MATH3320 Project Report
Topic: Image Compression

ZHANG Xinfang 1155141566
ZHANG Yifan 1155141570

Monday 8th November, 2021

1. Introduction

In recent years, with the rapid developement in technology, multimedia product of digital information grows increasingly fast, which requires a large memory space and sufficient bandwidth in the storage and transmission process. Therefore, data compression becomes extremely vital for reducing the data redundancy to save more hardware space and transmission bandwidth.

Image compression is the process of removing redundant and irrelevant information, and efficiently encoding or reverting what remains without affecting or degrading its quality. The objective of image compression is to store or transmit data in an efficient form and to reduce the storage quantity as much as possible.

One useful techniques in image compression is to decompose an image into linear combination of elementary images with specific properties. By truncating some less important components in the image decomposition, we can compress an image to reduce the image size and achieve transform coding.

In this paper, we will discuss some useful image decomposition methods, demonstrate the applications of these decomsotion methods for image compression and analyze their advantages, disadvantages and applicability.

2. Image Compression Models

2.1 Singular Value Decomposition (SVD)

2.1.1 Definition

Every $m \times n$ image g has a singular value decomposition.
For any $g \in M_{m\times n}$, the singular value decomposition (SVD) of g is a matrix factorization give by

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

with $U \in M_{m\times m}$ and $V \in M_{n\times n}$ both unitary, and $\Sigma \in \mathbb{R}^{m\times n}$ is a diagonal matrix with diagonal elements $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0$ with r $\leq$ min (m,n). The diagonal elements are called singular values.
Denote the columns of U by $\{\vec{u_1}, \vec{u_2}, \ldots, \vec{u_m}\}$ and the columns of V by $\{\vec{v_1}, \vec{v_2}, \ldots, \vec{v_n}\}$
We have

$$g = U\Sigma V^T = \sum_{i=1}^{r} \sigma_i \vec{u_i}\vec{v_i}^T,$$

where $\vec{u_i}\vec{v_i}^T$ is called the eigenimages of g under SVD.

2.1.2 Error Analysis

For any k with $0 \leq k \leq r$, we define

$$g_k = \sum_{j=1}^{k} \sigma_j \vec{u_j}\vec{v_j}^T,$$

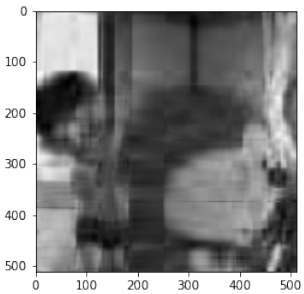
where g_k is called a rank-k approximation of g. This low rank matrix approximation can be applied to image compression.
Here we apply Frobenius norm to compute the error of approximation:
Let $f = \sum_{j=1}^{r} \sigma_j \vec{u_j}\vec{v_j}^T$ be the SVD of an $M \times N$ image f. For any k with

$k < r$, and $f_k = \sum_{j=1}^{k} \sigma_j \vec{u_j} \vec{v_j}^T$, we have

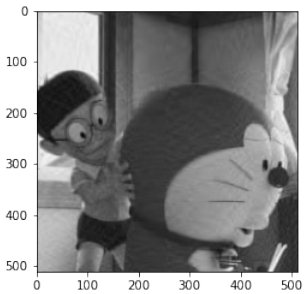
$$\| f - f_k \|_F^2 = \sum_{j=k+1}^{r} \sigma_j^2$$

2.1.3 Application

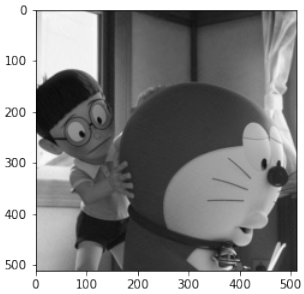
SVD can be applied for image compression, by removing terms(eigenimages) associated to small singular values. We will show examples of image compression using SVD, so-called 'rank-k approximation'. For simplicity, we convert all images into grayscale images.
First, we use Python to implement the rank-k approximation after computing SVD of inputed images. Here is the results for different rank-k.
Since the weightings of eigenimages depend on corresponding singular values, the larger the singular value is, the more important the eigen-image is. Hence, we consider ignoring singular values which are less than 0.5% of the largest singular value automatically. Our code and result are as follows:
Code: Results: Now we use the same algorithm to test another image called 'Lenna'. The comparison between SVD compressed images and the original image are as follows:



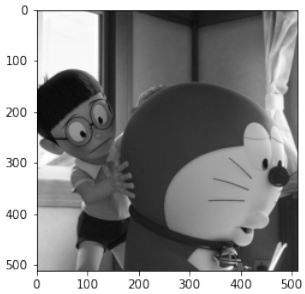
(a) k = 10



(b) k = 40



(c) k = 100



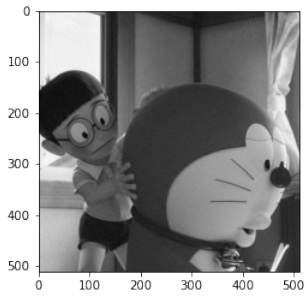
(d) Original

Figure 1: Doraemon and its compressed images using SVD

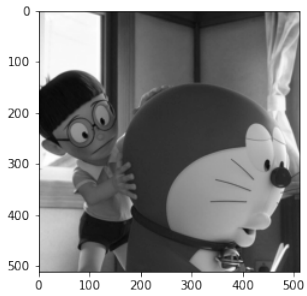
```python
# Use the SVD function to compute SVD of the image
u, s, vh = np.linalg.svd(lenna, full_matrices=True)
r, c = lenna.shape
# Generate a rank-k approximation image
img = np.zeros((r, c))
uh = np.array(u).T
k = 0

for i in range(0, r):
    if s[i]/s[0] >= 0.005:
        img = np.add(img, np.matmul(np.array([uh[i]]).T, np.array([vh[i]])) * s[i])
        k = i+1
    else:
        break

print(f'k = {k}')
plt.imshow(img, cmap="gray")
plt.savefig('lenna_auto_{k}.png')
```

✓ 0.3s

Figure 2: SVD Algorithm

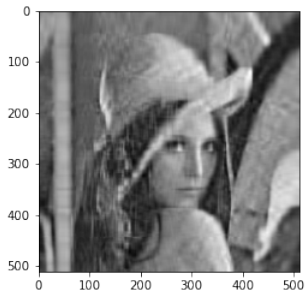


(a) Approximation with k = 86

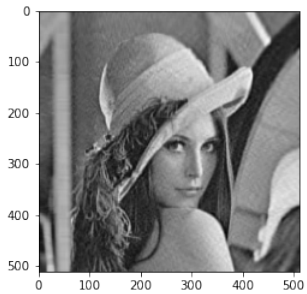


(b) Original

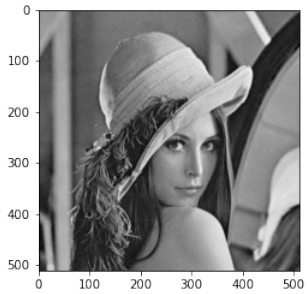
Figure 3: SVD algorithm results



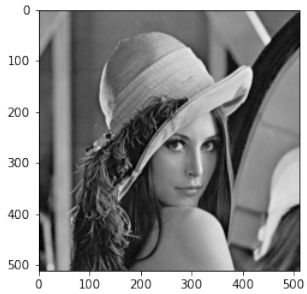
(a) k = 20



(b) k = 50



(c) Approximation with k = 107



(d) Original

Figure 4: Lenna and its compressed images using SVD