

Math 521 HW2

Raj Mohanty

raj.mohanty@student.csulb.edu

2 Computing

2.1 Problem 1

We construct the matrices as per code snippet below.

```
P1 = np.array( [[ 1, 1, 1, 1],  
                [ 1, 0, 0, 1],  
                [ 1, 0, 0, 1],  
                [ 1, 0, 0, 1],  
                [ 1, 1, 1, 1],] )
```

The above code snippet is for the first pattern P1.

We plot the 3 training patterns using matplotlib's imshow function in figure 1.1.

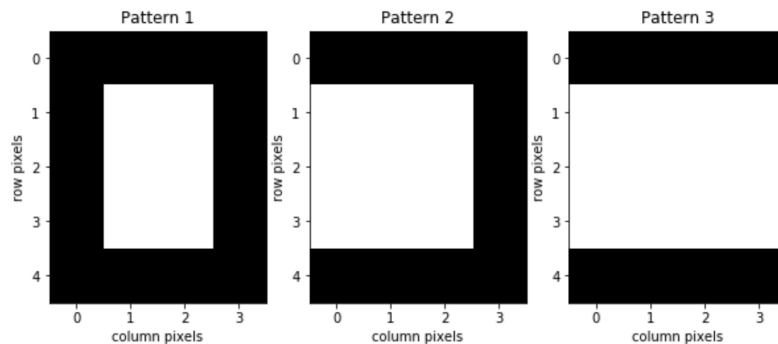


Figure 1.1 : The 3 training patterns

The new pattern Pnew is shown in figure 1.2

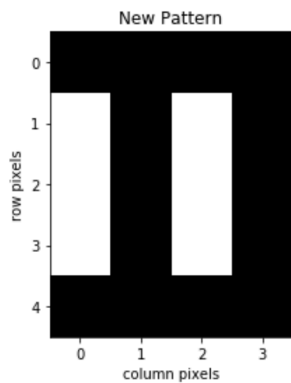


Figure 1.2: New pattern

We then flatten the 3 training pattern matrices as follows and stack them in the columns of matrix V (shown in figure 1.3).

```
P1_f=P1.flatten()
P2_f=P2.flatten()
P3_f=P3.flatten()
```

Code snippet for flattening matrices

```
V
array([[1, 1, 1],
       [1, 1, 1],
       [1, 1, 1],
       [1, 1, 1],
       [1, 0, 0],
       [0, 0, 0],
       [0, 0, 0],
       [1, 1, 0],
       [1, 0, 0],
       [0, 0, 0],
       [0, 0, 0],
       [1, 1, 0],
       [1, 0, 0],
       [0, 0, 0],
       [0, 0, 0],
       [1, 1, 0],
       [1, 1, 1],
       [1, 1, 1],
       [1, 1, 1],
       [1, 1, 1]])
```

Figure 1.3 : Flattened and combined pattern in matrix V

We then determine the orthonormal bases of V and call it M as follows:

```
M=spy.linalg.orth(V)
```

We calculate the projection matrix P where

$$P = MM^T$$

We flatten the Pnew pattern as follows:

```
P_new_f=P_new.flatten()
```

$$Proj_V P_{new} = PP_{new_f}$$

Code for the above equation is

```
proj_P_new = np.matmul(P,P_new_f)
```

Novelty pattern = $P_{new_f} - Proj_V P_{new}$

Code :

```
Novelty_pattern=np.subtract(P_new_f,proj_P_new).reshape(5,4)
```

Finally we plot the Novelty pattern in figure 1.4

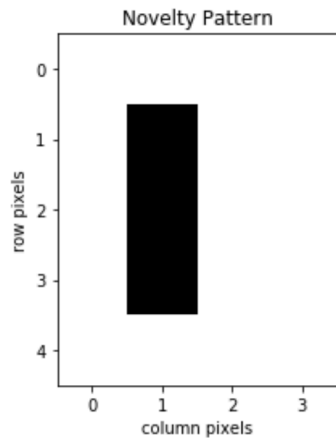


Figure 1.4: The derived novelty pattern in the new pattern

Here we notice that yes, the pattern shown in figure 4 was missing in the 3 training pattern.

2.2 Problem 2

We create matrix A as follows

```
A = np.array( [[ 1, 1, 1, 1],
                 [ 0, 1, 0, 1],
                 [ 0, 1, 0, 1],
                 [ 0, 1, 0, 1],
                 [ 1, 1, 1, 1], ] )
```

We display the pattern as shown in figure 2.1.

