SVD FOR IMAGE CLASSIFICATION

64170012 - Tuğçe Keskin

64170038 - Necibe Nursena Aksu

Problem Statement and Background

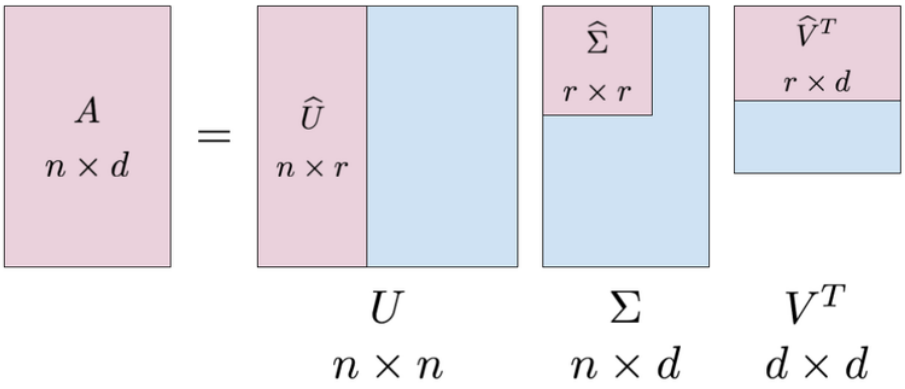
The purpose of the question is to use our knowledge about matrices on the real-life problem using SVD (singular value decomposition).

Matrices was factorized into U, S, and V. The aim of the singular value decomposition is to reduce the size of the matrices and factorize them into more informative versions.

Suppose that;

Matrix A is (n, d).

Factorize it to U (n, n), S (n, d), and V(d, d) → Same matrix (A) is obtained.

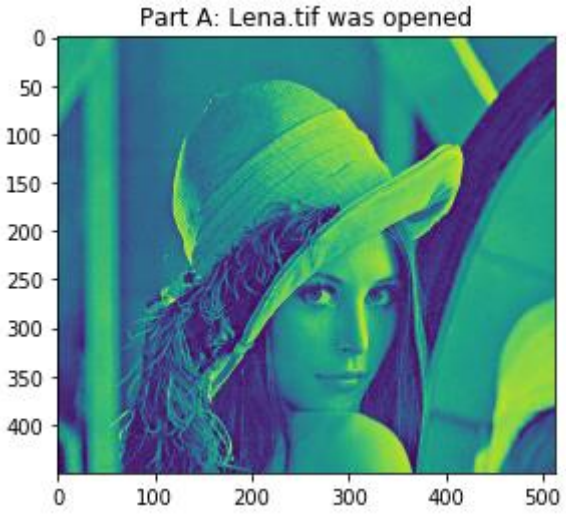


It is tried to factorize A into a more reduced form.

After factorizing A to U, S, and V → U (n, r), S (r,r), V(r, d).

When sizes of matrices are reduced → They lose some information, but reduced matrices make our problem easier.

