Imaging Compression Using SVD (Singular Value Decomposition)

Ying Xuan Su
7111095006

March 12, 2023

Introduction

- SVD is a matrix decomposition method that can be used for dimensionality reduction, image compression, and more.
- For an $m \times n$ matrix A, SVD can be expressed as follows:

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

where U is an $m \times m$ unitary matrix, Σ is an $m \times n$ diagonal matrix with singular values on the diagonal, and V is an $n \times n$ unitary matrix.

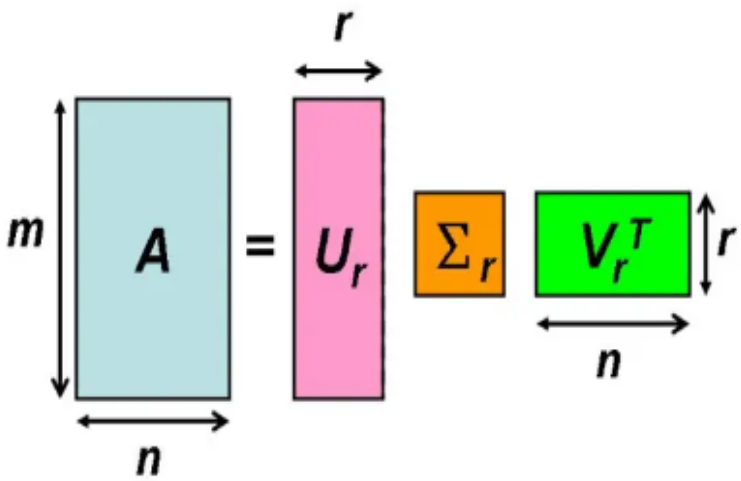

Figure 1: SVD, where rank is r

The original Image I use for this experiment

Figure 2: The original Image