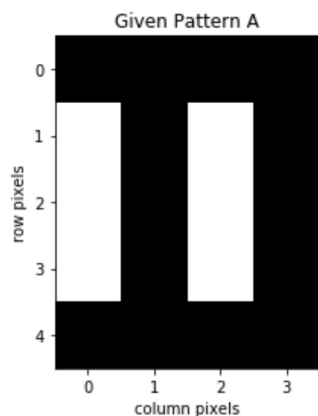


Figure 2.1: Pattern of matrix A

We then go ahead and determine the SVD as follows

```
U,S,V=np.linalg.svd(A)
```

We plot the reconstruction of A_1, A_2, A_3 and A_4 as shown in figure 2.2

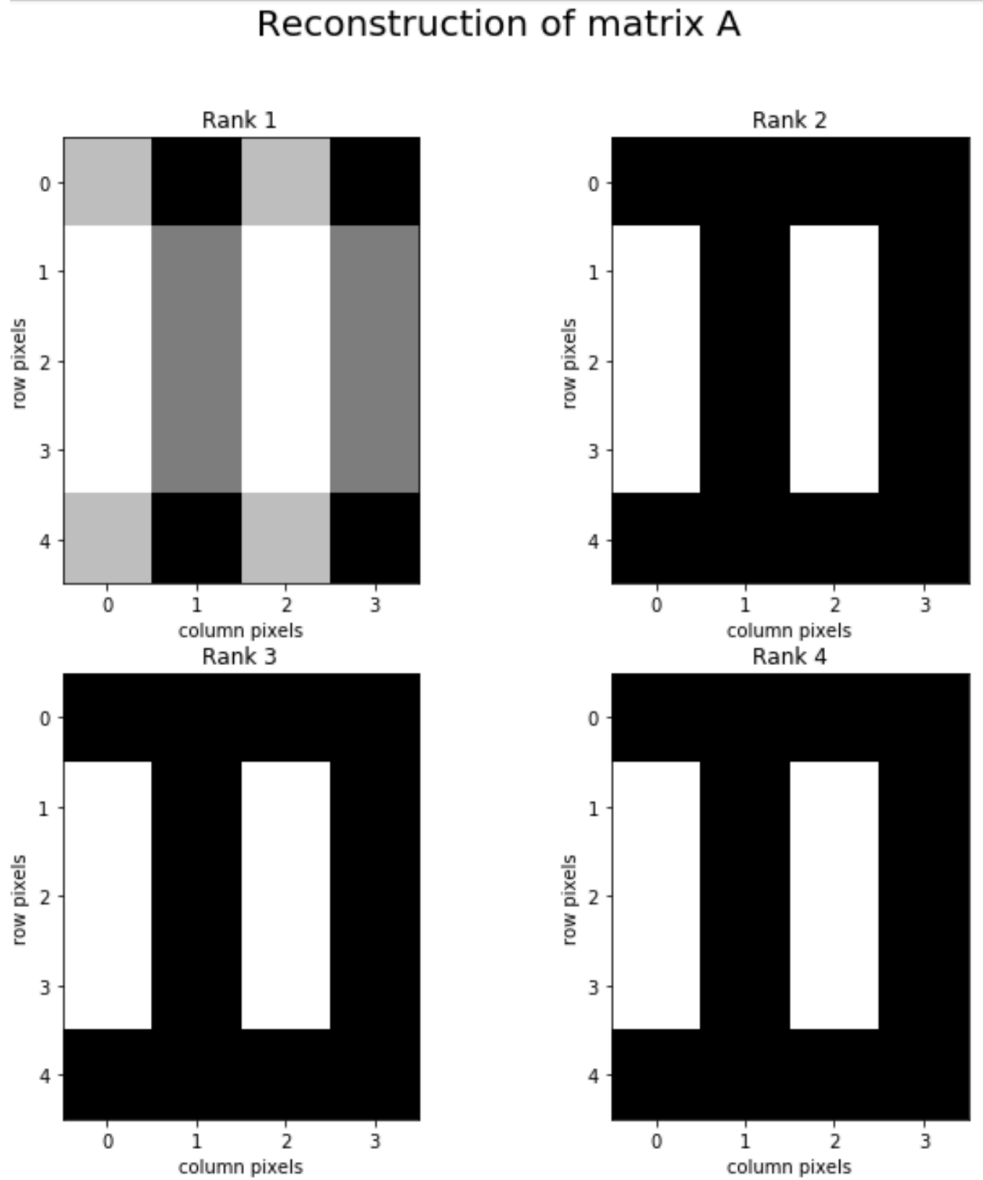


Figure 2.2: The 4 reconstruction of matrix A for rank 1,2,3 and 4 from the SVD

We notice that from rank 2 onwards the matrix A is fully reconstructed. Hence the first 2 principal components are enough to represent the variability of the data in the columns of A.

2.3 Problem 3

In this exercise we use a (384x384) greyscale image of a buffalo as shown in figure 3.1.