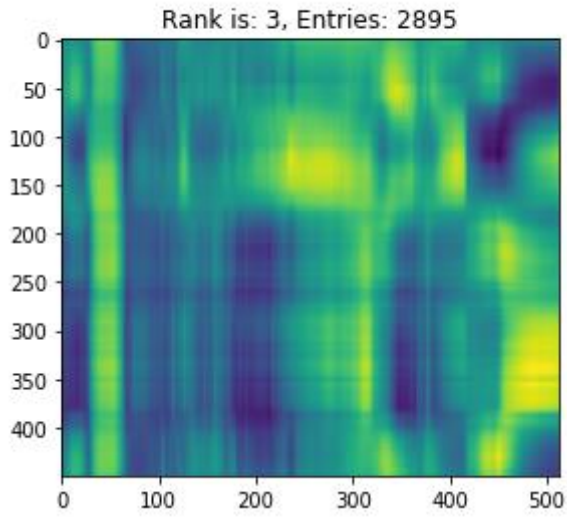
Results

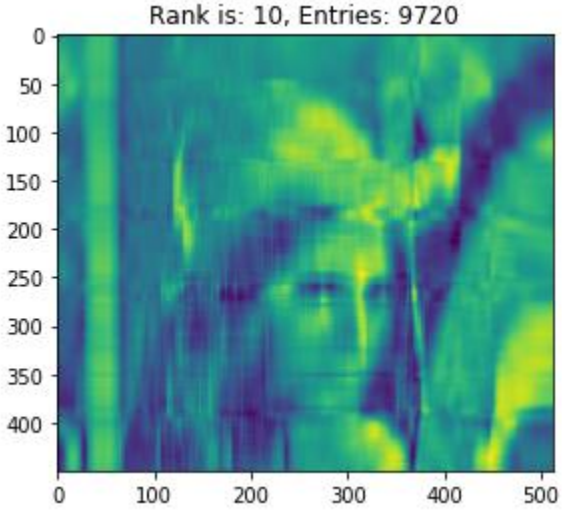


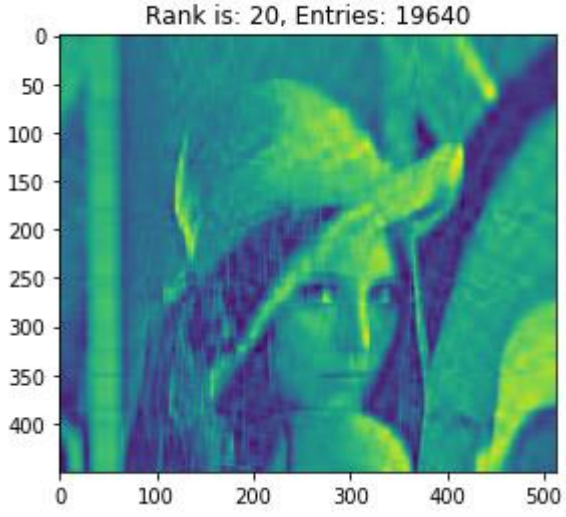
Part B: Total number entries that we need: 230400

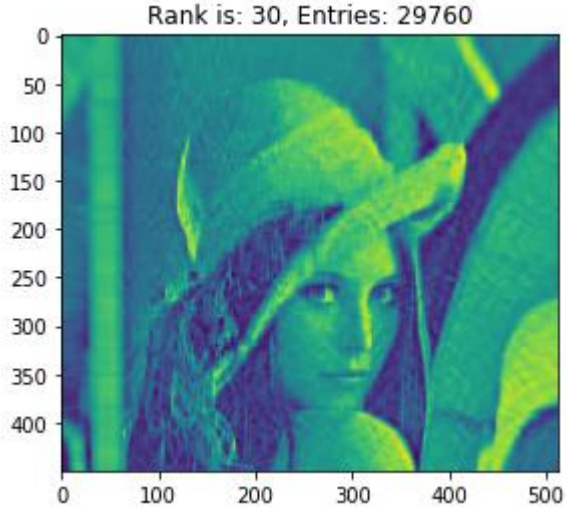
Part C:
The shape of U: (450, 450)
The shape of S: (450, 450)
The shape of V: (450, 512)

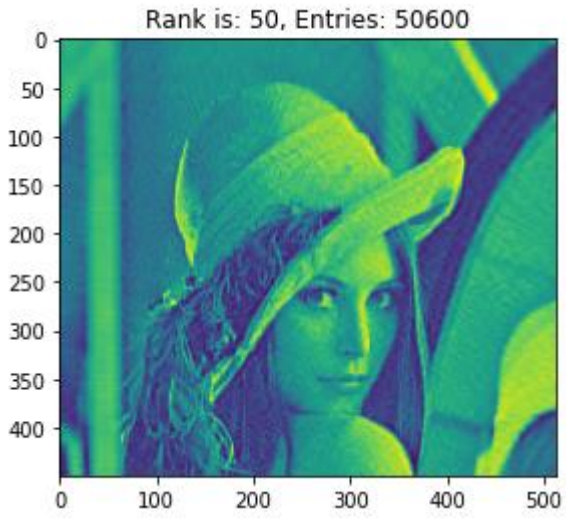
Part D and E:

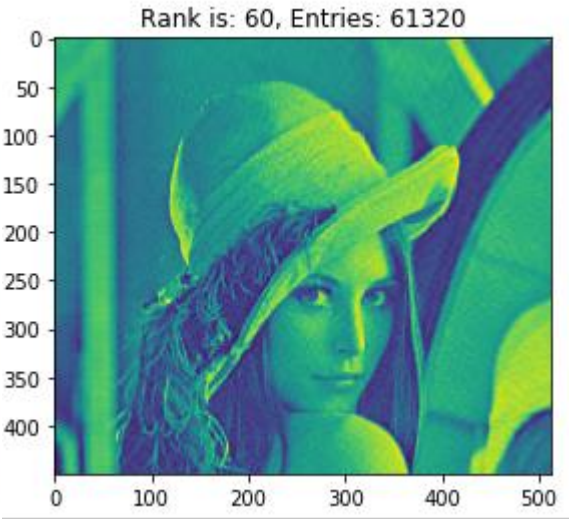


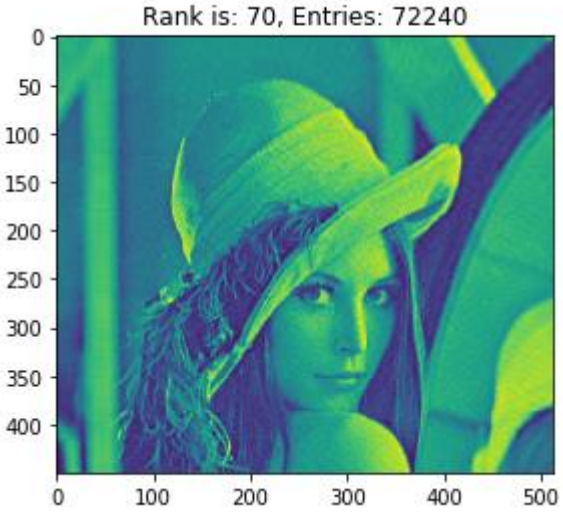


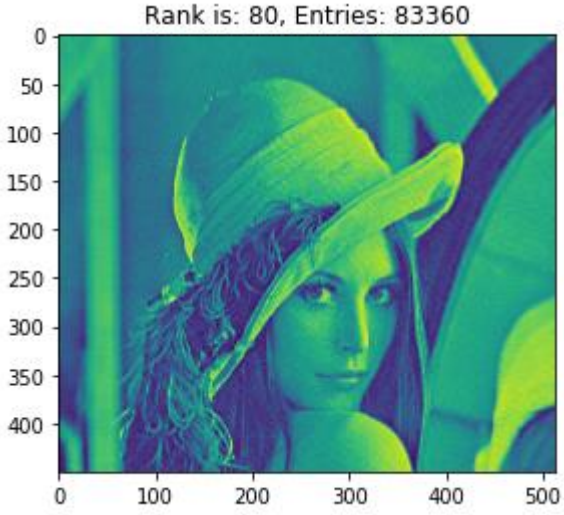


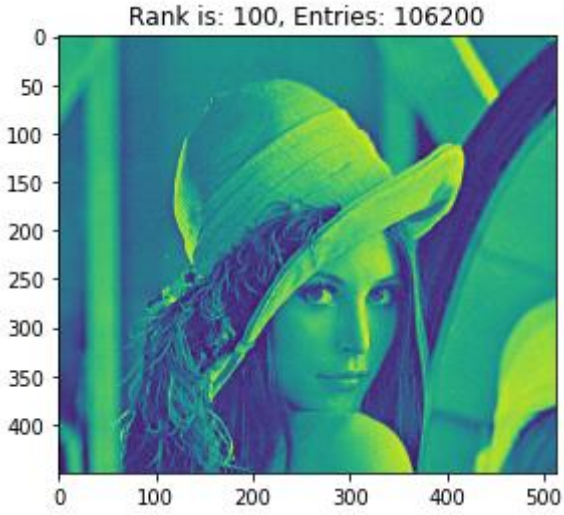



Rank is: 50, Entries: 50600
0
50
100
150
200
250
300
350
400
0
100
200
300
400
500


Rank is: 60, Entries: 61320
0
50
100
150
200
250
300
350
400
0
100
200
300
400
500


Rank is: 70, Entries: 72240
0
50
100
150
200
250
300
350
400
0
100
200
300
400
500


Rank is: 80, Entries: 83360
0
50
100
150
200
250
300
350
400
0
100
200
300
400
500


Rank is: 100, Entries: 106200
0
50
100
150
200
250
300
350
400
0
100
200
300
400
500

Conclusion

```
import numpy as np
import matplotlib.pylab as plt
Lena = plt.imread('Lena.tif')

#part A
plt.title('Part A: Lena.tif was opened')
plt.imshow(Lena)
plt.show()


#part B
print("\nPart B: Total number entries that we need: " +str(Lena.shape[0]*Lena.shape[1]))

#part C
U, S, V = np.linalg.svd(Lena,full_matrices = False)
S = np.diag(S)
print('\nPart C:')
print('The shape of U: '+str(U.shape))
print('The shape of S: '+str(S.shape))
print('The shape of V: '+str(V.shape))

#part D & E
print('\nPart D and E:')
LenaReproduce = np.zeros((450,512))
ranks = [3,10,20,30,50,60,70,80,100]
MSE = []

for r in ranks:
    LenaReproduce = np.matmul(S[0:r,0:r], V[0:r,0:512])
    LenaReproduce = np.matmul(U[0:450,0:r], LenaReproduce)

    MSE.append(np.sum(np.sqrt((Lena - LenaReproduce)**2)))
    NumberofEntries = LenaReproduce.shape[0]*r + \
                      r*r + r*LenaReproduce.shape[1]
    plt.title('Rank is: '+str(r)+', Entries: '+str(NumberofEntries))
    plt.imshow(LenaReproduce)
    plt.show()

plt.title('The Erros Values For Each Rank')
plt.xlabel('Number of Ranks')
plt.ylabel('The Errors')
plt.plot(ranks,MSE)
plt.show()
```

About the code:

In the Part A, Lena.tif was opened.

In the Part B, the total number of pixels in the image were printed as 230400.

In the Part C, U, S and V matrices were found as a result of SVD.

In the Part D, rank-approximations were calculated through SVD.

The rank is a linear independent columns or rows of an m × n matrix A. The point is that for a rank one matrix, there are lots of redundancy. For a low rank matrix with missing values, we can take advantage of this redundancy to fill in missing entries. As the number of ranks increased, so did the number of pixels. This improves the image quality.

In the Part E, mean square error and how many entries need to be stored were calculated for each rank approximation.

Part F: 30 rank was enough for us. 29760 entries were stored.

Part G: No. Adequate resolution has already been reached. 100 rank was enough for that.

Python 3.7 version was used for the problem solution. Some libraries such as matplotlib.pyplot (math library), numpy (linear algebra library) were used when writing the code.