# CECS 229 Final Exam

Joshua Kasanjian
016036421

**Directions:** Submit a report on SVD image compression that addresses the following:

1. (5 pts) Explain how SVD is used to compress images. You must also define what cumulative energy is, and how it is used to determine the compression rank.

2. (10 pts) Write a Python program that compresses images using SVD. You must

   (i) create and manipulate `Matrix` objects out of the images

   (ii) use the built-in function `numpy.linalg.eigvals()` to find the eigenvalues (Do not use any other built-in functions).

   (iii) use a modified solver from HW #5 to find the necessary eigenvectors (do not use any built-in functions to find the eigenvectors).

   Attach your code to the end of this report.

3. Choose your own grayscale image with dimensions larger than $250 \times 250$. Compress it to a rank-$k$ approximation, $\tilde{A}_k$, so that it retains a cumulative energy, $E_k$, of at least

   (i) 95%

   (ii) 99%

   For each case, compute the error by finding the Frobenius norm[1] of the error matrix, $A - \tilde{A}_k$. Convert the approximation matrices, $\tilde{A}_{k_1}$ and $\tilde{A}_{k_2}$, to images and attach them to this report. Include the error with each image.

4. Additionally, find the rank-$k$ approximations for $k = 50, 100$, and 200. For each case, compute the cumulative energy it retains, $E_k$, and the error as in 2. Attach the compressed images to this report.

You must demo your program by compressing one image during the alloted final exam time: Thursday 5/17, 8-10 AM (VEC 402). You must submit a printed copy of your LaTeX report when you demo.

[1] The Frobenius norm of a matrix $A_{m \times n}$ is defined to be the square root of the sum of the squared entries, i.e. $||A||_F = \sqrt{\sum_{j=1}^{n} \sum_{i=1}^{m} a_{ij}^2}$

**Theory: Using SVD to Compress Images**

SVD, or Single Value Decomposition, is a way to express the data of an image by calculating the eigenvalues and corresponding eigenvectors from the matrix of data. The rank of the matrix determines how many eigenvalues and eigenvector are calculated and used to later reconstruct the image. This can used to reduce the file size of the image, while still retaining most of the image characteristics. The smaller the rank, the less detailed the compressed image, and the less storage it takes up.

The process converts the image into a matrix $A_{mxn}$, and calculates matrices $U, S, andV$ such that $A = USV^T$. $U$ is made up of the eigenvectors obtained by multiplying $A_{mxn}A_{nxm}^T$. $V$ is made up of the eigenvectors obtained by multiplying $A_{mxn}A_{nxm}^T$. $S$ is a matrix of diagonals, where each diagonal is the singular value, obtained by taking the square root of an eigenvalue.

SVD alone will not compress the image, as it reconstructs the original image data. During SVD, we can retain fewer singular values. Because singular values are arranged in descending order, the first singular values contains the most data. The following singular values contain less and less data, so discarding them won't distort the image.

We can approximate A with the equation $\tilde{A}_k = USV^T$. This involves reducing $U$ to have $k$ columns, reducing $S$ to only retain $k$ singular values, and reducing $V^T$ to have $k$ rows. The resulting matrix has less data, but still retains most of the important image information. This is called the rank-$k$ approximation of $A$, and can be expressed by:

$$\tilde{A}_k = \begin{bmatrix}\vec{u}_1 & \cdots & \vec{u}_k\end{bmatrix}_{mxk} \begin{bmatrix}\sigma_1 & 0 \\ 0 & \sigma_k\end{bmatrix}_{kxk} \begin{bmatrix}\vec{V}_1^T \\ \vdots \\ \vec{V}_k^T\end{bmatrix}_{kxn}$$

To determine the best $k$-value to use to create $\tilde{A}_k$ we can look at the cumulative energy, $E_k$, of the compressed image. $E_k = \frac{\sigma_1^2+...+\sigma_k^2}{\sigma_1^2+...+\sigma_r^2}$ where $k \leq r$. It represents the percentage of information the compressed image retained from the original. By plugging in the corresponding singular values to this equation, you can determine the rank needed to generate an image with the desired $E_k$. The larger $k$ is, the closer the approximation is to the original image, and the more data it contains. High values for $k$ will have a cumulative energy close to 1, or 100%, which means they retain the majority of image information. Choosing the right $k$ value is important to ensure that the compressed image retains the original image's quality, while still significantly reducing the file size.

