریس U , S , VT بازسازی کنیم. میدانیم که این مقدار هر چقدر بیشتر باشد تعداد آرایه های خروجی ما بیشتر شده و ما را به همان اندازه به تصویر اصلی بیشتر نزدیک میکند. اما همچنانی باید بدانیم که تعداد آرایه های بیشتر یعنی بیشتر شدن حجم تصویر فشرده شده. با استفاده از ضرب ماتریسی $Xapprox$ را ساخته و دوباره آن را ترسیم میکنیم و به عنوان خروجی تعداد آرایه های S , VT را شمارده و با جمع آن را با تعداد اصلی مقایسه میکنیم.

خروجی:



```

image shape is: [(2417, 5), (5, 5), (5, 3753)]
total array in image: 30,855
Compression Ration is: 0.340175%

```

به عنوان مثال برای رنک 5 یعنی ما ماتریس S را به طول 5 در نظر گرفته ایم و ضرب ماتریس ها به صورت $[(2417,5), (5,5), (5, 3753)]$ خواهد بود که در نهایت خروجی یک ماتریس $[2417, 3753]$ داریم اما ما این ماتریس را با استفاده از تنها 30,855 آرایه تولید کردیم. و نسبت تعداد آرایه های عکس با رنک 5 به آرایه های تصویر اصلی یعنی 30,855 برابر 0.34% شده است. البته که میبینیم انتخاب رنک 5 برای این تصویر زیاد خوب نبوده، چرا که کلی اطلاعات پاک شده و در باز سازی تصویر عملا نمیتوان درکی از تصویر اصلی را از آن داشت. پس این کار را با رنک های بالاتر تکرار کردیم.



```
image shape is: [(2417, 25), (25, 25), (25, 3753)]  
total array in image: 154,275  
Compression Ration is: 1.700876%
```

کم کم با افزایش رنک و بزرگ کردن ماتریس S (و البته بقیه ماتریس های سطrix و ستونی) اطلاعات بیشتر شده و باز سازی تصویر بهتر. در اینجا با نسبت فشرده سازی 1.70%



image shape is: [(2417, 100), (100, 100), (100, 3753)]

total array in image: 617,100

Compression Ration is: 6.803504%



image shape is: [(2417, 400), (400, 400), (400, 3753)]

total array in image: 2,468,400

Compression Ration is: 27.214016%

در انتها میبینیم که با رنک حدود 400 و با تقریبا نسبت فشرده سازی 27% توانستیم به یک تصویر قابل قبول از تصویر اصلی دست یابیم. همچنین در خروجی برنامه میتوان حجم فایل های ذخیره شده در کامپیوتر را مشاهده کرد که به ترتیب برای تصویر اوژینال و سپس تصاویر از کیفیت بالا به کیفیت پایین تر برابر است با: 3.29Mb , 2.31Mb , 1.66Mb , 1.28Mb , 989Kb .

میبینیم که همچنین با کم کردن تعداد آرایه ها در بازسازی تصویر توانستیم که علاوه بر داشتن یک خروجی قابل قبول حجم اطلاعات را هم فشرده کرد.

حال که این کار را کردیم سعی میکنیم تا تصویر را به صورت رنگی و برای هر 3 ماتریس RGB تکرار کنیم و با تلفیق آنها در انتها دوباره تصویر اصلی را بازسازی کنیم.

```

plt.figure(figsize=(width/dpi,height/dpi))

plt.imshow(A)
plt.axis('off')
plt.savefig('Color_shepherd.jpg', bbox_inches='tight',pad_inches = 0)
plt.show()

print('color shape is: ' + str(np.shape(A)))
A_sum = np.count_nonzero(A)
print('total array in orginal image: ' + str(f"{A_sum:,}"))