The gray-level image compression with different value of k

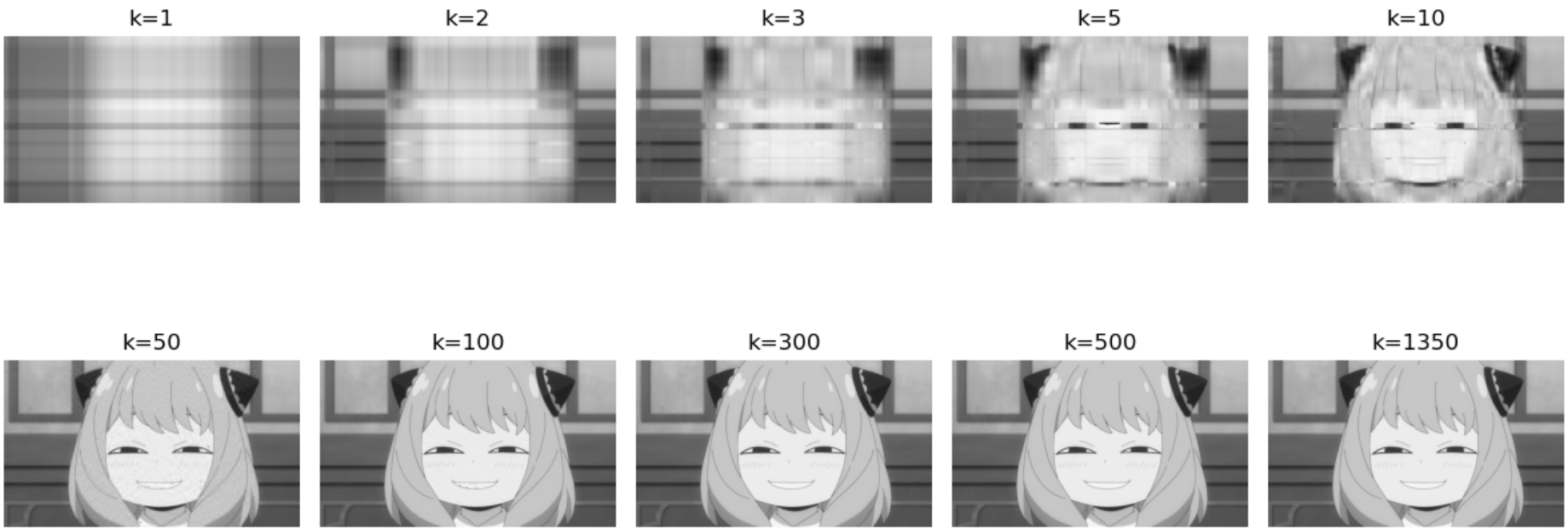

Figure 3: The compressed image using SVD

The gray-level image compression with different value of k

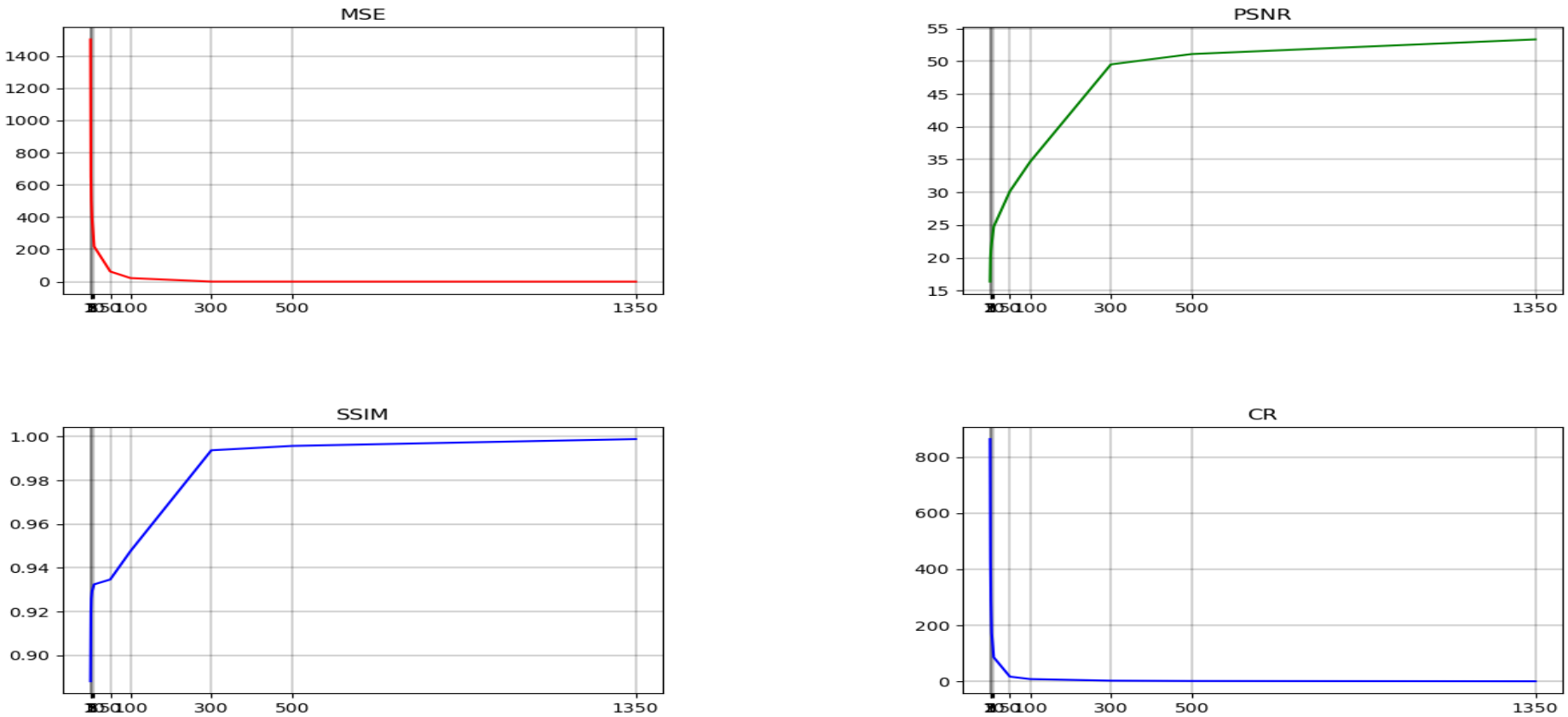

Figure 4: Performance

The gray-level image compression with different value of k

Table 1: Experiment Result

k	MSE	PSNR	SSIM	CR
1	1502.871	16.362	0.888	863.77
2	650.162	20.001	0.92	431.885
3	506.884	21.082	0.926	287.923
5	391.257	22.206	0.929	172.754
10	218.667	24.733	0.932	86.377
50	63.009	30.137	0.935	17.275
100	22.348	34.638	0.948	8.638
300	0.726	49.519	0.994	2.879
500	0.505	51.098	0.996	1.728
1350	0.302	53.332	0.999	0.64

The RGB image compression with different value of k

Figure 5: The compressed image using SVD

The RGB image compression with different value of k

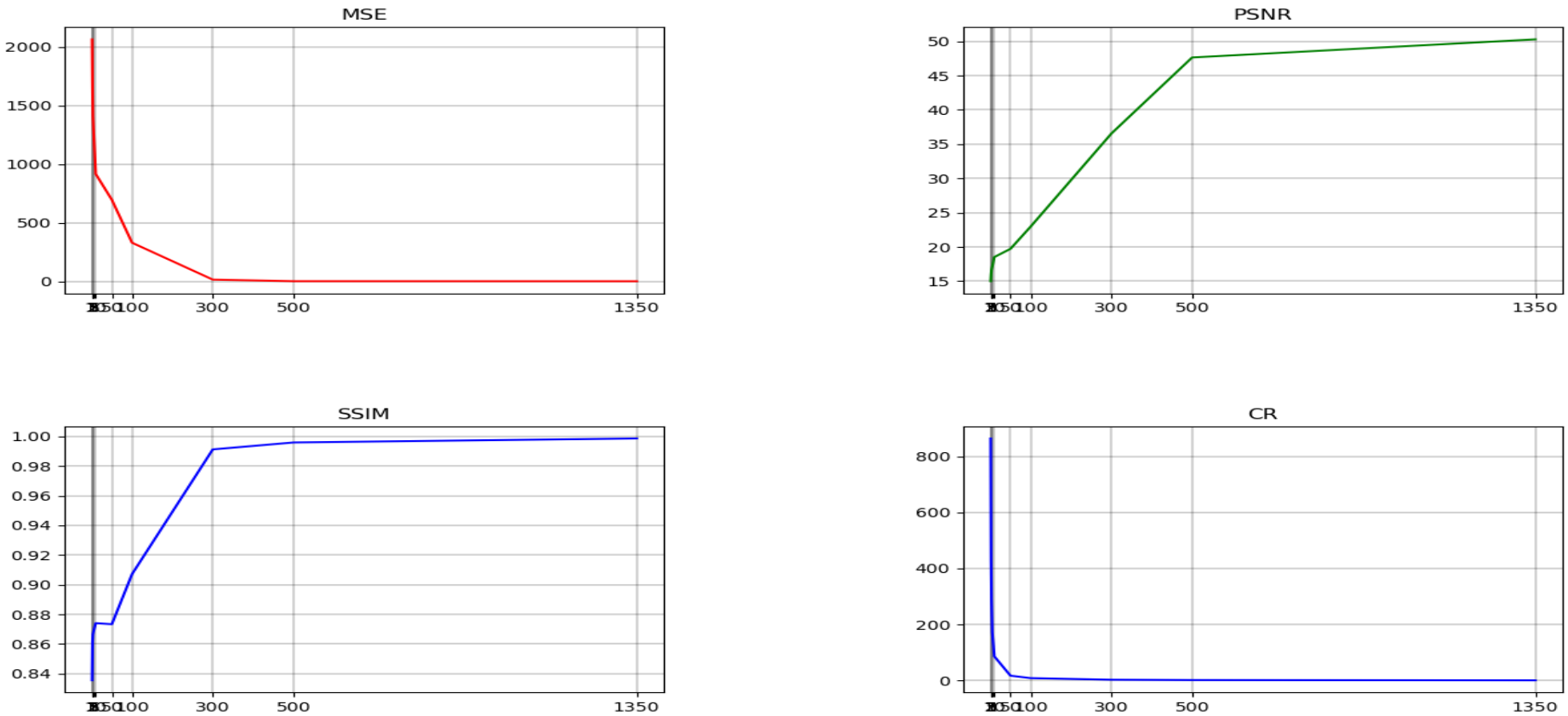

Figure 6: Performance

The RGB image compression with different value of k

Table 2: Experiment Result

k	MSE	PSNR	SSIM	CR
1.000	2064.763	14.982	0.835	863.770
2.000	1651.776	15.951	0.861	431.885
3.000	1412.896	16.630	0.866	287.923
5.000	1270.564	17.091	0.869	172.754
10.000	916.246	18.511	0.874	86.377
50.000	695.415	19.708	0.873	17.275
100.000	329.302	22.955	0.907	8.638
300.000	14.420	36.541	0.991	2.879
500.000	1.117	47.650	0.996	1.728
1350.000	0.608	50.295	0.999	0.640

More images

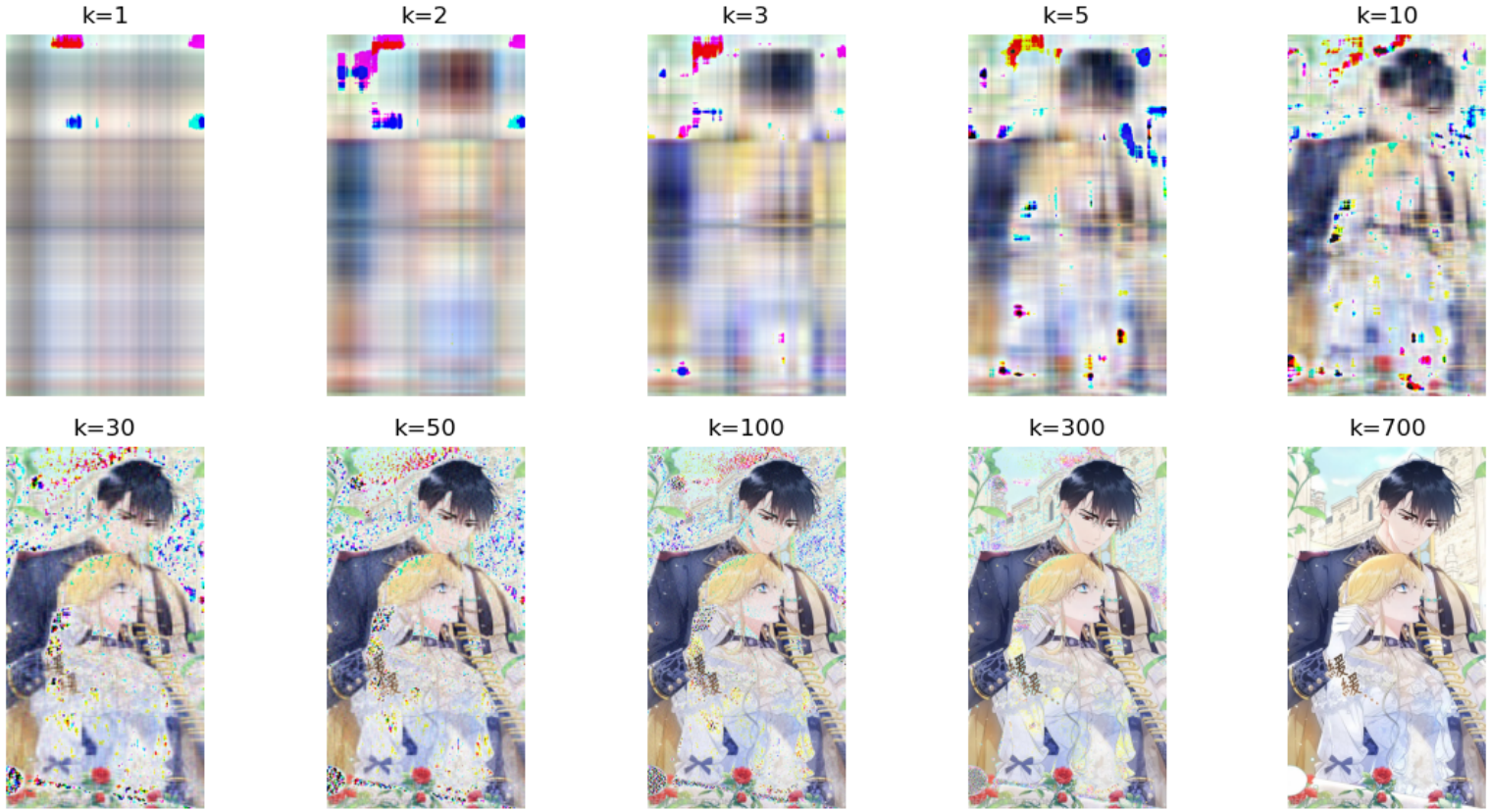

Figure 7: The compressed image using SVD

More images

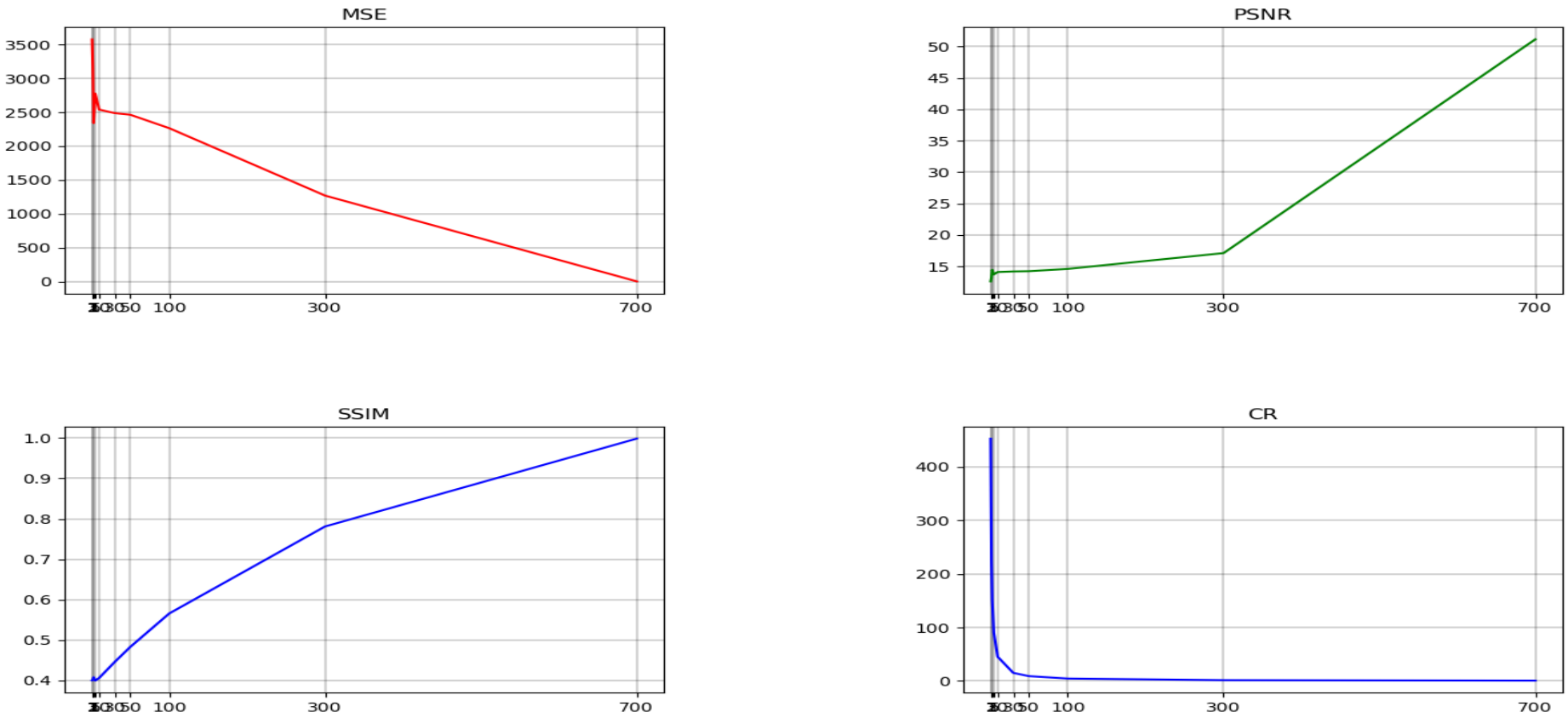

Figure 8: Performance

More images

Table 3: Experiment Result

k	MSE	PSNR	SSIM	CR
1	3578.48	12.594	0.4	452.714
2	3139.019	13.163	0.402	226.357
3	2340.613	14.438	0.408	150.905
5	2777.577	13.694	0.4	90.543
10	2540.669	14.081	0.406	45.271
30	2488.134	14.172	0.446	15.09
50	2464.304	14.214	0.483	9.054
100	2265.461	14.579	0.566	4.527
300	1268.919	17.096	0.781	1.509
700	0.5	51.142	0.998	0.647

More images

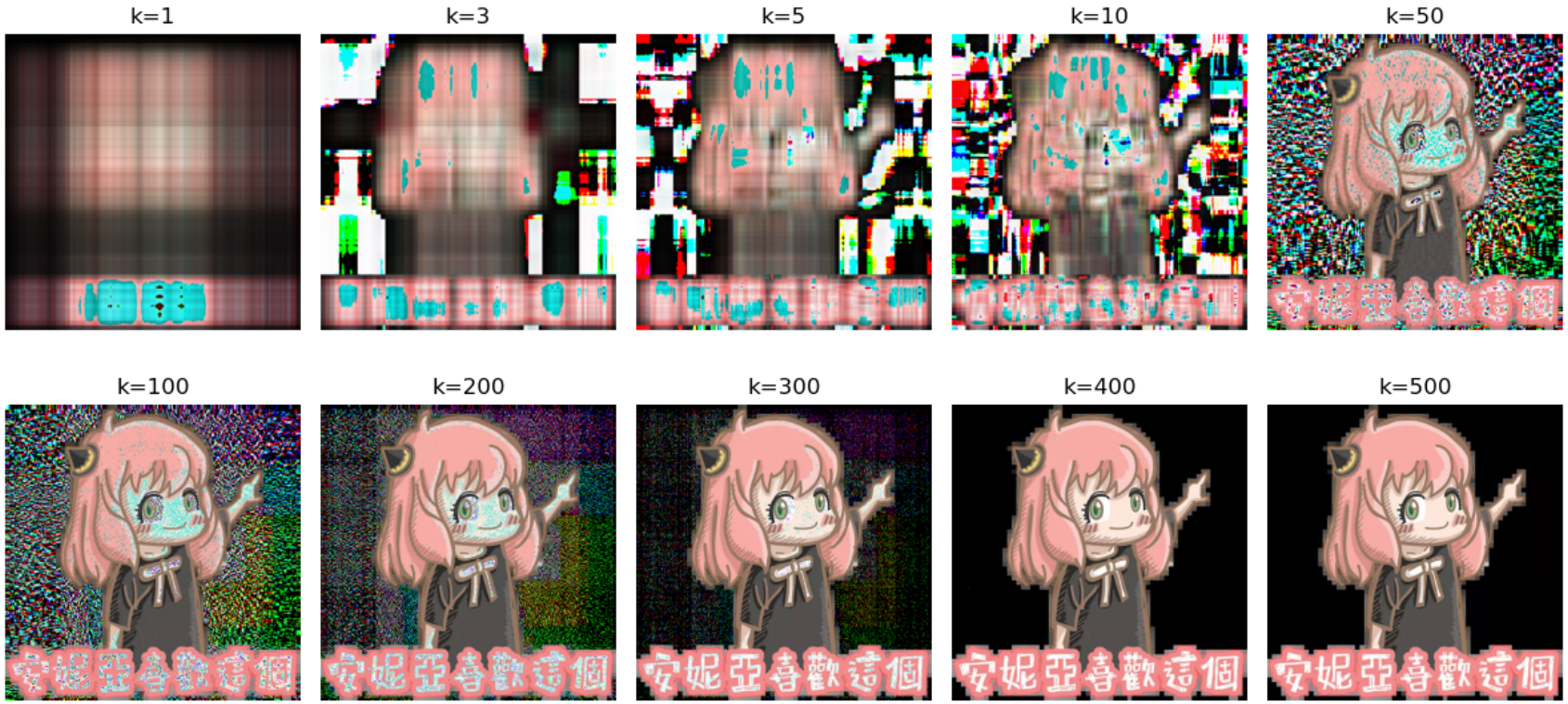

Figure 9: The compressed image using SVD

More images

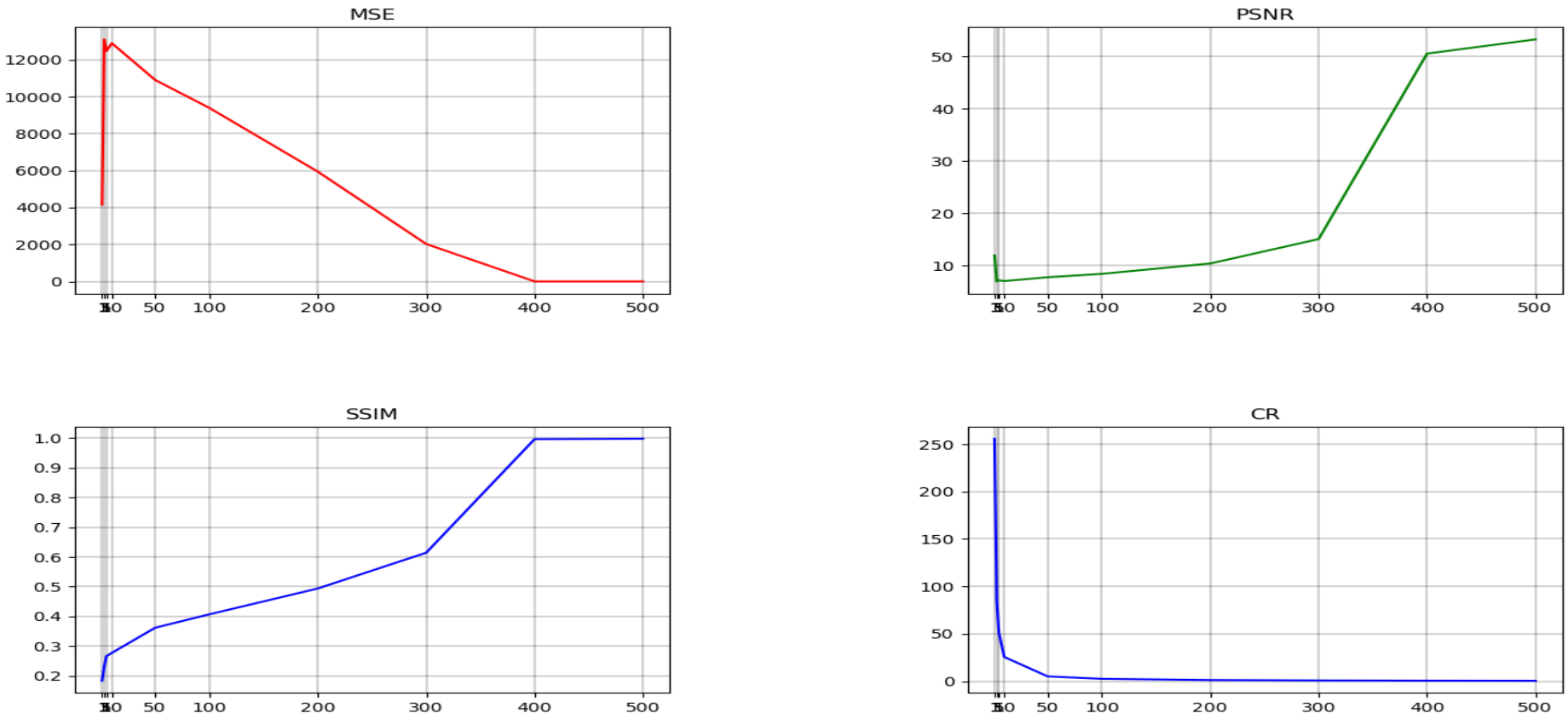

Figure 10: Performance

More images

Table 4: Experiment Result

k	MSE	PSNR	SSIM	CR
1	4152.984	11.947	0.183	255.75
2	10991.028	7.72	0.225	127.875
3	13107.346	6.956	0.231	85.25
5	12471.986	7.171	0.266	51.15
10	12900.396	7.025	0.277	25.575
30	12062.761	7.316	0.336	8.525
50	10907.907	7.753	0.361	5.115
100	9394.405	8.402	0.407	2.558
300	2027.355	15.062	0.614	0.853
500	0.303	53.32	0.998	0.512