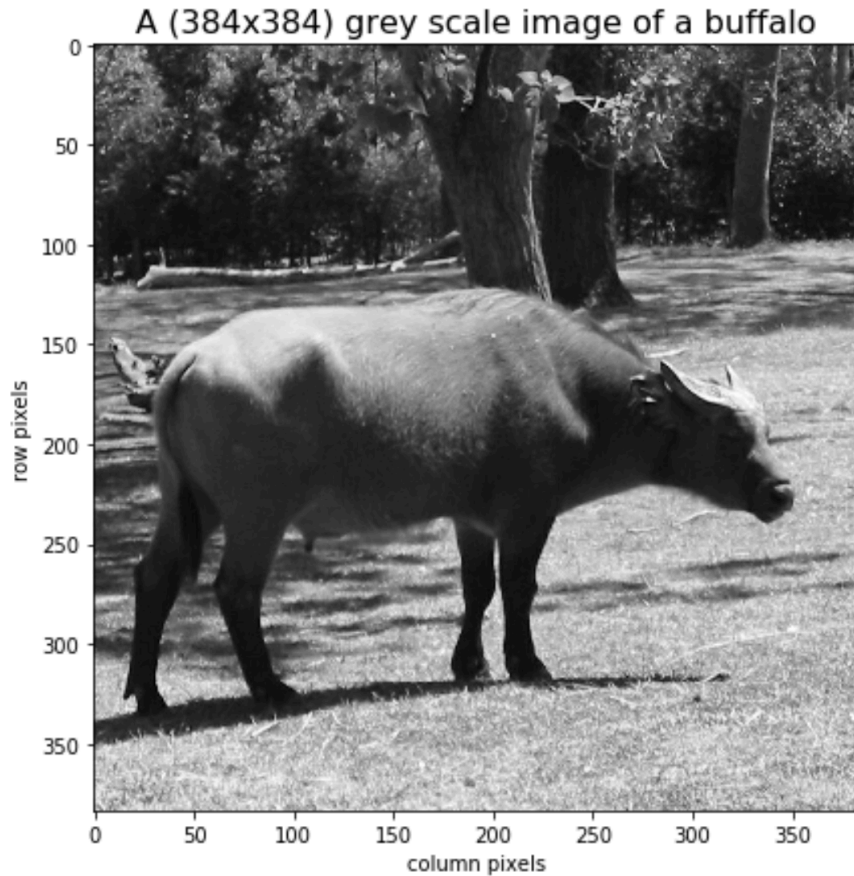


Figure 3.1 : The chosen image (matrix A)

We then convert the uint8 pixel values to float64 (or double) precision and determine the SVD as follows:

```
A=A.astype(np.float64)
```

```
U,S,V=np.linalg.svd(A)
```

Figure 3.2 shows the singular value distribution of matrix A

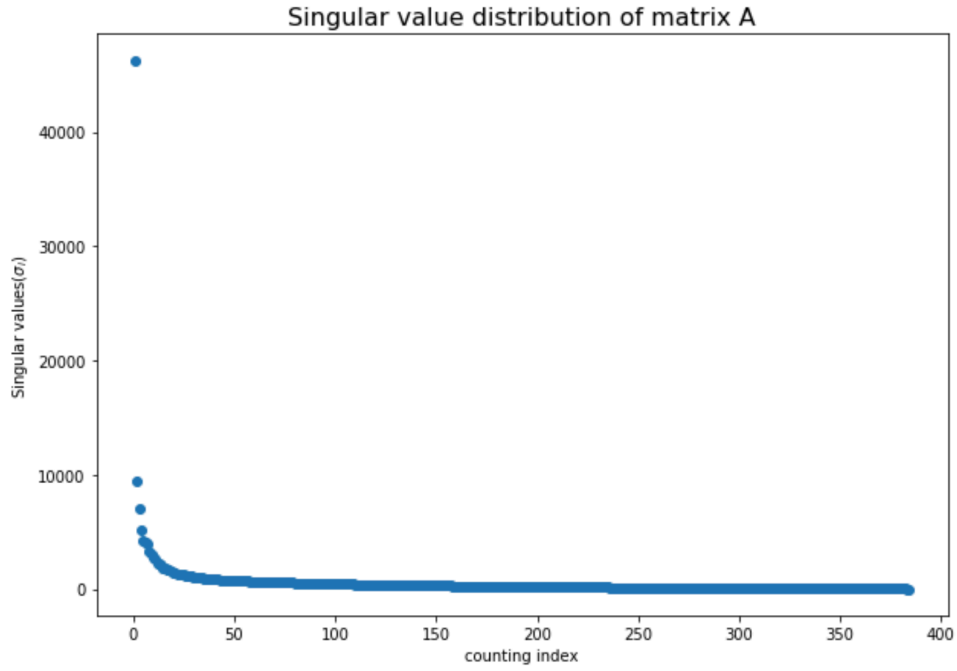


Figure 3.2: Singular value distribution of A

We then calculate the energy E_k as follows:

```
energy_cal=pd.DataFrame(list(zip(x, singular_values)),
                        columns=['k', 'singular values'])

energy_cal['sigma_sq']=energy_cal['singular values']**2

energy_cal['sigma_sq_cusum']=energy_cal.sigma_sq.cumsum()

energy_cal['E_k']=energy_cal['sigma_sq_cusum']/energy_cal.sigma_sq.sum()
```