**Compression Retaining 95% Cumulative Energy**

Figure 1: Compressed image $\tilde{A}_{23}$

- $k$ value = 23
- Error under $||*||_F$ = 9541.029353152238

**Compression Retaining 99% Cumulative Energy**

Figure 2: Compressed image $\tilde{A}_{94}$

- $k$ value = 94
- Error under $||*||_F$ = 4235.64054483804

**rank-50 Approximation**

Figure 3: Compressed image $\tilde{A}_{50}$

- $E_{50} = 0.9743725394887549$
- Error under $||*||_F = 6852.91108396479$

**rank-100 Approximation**

Figure 4: Compressed image $\tilde{A}_{100}$

- $E_{100} = 0.9914238271721179$
- Error under $||*||_F = 3964.3233693785423$

**rank-200 Approximation**

Figure 5: Compressed image $\tilde{A}_{200}$

- $E_{200} = 0.9993548639809772$
- Error under $|| * ||_F = 1087.2964132548116$

## Code

```
import image
import Matrix
import Vec
import math
import numpy
######################################################################

def SVD_WORKING(A,k):
   """
   Returns the rank-k approximation of matrix A
   INPUT: A - Matrix object with image data
        k - integer with k value for approximation
   OUTPUT: [0]- Matrix object: rank-k approximation of A
         [1]- list of all singular values obtained from A
   """
    values = numpy.linalg.svd(A.Rowsp)
    U = values[0]
    V = values[2]

    AAT = A.transpose() * A
    vals = numpy.linalg.eigvals(AAT.Rowsp) #CALLED FUNCTION TO GET EIGENVALUES
    s = []
    for v in vals:
        s.append(math.sqrt(abs(v)))
    s.sort()
    s = s[::-1] #now in ascending order

    m=[[0 for x in range(k)] for y in range(k)] #forming sigma matrix from first k sigma values
    for i in range(0,k):
        m[i][i] = s[i]

    newU = []
    l = len(U[0])
    for r in U:
        r2 = r[:-(l-k)]
        newU.append(r2) #removing columns so that there are k columns

    newV = []
    for a in range(k):
        newV.append(V[a]) #only taking first k rows

    U = Matrix(newU)
    S = Matrix(m)
    V = Matrix(newV)

    approx = U * S * V

    return approx, s



def cumulativeEnergy(s, k, r):
    top = 0
    for i in range(k):
        top += (s[i] * s[i])

    bot = 0
    for j in range(r):
        bot += (s[j]*s[j])
```

```
    return top / bot


def getKapprox(s, r, energy):
    bot = 0
    for i in range(r):
        bot += (s[i]*s[i])

    num = bot * energy
    top = 0
    k = 0
    while (top < num):
        top += (s[k]*s[k])
        k += 1

    return k


def calculateError(A, B):
    errorMatrix = A - B
    total = 0
    for row in errorMatrix.Rowsp:
        for entry in row:
            total += (entry * entry)
    return math.sqrt(total)




def part3and4():
    #PROBLEM 3
    M = png2graymatrix("pic.png")
    result = SVD_WORKING(M, 1)
    s = result[1] #gets list of all sigma values
    r = min(M.dim()) #gets rank of matrix

    print("FOR 95% CUMULATIVE ENERGY:")
    k1 = getKapprox(s, r, 0.95)
    print("k =", k1)
    M1 = SVD_WORKING(M, k1)[0]
    graymatrix2png(M1, "pic95.png")
    er1 = calculateError(M, M1)
    print("Error =", er1)

    print("\n\nFOR 99% CUMULATIVE ENERGY:")
    k2 = getKapprox(s, r, 0.99)
    print("k =", k2)
    M2 = SVD_WORKING(M, k2)[0]
    graymatrix2png(M2, "pic99.png")
    er2 = calculateError(M, M2)
    print("Error =", er2)


    #PROBLEM 4
    print("\n\nFOR RANK-50 APPROXIMATION:")
    M3 = SVD_WORKING(M, 50)[0]
    graymatrix2png(M3, "picRank50.png")
    e3 = cumulativeEnergy(s, 50, r)
    print("Cumulative energy:", e3)
    er3 = calculateError(M, M3)
```

```
    print("Error:", er3)

    print("\n\nFOR RANK-100 APPROXIMATION:")
    M4 = SVD_WORKING(M, 100)[0]
    graymatrix2png(M4, "picRank100.png")
    e4 = cumulativeEnergy(s, 100, r)
    print("Cumulative energy:", e4)
    er4 = calculateError(M, M4)
    print("Error =", er4)

    print("\n\nFOR RANK-200 APPROXIMATION:")
    M5 = SVD_WORKING(M, 200)[0]
    graymatrix2png(M5, "picRank200.png")
    e5 = cumulativeEnergy(s, 200, r)
    print("Cumulative energy:", e5)
    er5 = calculateError(M, M5)
    print("Error =", er5)


def main():
    part3and4()

    #DEMO:
    data = png2graymatrix("dog.png")
    compressedData = SVD_WORKING(data, 50)[0] #rank 20 approximation
    graymatrix2png(compressedData, "compressedDog.png")


main()
```