Project2 Report

Xueying Sui 1001682442

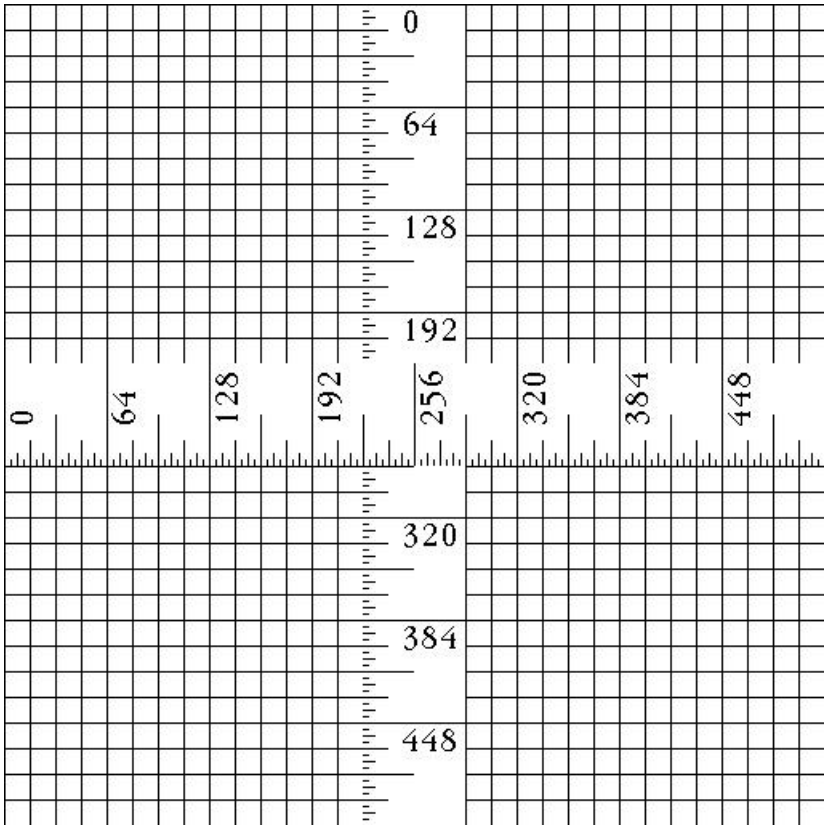
Problem

We need to use SVD(Singular Value Decomposition) find the best low rank approximation matrix. In other words, first we consider the tiff file as a matrix, use the SVD method to find the low rank approximation matrix of different ranks, and then view these matrices as images. Take the rank-5 approximation as an example, we know that $A = U\Sigma V^T$, suppose A is a matrix of m x n, then U is a matrix of m x m, Σ is a matrix of m x n, V is a matrix of n x n. If we want to find the rank-5 approximation, we only need to pick 5 largest singular values and their corresponding eigenvectors. Now U is a matrix of m x 5, Σ is a matrix of 5 x 5, V is a matrix of 5 x n. When we multiply the U, Σ, V^T matrices, the resulting A1 is still a matrix of m x n. Then A1 is the best rank-5 approximation matrix to the A.
Now we need to use SVD find the best rank-5, rank-10, rank-20, rank-50, rank-100 and rank-200 approximation matrices to the matrix (ruler.512.tiff).

Data

We download ruler.512.tiff from http://sipi.usc.edu/database.php?volume=misc

ruler.512.tiff



This is a grayscale image with values between 0 and 255.The result of reading the tiff file as a matrix is:

```
[[  0   0   0 ...   0   0   0]
 [  0 255 255 ... 255 255 255]
 [  0 255 255 ... 255 255 255]
 ...
 [  0 255 255 ... 255 255 255]
 [  0 255 255 ... 255 255 255]
 [  0 255 255 ... 255 255 255]]
```

Method(SVD)

We call the matrix shown above as A, we can represent A as:

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

$$\left(U^T U = I, V^T V = I\right)$$

To calculate U, Σ, V^T, we first calculate AA^T and $A^T A$.

$$AA^T = \left(U\Sigma V^T\right)\left(U\Sigma V^T\right)^T = U\Sigma V^T V\Sigma^T U^T = U\Sigma\Sigma^T U^T = U\Sigma^2 U^T$$

$$A^T A = \left(U\Sigma V^T\right)^T\left(U\Sigma V^T\right) = V\Sigma^T U^T U\Sigma V^T = V\Sigma^T\Sigma V^T = V\Sigma^2 V^T$$

Now we know that U is a matrix composed of eigenvectors of the matrix AA^T, Σ is the diagonal matrix composed of the square root of the eigenvalues of the matrix AA^T, V is a matrix composed of eigenvectors of the matrix $A^T A$.

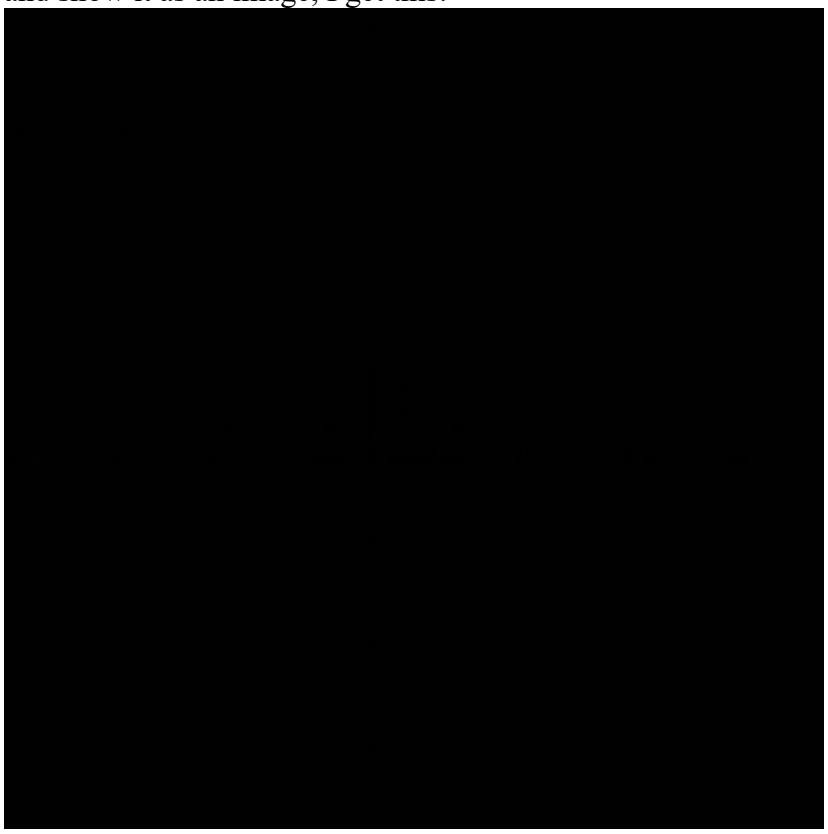
As the description of 'Problem' part, we need to pick k largest singular values and their corresponding eigenvectors to calculate rank-k approximation.

(1) Initial attempt

As mentioned above. In the implementation part, I first calculated AA^T, $A^T A$ and their eigenvalues, eigenvectors. Like this:

```
AAT = np.dot(img, img.transpose())
ATA = np.dot(img.transpose(), img)
values1, vectors1 = np.linalg.eig(AAT)
values2, vectors2 = np.linalg.eig(ATA)
```

Then I calculate the U, Σ and V, when I want to get rank-200 approximation matrix and show it as an image, I get this:



And the result of rank-200 approximation matrix is:

```
[[0.         +0.00000000e+00j 0.         +0.00000000e+00j
  0.         +0.00000000e+00j ... 0.         +0.00000000e+00j
  0.         +0.00000000e+00j 0.         +0.00000000e+00j]
 [0.         +0.00000000e+00j 0.69499031+6.31839693e-10j
  0.69499031+1.24188550e-10j ... 0.69499031+1.19730210e-10j
  0.69499031+1.19730210e-10j 0.69499031+1.19730210e-10j]
 [0.         +0.00000000e+00j 0.69499031-2.25199477e-10j
  0.69499031+3.62077688e-13j ... 0.69499031-1.38940801e-11j
  0.69499031-1.38940801e-11j 0.69499031-1.38940801e-11j]
 ...
 [0.         +0.00000000e+00j 0.69499031-1.81503167e-10j
  0.69499031-1.13911933e-11j ... 0.69499031-7.30770797e-12j
  0.69499031-7.30770797e-12j 0.69499031-7.30770792e-12j]
 [0.         +0.00000000e+00j 0.69499031-1.81503167e-10j
  0.69499031-1.13911933e-11j ... 0.69499031-7.30770797e-12j
  0.69499031-7.30770797e-12j 0.69499031-7.30770792e-12j]
 [0.         +0.00000000e+00j 0.69499031-1.81503167e-10j
  0.69499031-1.13911933e-11j ... 0.69499031-7.30770789e-12j
  0.69499031-7.30770789e-12j 0.69499031-7.30770764e-12j]]
```

This is completely different from the original matrix. When I try the rank of other values, the image I get is still black.

Then I found a problem, when we calculate eigenvectors of a matrix, we often get different results like the values of different answers are opposite to each other. This is no problem, but when we calculate low rank approximation matrix, we need to use V's transposed matrix. When some eigenvectors take the opposite number, and some are not taken, the matrix after transposition will become confused.

(2) Close to success

Then I tried another method, only to calculate the eigenvalues and eigenvectors of AA^T, because the eigenvalues of AA^T and A^TA are the same, the eigenvectors of A^TA are equal to the results of transpose of A by eigenvectors of AA^T. So we can use the resultS of AA^T to calculate the eigenvectors of A^TA.

I continued to implement SVD using the method described in the previous paragraph.It should be noted that the eigenvectors of A^TA obtained by AA^T needs to be standardized. But I found the results were still not working.

(3) Got it

Then I found out that the loss of data during the calculation process is very large by checking the intermediate results. I think the problem is that the value in the original matrix is not floating point. Then I changed the value of the original matrix to float, like this:

```
[[  0.   0.   0. ...   0.   0.   0.]
 [  0. 255. 255. ... 255. 255. 255.]
 [  0. 255. 255. ... 255. 255. 255.]
 ...
 [  0. 255. 255. ... 255. 255. 255.]
 [  0. 255. 255. ... 255. 255. 255.]
 [  0. 255. 255. ... 255. 255. 255.]]
```