```

```

mypic_Red=A[:, :, 0]
mypic_Green=A[:, :, 1]
mypic_Blue=A[:, :, 2]

U_Red, S_Red, VT_Red = np.linalg.svd(mypic_Red)
U_Green, S_Green, VT_Green = np.linalg.svd(mypic_Green)
U_Blue, S_Blue, VT_Blue = np.linalg.svd(mypic_Blue)

```

با مشخص کردن ابعاد اندازه تصویر و نمایش آن ابعاد ماتریس آن را بدست می‌اوریم. میبینیم که ماتریس اصلی تصویر رنگی به صورت [2417, 3753, 3] میباشد. پس 3 ماتریس بعد سوم را به سه ماتریس دو بعدی فرمز و سبز و آبی تبدیل میکنیم و هر ماتریس دوباره جدا گانه به سه ماتریس U, S, VT میباشد. یعنی در انتها به 9 ماتریس دو بعدی.



```

color shape is: (2417, 3753, 3)
total array in orginal image: 27,115,093

```

میبینیم که تصویر اصلی دارای 27,115,093 آرایه برای تشکیل خود میباشد. پس با استفاده از 9 ماتریس ساخته شده و کاوش رنک آن سعی در بازسازی تصویر با تعداد آرایه های کمتر میکنیم.

```

S_Red = np.diag(S_Red)
S_Green = np.diag(S_Green)
S_Blue = np.diag(S_Blue)
# print('x shape is: ' + str(S_Red.shape))

for r in (5, 25, 100, 400):