More images

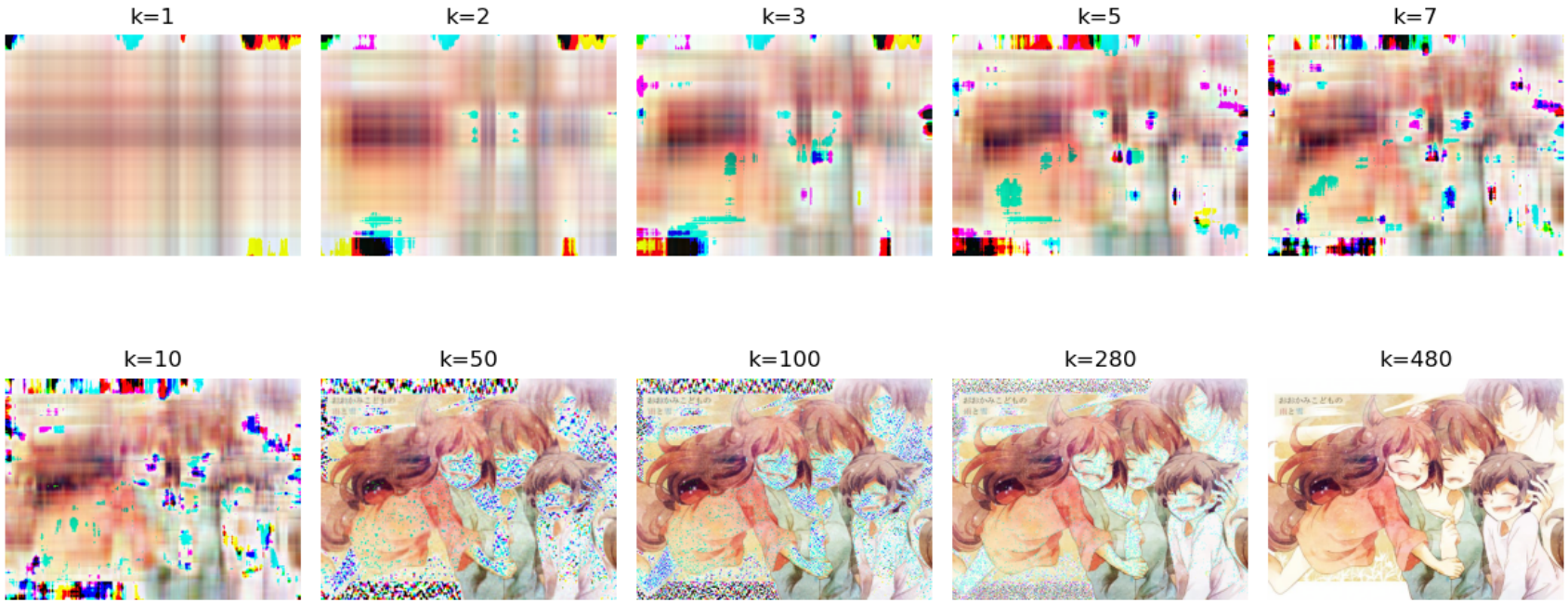

Figure 11: The compressed image using SVD

More images

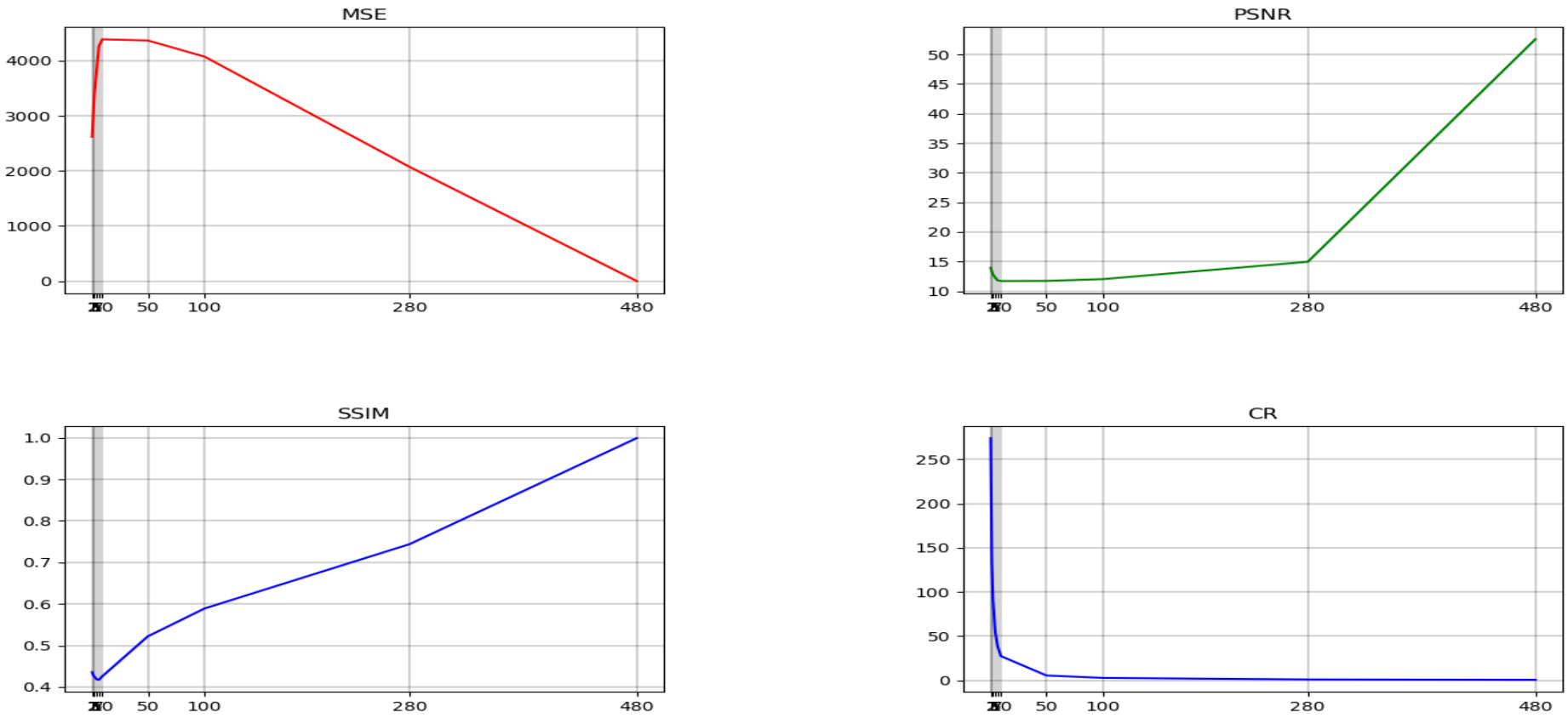

Figure 12: Performance

More images

Table 5: Experiment Result

k	MSE	PSNR	SSIM	CR
1	2619.806	13.948	0.436	274.041
2	2989.401	13.375	0.429	137.021
3	3398.88	12.817	0.425	91.347
5	3832.113	12.296	0.418	54.808
7	4263.518	11.833	0.417	39.149
10	4389.043	11.707	0.426	27.404
50	4368.499	11.727	0.522	5.481
100	4074.188	12.03	0.589	2.74
280	2075.732	14.959	0.744	0.979
480	0.359	52.576	0.999	0.571

Code

```
def SVD(img, k):
    compresed_img = np.zeros_like(img)
    for ch in range(3):
        U, Sigma, V = np.linalg.svd(img[:,:,ch])
        compresed_img[:,:,ch] = U[:,:k].dot(np.diag(Sigma[:k])) \
                                                .dot(V[:k,:])
    return compresed_img
```

Click here to get the while code.

Dicussion

- According to Figure 10,12, MSE are not always strictly increasing or decreasing, but I don't know what it the reason for it.
- PSNR does not increase faster as it is smaller, I think it is relative to how much details in the original image, but I'm not very sure.

Other application for SVD - watermark

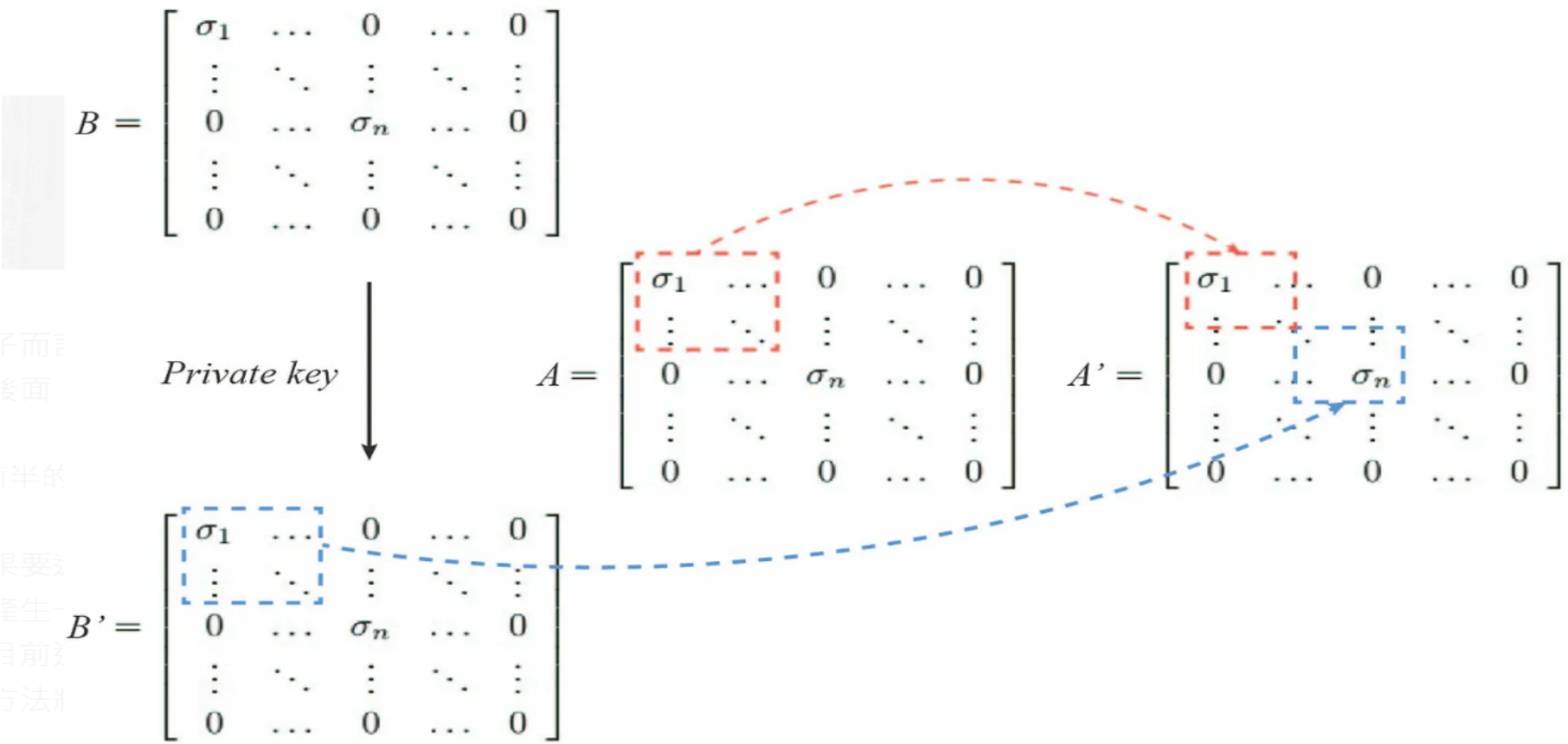

Figure 13: SVD for watermark

Other application for SVD - watermark

The watermarked image is similar to the original image.

Figure 14: original image

Figure 15: watermarked image

Other application for SVD - watermark

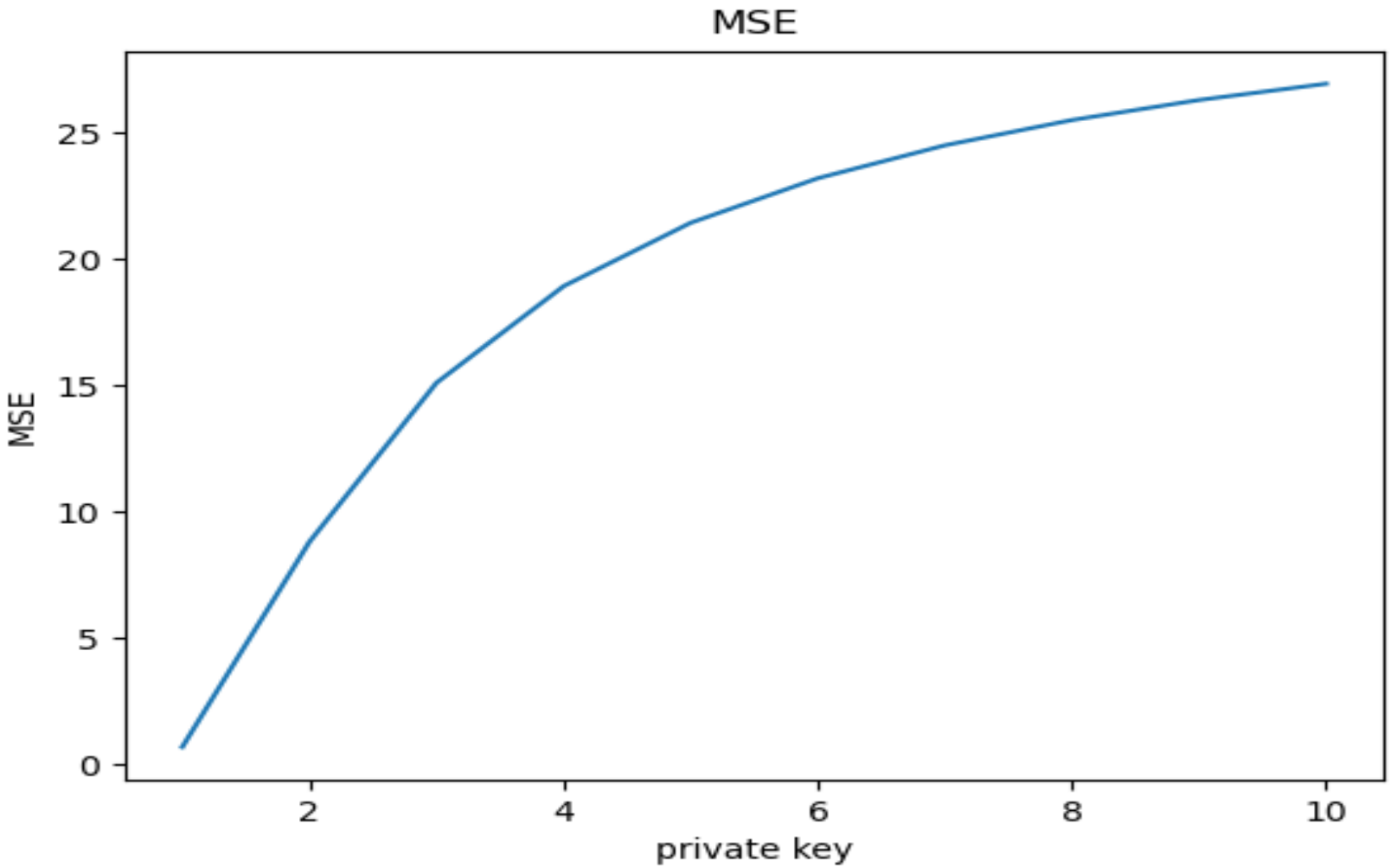

Figure 16: Performance

Code

```
private_key = 5

U1, Sigma1, V1 = np.linalg.svd(img1)
U2, Sigma2, V2 = np.linalg.svd(img2)
len1, len2 = len(Sigma1), len(Sigma2)  # chack they are the same.

Sigma = np.concatenate((Sigma1[:round(len1/2)], Sigma2[round(len1
    /2):]/private_key), axis=None)
#Sigma = Sigma1 + private_key * Sigma2

Sigma_diag = np.zeros_like(np.diag(Sigma1))
for i in range(len(Sigma)):
    Sigma_diag[i][i] = Sigma[i]
result_img = U1[:,:len(Sigma)].dot(Sigma_diag).dot(V1[:len(Sigma)
    ,:])
MSE_list.append(mean_squared_error(img1, result_img))
cv2.imwrite("watermarked.png", result_img)
```

Reference

https://reurl.cc/eXdEWK
https://reurl.cc/OVjp39

Thank You!