We notice that we reach 95% of energy at rank $k = 7$ as shown in figure

3.3

```

: energy_cal

```

	k	singular values	sigma_sq	sigma_sq_cusum	E_k
0	1	46281.329948	2.141962e+09	2.141962e+09	0.865386
1	2	9386.268303	8.810203e+07	2.230064e+09	0.900981
2	3	6968.066915	4.855396e+07	2.278617e+09	0.920598
3	4	5201.601380	2.705666e+07	2.305674e+09	0.931529
4	5	4252.804230	1.808634e+07	2.323760e+09	0.938836
5	6	4112.323483	1.691120e+07	2.340672e+09	0.945669
6	7	3931.462750	1.545640e+07	2.356128e+09	0.951913




Figure 3.3: Level of energy at different rank k values

The relative error is calculated as follows

```
relative_error = round(S[k]/S[0],4)
```

Finally we show the rank-10, rank-50, rank-100 and rank-200 approximation of image A in figure 3.4.

Rank Approximations

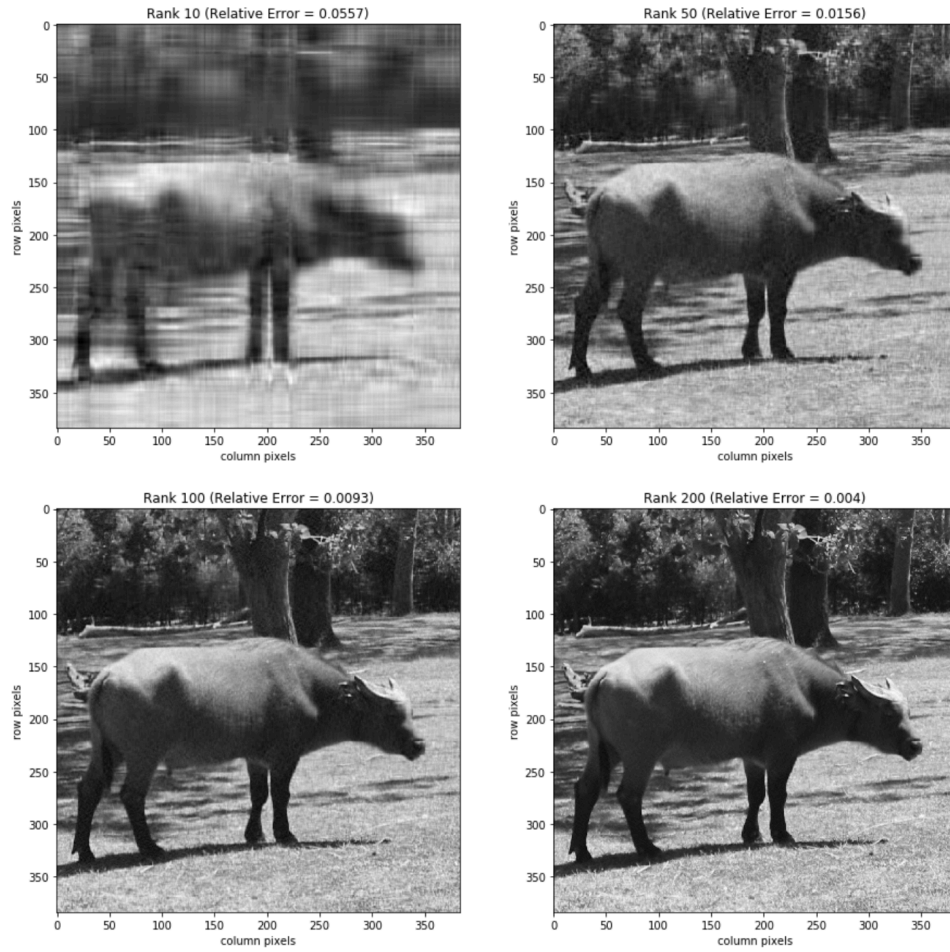


Figure 3.4 : Various rank approximation of the original image

We notice that we almost reconstruct the original image from at rank-50.