#     construct approximate aimage
    Redapprox = U_Red[:, :, r] @ S_Red[:, r, :] @ VT_Red[:, r, :]
    Greenapprox = U_Green[:, :, r] @ S_Green[:, r, :] @ VT_Green[:, r, :]
    Blueapprox = U_Blue[:, :, r] @ S_Blue[:, r, :] @ VT_Blue[:, r, :]

    Redapprox=Redapprox.astype(int)
    Greenapprox=Greenapprox.astype(int)
    Blueapprox=Blueapprox.astype(int)

plt.figure(figsize=(width/dpi,height/dpi))
compressed_array=np.stack((Redapprox, Greenapprox, Blueapprox), axis=2)

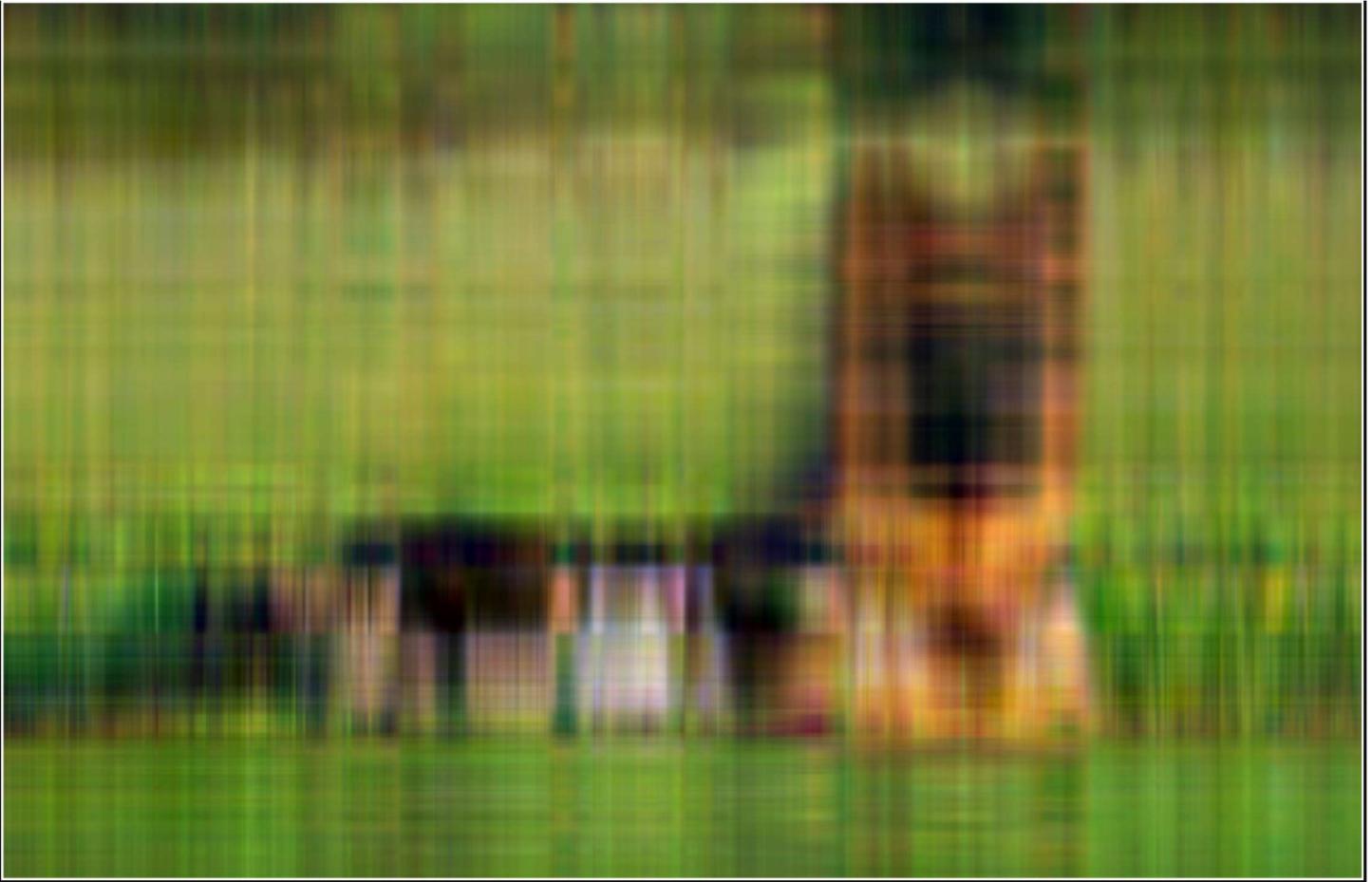
compressed_array = compressed_array/np.amax(compressed_array)
compressed_array = np.clip(compressed_array, 0, 1)

img = plt.imshow(compressed_array)
#     img.set_cmap('gray')
plt.axis('off')
#     plt.title('r = ' + str(r))
plt.savefig('Color_' + str(r) + '.jpg', bbox_inches='tight', pad_inches = 0)
plt.show()

print('image shape for Red is: ' + str([np.shape(x) for x in [U_Red[:, :, r], S_Red[:, r, :], VT_Red[:, r, :]]]))
print('image shape for Green is: ' + str([np.shape(x) for x in [U_Green[:, :, r], S_Green[:, r, :], VT_Green[:, r, :]]]))
print('image shape for Blue is: ' + str([np.shape(x) for x in [U_Blue[:, :, r], S_Blue[:, r, :], VT_Blue[:, r, :]]]))

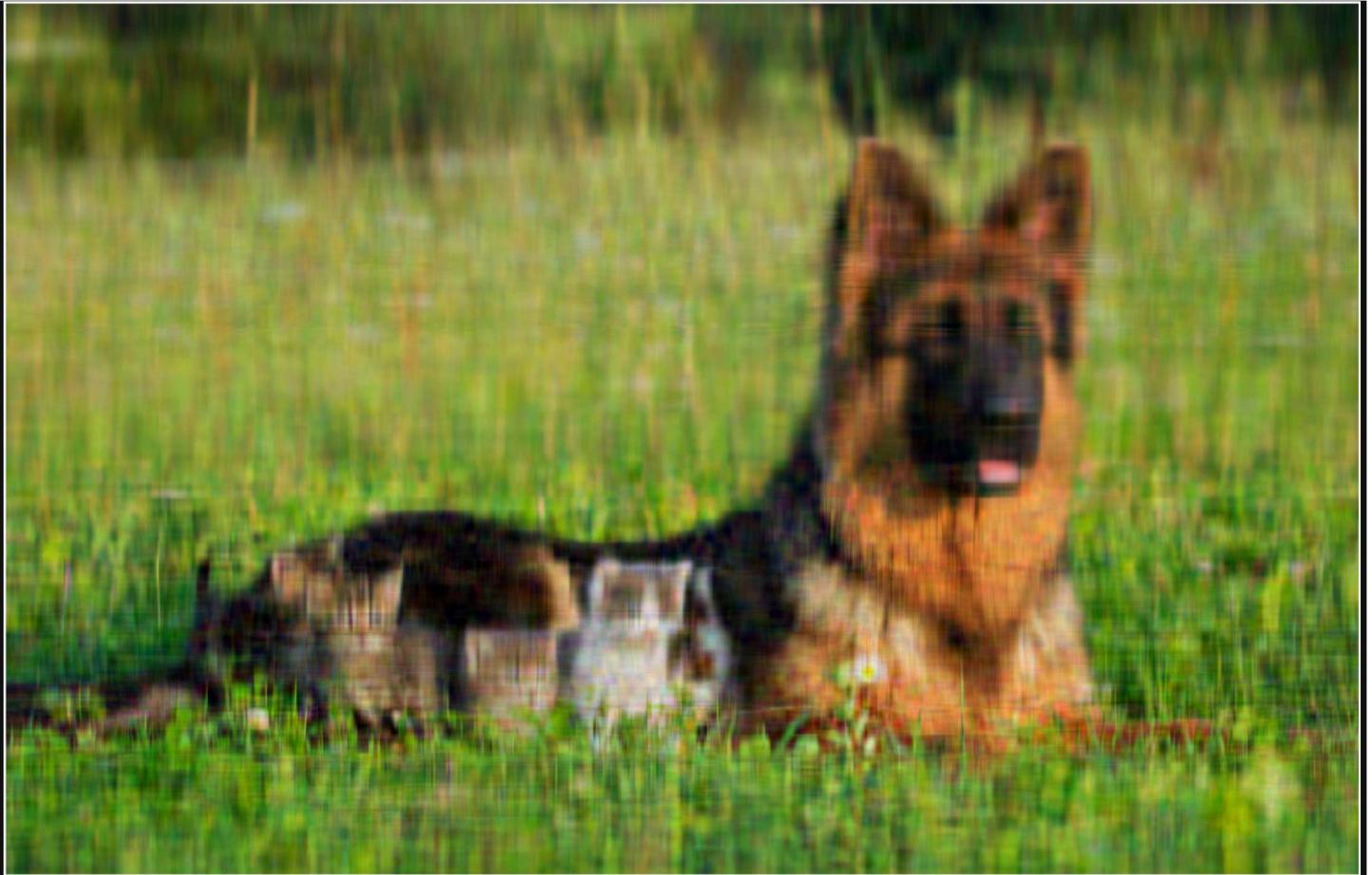
U_Red_s = np.count_nonzero(U_Red[:, :, r])
S_Red_s = np.count_nonzero(S_Red[:, r, :])
VT_Red_s = np.count_nonzero(VT_Red[:, r, :])
U_Green_s = np.count_nonzero(U_Green[:, :, r])
S_Green_s = np.count_nonzero(S_Green[:, r, :])
VT_Green_s = np.count_nonzero(VT_Green[:, r, :])
U_Blue_s = np.count_nonzero(U_Blue[:, :, r])
S_Blue_s = np.count_nonzero(S_Blue[:, r, :])
VT_Blue_s = np.count_nonzero(VT_Blue[:, r, :])
total_arr = U_Red_s+S_Red_s+VT_Red_s+U_Green_s+S_Green_s+VT_Green_s+U_Blue_s+S_Blue_s+VT_Blue_s
print('total of array in image: ' + str(f'{total_arr:,}'))
print('Compression Ration is: ' + str(f'{total_arr/A_sum:.%}'))
```

همانند قیل هر 3 ماتریس S, VT, U را با رنک های برابر با مثال قیل 400 , 25 , 100 , 5 به یک ماتریس تبدیل میکنیم. سپس اعداد آن را به یک مقدار صحیح تبدیل کرده و 3 ماتریس RGB را با هم ادغام میکنیم. سپس آرایه ها را به نسبتی با ماکسیمم 1 و مینیمم 0 را برای بازسازی دوباره تصویر تبدیل میکنیم. با نمایش تصویر و گرفتن خروجی تعداد آرایه ها میتوانیم یک مقایسه خوبی نسبت به تصویر اصلی داشته باشیم.



```
image shape for Red is: [(2417, 5), (5, 5), (5, 3753)]
image shape for Green is: [(2417, 5), (5, 5), (5, 3753)]
image shape for Blue is: [(2417, 5), (5, 5), (5, 3753)]
total of array in image: 92,565
Compression Ration is: 0.341378%
```

با رنک 5 جمع 3 ماتریس RGB برابر 92,565 شده و نسبت فشرده سازی برابر 0.34 %. میبینیم که هماننده تصویر سیاه و سفید تنها شبی از تصویر اصلی قابل مشاهده است.



```
image shape for Red is: [(2417, 25), (25, 25), (25, 3753)]
image shape for Green is: [(2417, 25), (25, 25), (25, 3753)]
image shape for Blue is: [(2417, 25), (25, 25), (25, 3753)]
total of array in image: 462,825
Compression Ration is: 1.706891%
```

با رنک 25 تقریبا تصویر دیده است اما کیفیت قابل قبولی ندارد. نسبت فشرده سازی 1.70%



```
image shape for Red is: [(2417, 100), (100, 100), (100, 3753)]
image shape for Green is: [(2417, 100), (100, 100), (100, 3753)]
image shape for Blue is: [(2417, 100), (100, 100), (100, 3753)]
total of array in image: 1,851,300
Compression Ration is: 6.827563%
```

با افزایش رنک به 100 تقریبا تصویر قابل قبول بوده. نسبت فشرده سازی % 6.82



```
image shape for Red is: [(2417, 400), (400, 400), (400, 3753)]  
image shape for Green is: [(2417, 400), (400, 400), (400, 3753)]  
image shape for Blue is: [(2417, 400), (400, 400), (400, 3753)]  
total of array in image: 7,405,200  
Compression Ration is: 27.310251%
```

با افزایش بیش از اندازه رنک به عدد 400 تصویر تقریبا با تصویر اصلی برابر خواهد بود و توانسیتم تعداد 27,115,093 آرایه را به 7,405,200 کاهش دهیم و نسبت فشرده سازی را 27.31% کنیم.

همچنین در خروجی برنامه میتوان حجم فایل های ذخیره شده رنگ را در کامپیوتر مشاهده کرد که به ترتیب برای تصویر اوجینال رنگی و سپس تصاویر از کیفیت بالا به کیفیت پایین تر برابر است با: 3.66Mb , 1.08Mb , 1.52Mb , 2.02Mb , 2.67Mb . میبینیم که همچنین با کم کردن تعداد آرایه ها در بازسازی تصویر رنگی هم توانستیم که علاوه بر داشتن یک خروجی قابل قبول حجم اطلاعات را هم فشرده کنیم.

نتیجه گیری:
اگرچه که این مقاله بیشتر برای یادگیری فشرده سازی صورت گرفت و هدف آن یادگیری الگوریتم SVD بوده است. اما یکی از استفاده های اساسی این الگوریتم در کاهش داده های با ابعاد بالا در مسائل یادگیری ماشین است.