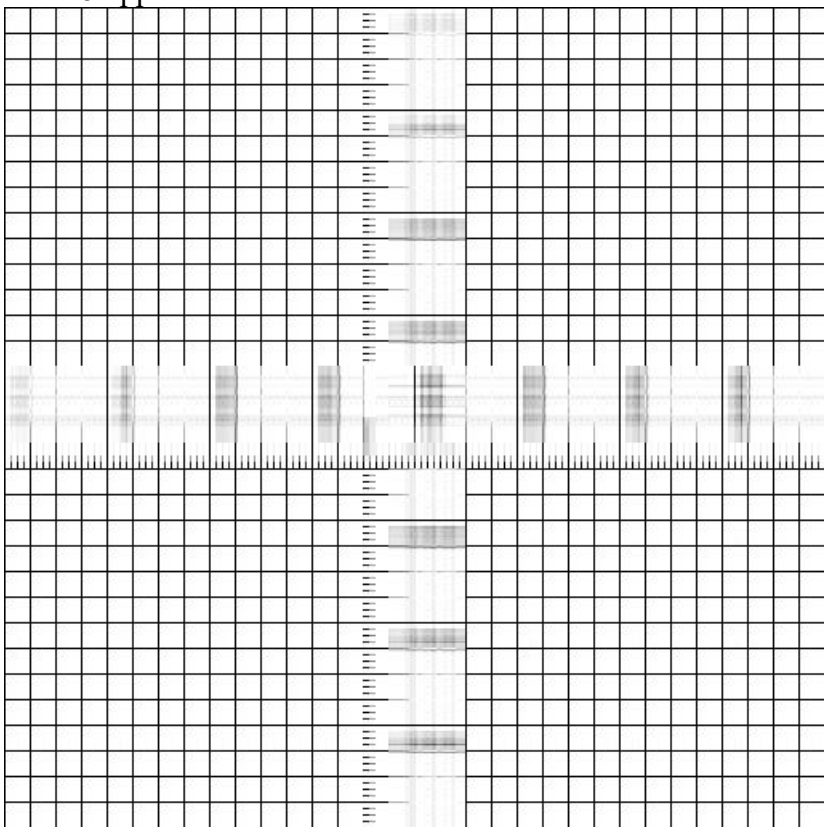
And this time, I got the correct results.

The core code part is shown below:

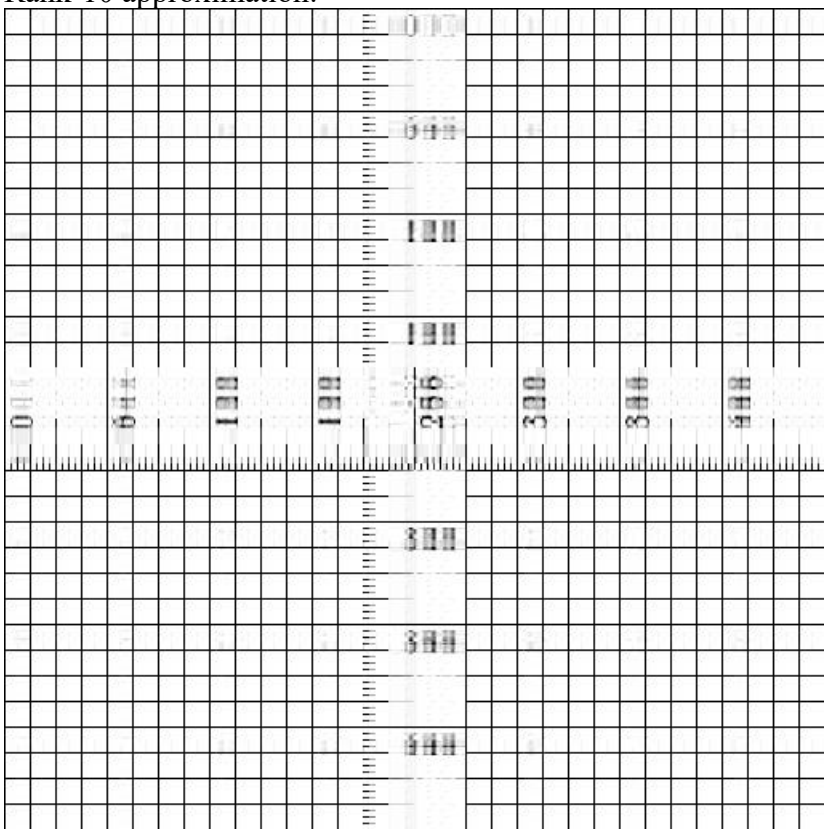
```
# calculate low rank approximation matrix
def approximation(rank):
    # newvectors1 represents the matrix U
    newvectors1 = np.zeros([len(vectors1), rank], dtype=complex)
    # newvectors2 represents transpose of the matrix V
    newvectors2 = np.zeros([rank, len(vectors2)], dtype=complex)
    for i in range(rank):
        sum = np.linalg.norm(vectors2[:, (np.argwhere(values1 == (newvalues2[i])))[0][0]])
        if sum == 0:
            newvectors2[i, :] = vectors2[:, (np.argwhere(values2 == (newvalues2[i])))[0][0]]
        else:
            # standardized vectors and store in newvectors2
            newvectors2[i, :] = vectors2[:, (np.argwhere(values2 == (newvalues2[i])))[0][0]]/sum
        # store in newvectors1
        newvectors1[:, i] = vectors1[:, (np.argwhere(values1 == (newvalues1[i])))[0][0]]
    diagsort = np.diag(np.sqrt(newvalues1))
    # diag represents the diagonal matrix of singular values
    diag = np.zeros([rank, rank], dtype=complex)
    for c in range(rank):
        diag[c][c] = diagsort[c][c]
    # calculate low rank approximation matrix
    resmatrix = newvectors1.dot(diag).dot(newvectors2)
    return resmatrix
```

Results

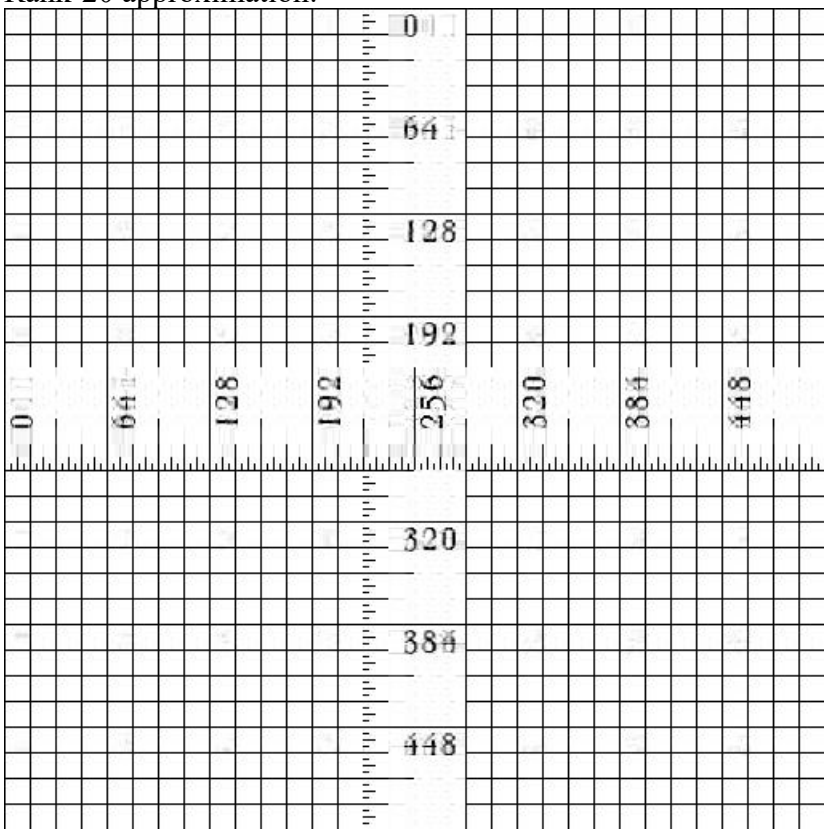
Rank-5 approximation:



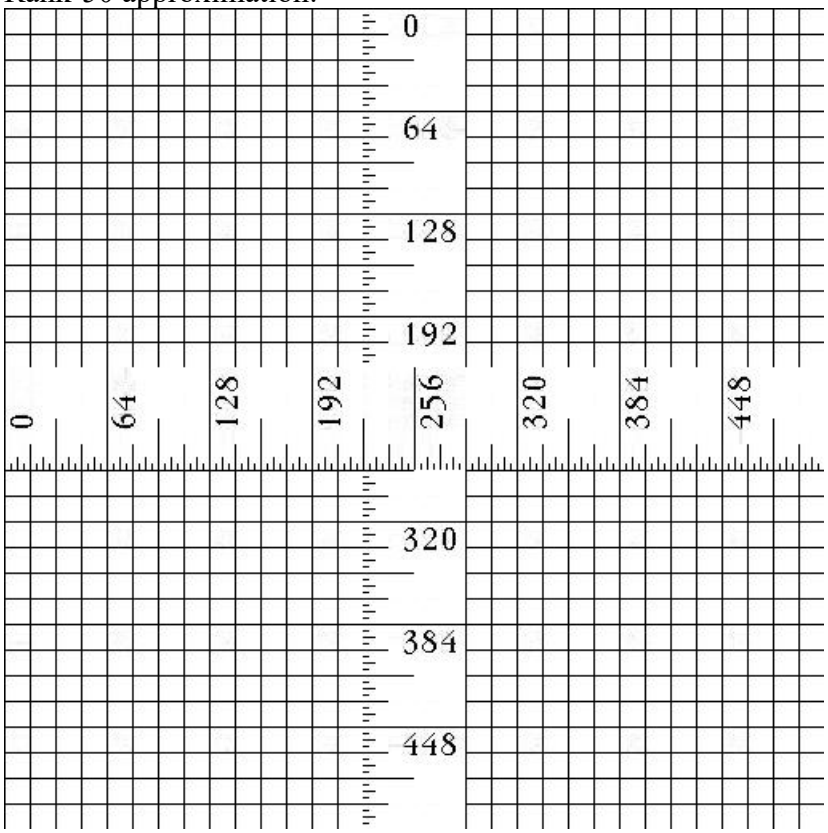
Rank-10 approximation:



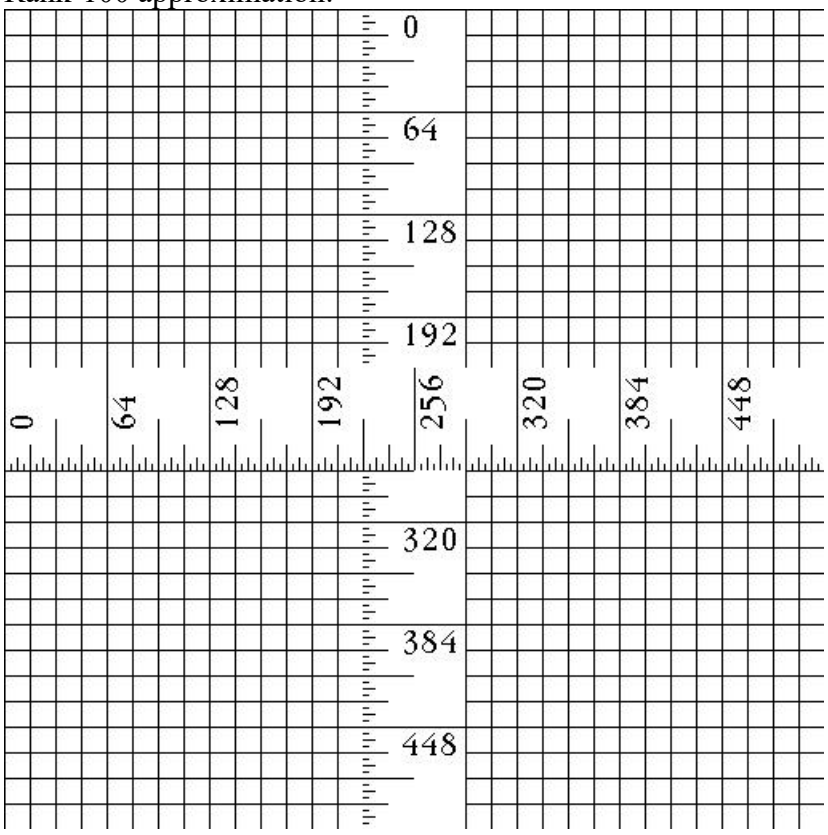
Rank-20 approximation:



Rank-50 approximation:



Rank-100 approximation:



Rank-200 approximation: