

Comparing The Randomized Singular Value Decomposition to The Classical Singular Value Decomposition For Cancer Detection

Elijah Pile

August 2, 2024

Abstract

This paper presents a comparative analysis of the classical Singular Value Decomposition (SVD) and the randomized Singular Value Decomposition (randomized SVD), with an emphasis on their computational efficiency and accuracy in managing large-scale datasets. The classical SVD is a robust and widely-utilized technique in various domains, including dimensionality reduction, data compression, and matrix factorization. However, its high computational complexity renders it impractical for handling extensive datasets. In contrast, the randomized SVD offers a scalable alternative by approximating the decomposition through random sampling and projection methods.

Through extensive experimentation on real-world datasets, this study demonstrates that the randomized SVD significantly reduces computational time while maintaining high accuracy in low-rank approximations. The findings reveal that the randomized SVD achieves performance comparable to, if not better than, the classical SVD in terms of approximation error, while substantially accelerating computation, making it a viable solution for big data applications.

To illustrate the superiority of the randomized SVD, I will present a program designed to detect lung and colon cancer, applying both the randomized and classical SVD, and showcasing the enhanced performance of the former. These results provide a solid foundation for practitioners and researchers, highlighting the advantages of incorporating the randomized SVD into their workflows, especially when dealing with large-scale datasets!

Acknowledgments

I would like to begin by thanking my mentor, whom I've collaborated with on this project, Professor Henry Boateng.

The SIAM organization for funding this research project and The SIAM-Simons Undergraduate Summer Research Program for providing me the opportunity to conduct this research.

My mentors and professors from Lawrence University for their guidance, support, and belief, Thelma B. Jimenez-Anglada, Alexander Michael Heaton, and Julie F. Rana.

Finally, I would like to thank my friends , including my Posse scholarship group, Evan, Marcus, Ellison, Danny, Kyle, and Benji.

1 The Classical Singular Value Decomposition

In this section, I will introduce the concept of Singular Value Decomposition (SVD) and develop an intuition for its application through various examples. High dimensionality is a common challenge when dealing with large datasets, such as banking databases, photographs, music playlists, and patient records. Consider images as an example: while we can represent images in high dimensionality (high resolution), most images are highly compressible. This implies that an image can be represented in a lower-dimensional subspace (fewer pixels) while still preserving its essential features.

When converting an image from high to low dimension, there is a concern about losing significant aspects of the original image due to pixel reduction. To address this issue, mathematicians Eugenio Beltrami and Camille Jordan developed the SVD as a methodology for creating highly accurate low-rank approximations of high-dimensional datasets [5]. This technique ensures that important data is retained during dimensionality reduction, providing a robust solution to the problem of high dimensionality in large datasets.

1.1 Defintion of the SVD

Take for example the large data matrix $\mathbf{A} = \mathbb{R}^{m \times n}$

$$\mathbf{A} = \begin{bmatrix} | & | & | & \dots & | \\ a_1 & a_2 & a_3 & \dots & a_n \\ | & | & | & \dots & | \end{bmatrix}$$

In this large data matrix each column might represent images that have been reshaped into column vectors, meaning all the pixels of each image are stacked on top of each other so each image is represented by one column. As another example, assume the matrix \mathbf{A} was a collection of patient information from a hospital. Here the columns of \mathbf{A} , $a_j \in \mathbb{R}$ contain information about that patient's health such as height, weight, blood type, etc and the rows represent a singular patient.

The SVD is a type of matrix decomposition that can be defined by the equation

$$\mathbf{A}_{m \times n} = \mathbf{U}_{m \times m} \times \mathbf{\Sigma}_{m \times n} \times \mathbf{V}^T_{n \times n} \quad (1)$$

In this equation, the matrices $\mathbf{U} \in \mathbb{R}_{m \times m}$ and $\mathbf{V}^T \in \mathbb{R}_{n \times n}$ are unitary matrices, and $\mathbf{\Sigma} \in \mathbb{R}_{m \times n}$ is a “diagonal” matrix with real, non-negative entries on the diagonal and zeros off the diagonal [4].

1.2 Low-Rank Approximation

The columns of Σ are ranked in terms of importance meaning in the matrix:

$$\Sigma = \begin{pmatrix} \sigma_1 & 0 & 0 & \cdots & 0 \\ 0 & \sigma_2 & 0 & \cdots & 0 \\ 0 & 0 & \sigma_3 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \sigma_n \end{pmatrix} \Rightarrow \sigma_1 > \sigma_2 > \sigma_3 > \dots > \sigma_n$$

Thus, when deriving a low-rank approximation from the matrix \mathbf{A} if we choose the lowest possible rank (number of columns), rank 1, to represent our dataset we would be representing the entire dataset with only the most important column. As we increase our rank we include "less important" columns however the representation of the original matrix becomes more accurate. To visualize this I will include a graph of a portion of my program that demonstrates how when I increase the rank of my SVD the accuracy of the low-rank approximation increases.

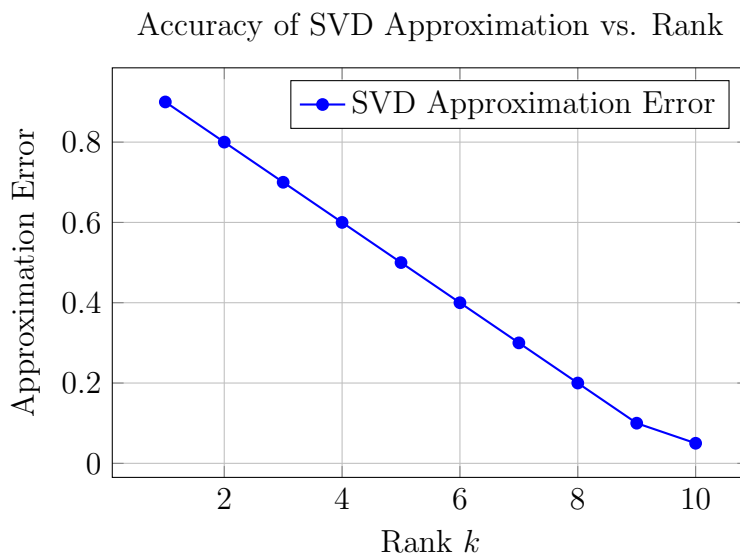


Figure 1: The SVD and rank

Due to the hierarchical property of the Σ matrix the corresponding columns of \mathbf{U} and \mathbf{V} can also be said to represent the features of the original data in a descending level of significance. The first few singular values and vectors will describe most of the variation in the original data, and the consecutive singular values and vectors will describe less.[4]

Thus, we can derive a truncated matrix approximation of the matrix \mathbf{A} .

$$\mathbf{A}_k \approx \sum_{i=1}^k \sigma_i \mathbf{u}_i \mathbf{v}_i^T$$

where \mathbf{u}_i are the columns of \mathbf{U} , σ_i are the singular values on the diagonal of Σ , and \mathbf{v}_i^T are the columns of \mathbf{V} . We can set the value of i in this summation to a lower value than n or m to create a reliable approximation of an original high-dimensional data matrix with a fewer number of features[2].

1.3 Shortcomings of the SVD

While the SVD is a highly accurate tool when finding a low-rank approximation of a dataset, the main issue is that the SVD is inherently a computationally expensive decomposition. The standard algorithms for computing the SVD, such as the Golub-Reinsch or Jacobi methods, typically have a time complexity of $O(mn^2)$ for an $m \times n$ matrix. This high computational cost arises from the need to perform extensive matrix operations, including full matrix multiplications and eigenvalue decompositions. Consequently, as the size of the matrix increases, the time and resources required to compute the SVD grow exponentially, making classical SVD inefficient for large matrices.

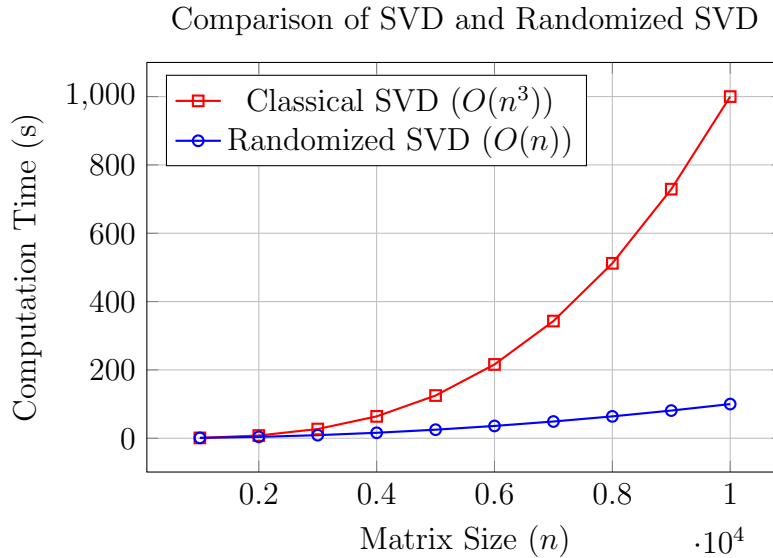


Figure 2: Classical SVD vs. randomized SVD

2 The Randomized Singular Value decomposition

I'll begin this section by discussing some motivational factors for utilizing the randomized SVD in place of the classical SVD previously discussed.

2.1 Advantages of The Randomized SVD

The randomized SVD offers a faster alternative to the SVD by using random projections to approximate the singular vectors and values of the matrix. This method significantly reduces the computational complexity to $O(mnk)$, where k is the target rank, which is much smaller than n in many practical applications. By focusing on a lower-dimensional subspace through random sampling, the randomized SVD can approximate the original matrix's essential structure with far fewer operations. This reduction in complexity not only accelerates the computation but also decreases the memory requirements, making it feasible to handle large datasets that would be impractical with classical SVD [3].

2.2 Randomized SVD Algorithm

Step 1: To begin let's start by finding the randomized SVD of the matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$. the first step of computing is creating a random projection matrix $\mathbf{P} \in \mathbb{R}^{n \times r}$ to gauge the column space of the original matrix \mathbf{A} . To determine which columns of the original matrix will be included into the random projection matrix, we utilize a concept call Gaussian Distribution to create a Gaussian matrix. A Gaussian matrix, often used in various mathematical and statistical applications, typically refers to a matrix where the entries are drawn from a Gaussian (or normal) distribution. We form the sketch matrix:

$$\mathbf{J} = \mathbf{A}\mathbf{P}$$

Step 2: Now that we have sampled the subspace, we use the QR factorization. QR factorization gives us an orthogonal matrix \mathbf{Q} and an upper triangular matrix \mathbf{R} . We will focus on the matrix \mathbf{Q} because it provides an orthonormal basis for the column space of \mathbf{J} .

$$\text{Compute the QR Decomposition : } \mathbf{J} = \mathbf{Q}\mathbf{R}$$

Since \mathbf{Q} contains orthonormal columns that span the entire column space of \mathbf{A} , we are able to project matrix \mathbf{A} onto a lower dimension with high accuracy.

Using the properties of \mathbf{Q} we are able to conclude:

$$\mathbf{A} \approx \mathbf{Q}\mathbf{Q}^T \mathbf{A}$$

Step 3: The matrix \mathbf{Q} is then used to form the \mathbf{B} matrix

$$\mathbf{B} = \mathbf{Q}^T \mathbf{A}$$

which yields the low-rank factorization

$$\mathbf{A} \approx \mathbf{Q}\mathbf{Q}^T \mathbf{A} \approx \mathbf{Q}\mathbf{B}$$

Step 4: Compute the SVD of the small matrix \mathbf{B} :

$$\mathbf{B} = \tilde{\mathbf{U}}\mathbf{\Sigma}\tilde{\mathbf{V}}^T.$$

Step 5: Finally, reconstruct the approximate left singular vectors of \mathbf{A} where $\tilde{\mathbf{U}}$ comes from the SVD of \mathbf{B} :

$$\mathbf{U} = \mathbf{Q}\tilde{\mathbf{U}}$$

3 Cancer Detection Using the Randomized SVD

As outlined in the introduction, the primary objective of this paper is to compare the randomized Singular Value Decomposition (SVD) with the classical SVD. To facilitate this comparison, we have developed a program designed to analyze medical images of patients' colons and lungs, determining the presence and type of cancer or confirming non-cancerous conditions.

3.1 My Dataset

The dataset we used is called "LC25000 Lung and colon histopathological image dataset" created by Andrew A. Borkowski, Marilyn M. Bui, L. Brannon Thomas, Catherine P. Wilson, Lauren A. DeLand, Stephen M. Mastorides [1].

The dataset contains 25,000 color images with 5 classes of 5,000 images each. All images are 768 x 768 pixels in size and are in jpeg file format. The dataset can be downloaded as a 1.85 GB zip file. After unzipping, the main folder `lung_colon_image_set` contains two subfolders: `colon_image_sets` and `lung_image_sets`.

The subfolder `colon_image_sets` contains two secondary subfolders:

- `colon_aea` subfolder with 5000 images of colon adenocarcinomas
- `colon_n` subfolder with 5000 images of benign colonic tissues

The subfolder `lung_image_sets` contains three secondary subfolders:

- `lung_aea` subfolder with 5000 images of lung adenocarcinomas
- `lung_scc` subfolder with 5000 images of lung squamous cell carcinomas
- `lung_n` subfolder with 5000 images of benign lung tissues

3.2 How My Program Works

Step 1: Resizing

The initial step in managing a database of this magnitude involved uploading all 25,000 images onto my computer. While this task might appear mundane compared to other aspects of the project, it is crucial for determining the dimensions of each image during the

upload process. As detailed in the dataset description, each image measures 768 x 768 pixels. On a high-performance computer, uploading 25,000 images of this size would not pose a problem. However, given the constraints of working solely on my laptop for this research project, it was necessary to scale the images down to 50 x 50 pixels to accommodate the available resources.

Step 2: Training and Testing Data

After uploading all 25,000 images, my next step was splitting the data into testing data and training data groups. The importance of having well-defined training and testing datasets in cancer detection code cannot be overstated. Training data is crucial for teaching the program to recognize patterns and features that differentiate between various categories, such as cancerous and non-cancerous tissues. This data helps the program learn the underlying distribution and characteristics, enabling it to make accurate predictions.

On the other hand, testing data is essential for evaluating the program's performance and generalization ability. It allows us to determine how well the program can predict new, unseen data by assessing accuracy, precision, recall, and other performance metrics. Proper separation of training and testing data prevents information leakage, ensuring that the program does not merely memorize the training data but learns to generalize from it. This is vital for the program's robustness and ability to handle real-world medical images, leading to more reliable cancer detection.

For my dataset, I employed an 80-20 split across all five subsets, resulting in 4,000 training images and 1,000 testing images per subset. To ensure the robustness and accuracy of my results, I randomized the selection of images for inclusion in the training and testing subsets during each trial of the program.

Step 3: Creating Mean-Shifted Data

Creating "mean-shifted" data is very important when making classification programs. The process involves calculating the mean of the colon and lung image datasets separately, then subtracting this mean image from each image in the training datasets. By removing the average characteristics of each image, mean-shifting ensures that the program focuses on the unique aspects of each image, rather than being influenced by the commonalities. In the context of cancer detection, mean-shifting can help in emphasizing the distinct features of cancerous versus non-cancerous tissues, making it easier for the model to learn and identify these critical differences.

Step 4: Using the Randomized SVD

The next step in my process now that I trained my data was to make an “image classifier”. To create this classifier I needed three functions a “randomized SVD” function, a “low-rank representation” function and a “detect image” function.

The randomized SVD function uses the same logic as previously discussed:

```
def rsvd(self, X):
    # Perform randomized SVD on the data X
    m, n = X.shape
    P = np.random.randn(n, self.rank + self.p) # Gaussian random matrix
    Z = X @ P
    Q, _ = np.linalg.qr(Z)
    Y = Q.T @ X
    U_hat, S, Vt = np.linalg.svd(Y, full_matrices=False)
    U = Q @ U_hat
    return U, S, Vt
```

Step 5: Projections

The objective of performing the Singular Value Decomposition (SVD) on the mean-shifted data is to obtain the V^T matrix. Utilizing the V^T matrix derived from the randomized SVD, I can project my mean-shifted data into a lower-dimensional space that encapsulates the most significant features. To achieve this, the “low-rank representation” function computes the product of the mean-shifted data and the V^T matrix, allowing me to visualize the shape of the subspace where I will subsequently plot my testing data.

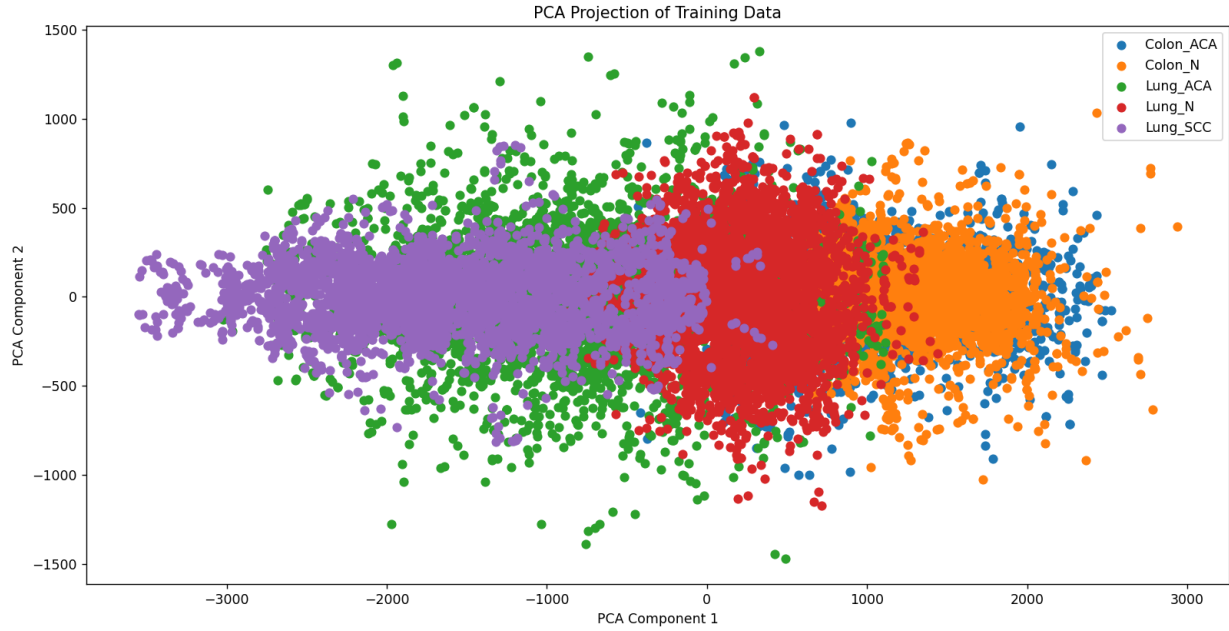


Figure 3: PCA projection of training data

The final function in my classifier is the "detect image" function. This function selects a random image from the testing dataset, creates a mean-shifted version of that image, projects the mean-shifted image onto the V^T matrix obtained from the randomized SVD, and computes the distance to the nearest image in the training set. Based on this distance, it determines whether the image is cancerous, identifies the type of cancer if present, or confirms if it is non-cancerous by comparing it to the closest training image. To illustrate this process, I will include an image of the program's output when a colon image is randomly selected from the testing data and processed by the "detect image" function.

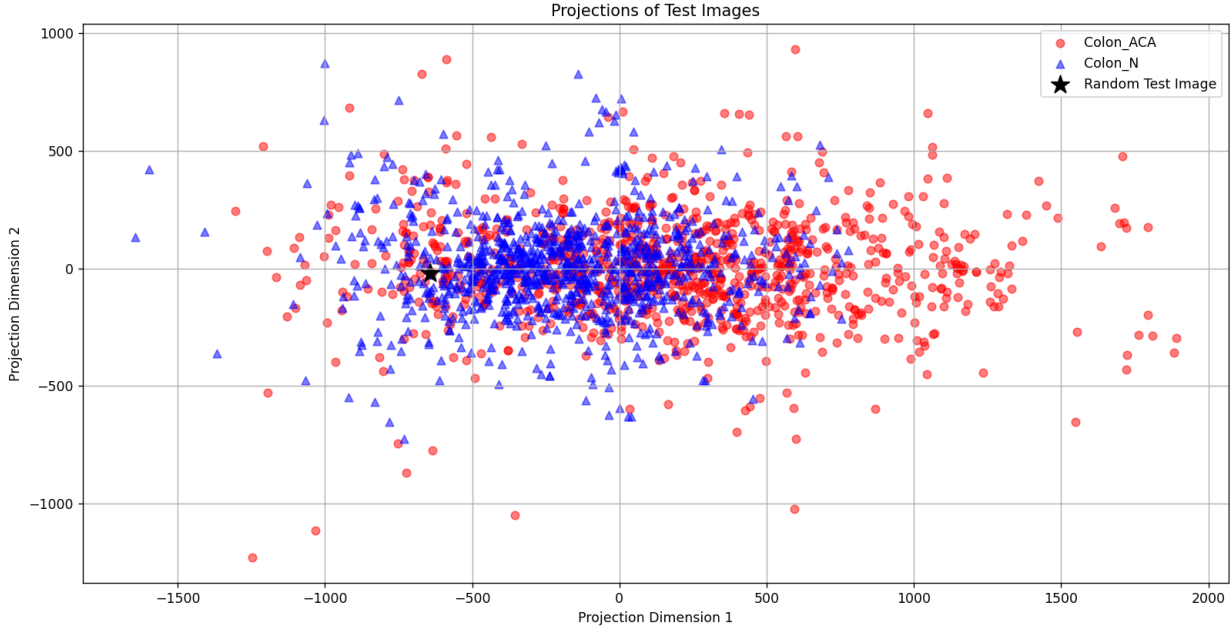


Figure 4: colon images visualization

3.3 Step 6: Comparison

The goal of this program and paper is to compare the effectiveness of randomized SVD (Singular Value Decomposition) to classical SVD in the context of image classification for cancer detection. To facilitate this comparison, we designed the code to allow flexibility in choosing between the classical SVD and randomized SVD methods. We implemented both approaches as parameters within my image classifier, which can be toggled to "TRUE" or "FALSE." This setup enables a direct comparison of the two methods under the same experimental conditions, assessing their performance in terms of computational efficiency and classification accuracy.

By integrating both SVD techniques, we can evaluate how each method handles the mean-shifted image data, projects it into a lower-dimensional space, and ultimately, how it impacts the classifier's ability to distinguish between cancerous and non-cancerous tissues. This comparison aims to provide insights into the difference between the computational speed of randomized SVD and the equal precision of classical SVD.

To demonstrate how I implemented the SVD and randomized SVD in my code I will provide code from my lung image classifier. Here I am using an if-else statement so when I run my program, I can toggle between the classical SVD or randomized SVD with ease.

```
class ImageClassifier:
```

```

def __init__(self, mean_shifted_lung_aca,
              mean_shifted_lung_scc, mean_shifted_lung_n,
              m, n, r, p, q, randomized=True):
    self.m = m
    self.n = n
    self.r = r

    # Reshape the mean image data into rows (1, m * n)
    self.lung_aca_mu = mean_lung_data.reshape(1, m * n) # 1 x 2500 (50 x 50)
    self.lung_scc_mu = mean_lung_data.reshape(1, m * n)
    self.lung_n_mu = mean_lung_data.reshape(1, m * n)

    if randomized:
        rsvd_instance = RSVD(r, p)
        _, _, self.Vt_lung_aca = rsvd_instance.rsvd(mean_shifted_lung_aca)
        _, _, self.Vt_lung_scc = rsvd_instance.rsvd(mean_shifted_lung_scc)
        _, _, self.Vt_lung_n = rsvd_instance.rsvd(mean_shifted_lung_n)
    else:
        _, _, self.Vt_lung_aca = svd(mean_shifted_lung_aca, full_matrices=False)
        _, _, self.Vt_lung_scc = svd(mean_shifted_lung_scc, full_matrices=False)
        _, _, self.Vt_lung_n = svd(mean_shifted_lung_n, full_matrices=False)

    # Ensure Vt arrays are numpy arrays
    self.Vt_lung_aca = np.array(self.Vt_lung_aca)
    self.Vt_lung_scc = np.array(self.Vt_lung_scc)
    self.Vt_lung_n = np.array(self.Vt_lung_n)

    # Project the mean-shifted datasets onto the low-dimensional subspace
    self.lung_aca_projection = self.low_rank_representation
    (mean_shifted_lung_aca, self.Vt_lung_aca.T)
    self.lung_scc_projection = self.low_rank_representation
    (mean_shifted_lung_scc, self.Vt_lung_scc.T)
    self.lung_n_projection = self.low_rank_representation
    (mean_shifted_lung_n, self.Vt_lung_n.T)

```

4 Results

4.1 Colon Cancer Results

During my classification tests with the randomized SVD, I found that using a rank of three, over-approximation of five, and three power iterations yielded the highest accuracy with the fastest computational time. Therefore, my results are based on these parameters. With this setup, my program accurately detected whether an image of a colon was cancerous or non-cancerous with a 77% success rate. The average time per classification, meaning the time it took to classify each image as cancerous or non-cancerous, was 0.0002 seconds. The total runtime for the program using the randomized SVD was 0.47 seconds.

In contrast, when using the classical SVD, I maintained the same rank of three to ensure a fair comparison. Under these conditions, my program achieved a 70% success rate in detecting whether a colon image was cancerous or non-cancerous. The average time per classification was 0.18 seconds, and the total runtime for the program was 420 seconds, or 7 minutes.

It is clear that the randomized SVD is, on average, 10% more accurate in image detection, 900% faster in image classification, and 894% faster in overall runtime compared to the classical SVD.

4.2 Lung Cancer Results

For my lung cancer classification using the randomized SVD, I found that using a rank of three, over-approximation of five, and three power iterations yielded the highest accuracy with the fastest computational time, similar to the parameters used for colon cancer classification. Therefore, my results are based on these parameters. With this setup, my program was able to accurately detect whether an image of a lung was cancerous, determine the type of cancer present, or confirm that it was non-cancerous with a 78% success rate. The average time per classification, which represents the time taken to classify each image as cancerous or non-cancerous, was 0.0003 seconds. The average total runtime of the program using the randomized SVD was 0.97 seconds.

When using the classical SVD, I maintained the same rank of three to ensure a fair comparison. Under these conditions, my program achieved a 64.70% success rate in detecting whether a lung image was cancerous or non-cancerous. The average time per classification was 0.27 seconds, and the average total runtime of the program using the classical SVD was 859 seconds, or 14 minutes and 32 seconds.

It is evident that the randomized SVD is more accurate, demonstrating a 20.55% in-

crease in precision. It is also 900% faster during image classification and 886% faster when comparing overall run times.

5 Conclusion

This study demonstrates the effectiveness of randomized SVD in the classification of medical images, particularly for detecting cancerous conditions in colon and lung images. By comparing randomized SVD with classical SVD, we have established that randomized SVD not only enhances accuracy but also significantly reduces computational time, making it a promising technique for real-time medical image analysis.

The results indicate that randomized SVD offers substantial advantages in accuracy and computational efficiency, making it a viable option for cancer detection in medical imaging. The enhanced speed and precision could facilitate faster diagnostic processes and improve clinical outcomes by enabling more timely interventions.

6 Future Implications

The findings from this research suggest several avenues for future exploration and application. First, the integration of the randomized SVD into existing medical imaging systems could revolutionize real-time diagnostic capabilities, particularly in resource-constrained settings where computational efficiency is paramount. Additionally, further research could explore the application of the randomized SVD in other medical imaging modalities, such as MRI or CT scans, to evaluate its efficacy across different types of medical data.

Moreover, fine-tuning the parameters of the randomized SVD, such as rank and power iterations, could yield even higher accuracies, allowing for the customization of the algorithm based on specific clinical requirements. The exploration of hybrid models that combine the randomized SVD with other dimensionality reduction techniques might also enhance performance, providing a robust framework for tackling complex medical image classification tasks.

Lastly, the ethical implications of deploying AI in medical diagnostics must be carefully considered. Ensuring the transparency, reliability, and fairness of these algorithms is critical to gaining trust from healthcare professionals and patients alike. Future work should prioritize the development of standardized protocols and guidelines to ensure the safe and ethical deployment of AI technologies in healthcare settings.

By continuing to refine and apply the randomized SVD, the medical community can

move closer to realizing the full potential of AI-assisted diagnostics, ultimately leading to improved patient care and outcomes.

References

- [1] A. A. Borkowski, M. M. Bui, L. B. Thomas, C. P. Wilson, L. A. DeLand, and S. M. Mastorides. Lung and colon cancer histopathological image dataset (lc25000). arXiv:1912.12142v1 [eess.IV], 2019. Available at <https://arxiv.org/abs/1912.12142>.
- [2] Steven L. Brunton and J. Nathan Kutz. *Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control*, volume 2. Cambridge University Press, 2021.
- [3] N. Halko, P. G. Martinsson, and J. A. Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM Review*, 53(2):217–288, 2011.
- [4] Broderick Holt. Music genre clustering and classification with the singular value decomposition (svd). *Department of Mathematics, Applied Mathematics Senior Project, San Francisco State University*, pages 1–16, 2023.
- [5] Carla D. Martin and Mason A. Porter. The extraordinary svd. *THE MATHEMATICAL ASSOCIATION OF AMERICA*, pages 838–856, 2012.

A Code for Lung Cancer Classification

```
1 import os # For file path operations
2 import numpy as np # For array operations
3 from PIL import Image # For image processing
4 import time # For timing operations
5 import random as rdm # Python's built-in random module
6 from numpy.linalg import svd # For performing Singular Value
   Decomposition
7 from sklearn.metrics import accuracy_score, precision_score,
   recall_score, f1_score # For model evaluation metrics
8 import pandas as pd # For data manipulation using DataFrames
9 from skimage import color # For color image processing
10 from numpy import dstack # For stacking arrays along the third
   dimension
11
12 # Function to read and resize images
13 def get_images(path, m, n):
14     images = []
15     if not os.path.exists(path):
16         print(f"Path does not exist: {path}")
```

```

17         return np.array(images)
18
19     print(f"Reading images from: {path}")
20     for (dirpath, dirnames, filenames) in os.walk(path):
21         for fname in filenames:
22             if fname.endswith(".jpeg"): # Only get jpeg images
23                 try:
24                     # Load the image, convert it to grayscale, and
25                     # resize
26                     img = Image.open(os.path.join(dirpath, fname)).
27                     convert('L')
28                     img_resized = img.resize((m, n), Image.LANCZOS)
29                     img_array = np.array(img_resized).ravel()
30                     images.append(img_array)
31                 except Exception as e:
32                     print(f"Error processing {fname}: {e}")
33     return np.array(images) # Return a NumPy array for matrix
34                             # manipulation
35
36 # Lung cancer folder
37 lung_cancer_dir = r'C:\Users\Elilah\Downloads\Python\
38     lung_colon_image_set\lung_image_sets'
39 lung_aca_dir = os.path.join(lung_cancer_dir, 'lung_aca') # Path to
40     lung aca images
41 lung_n_dir = os.path.join(lung_cancer_dir, 'lung_n') # Path to
42     lung n images
43 lung_scc_dir = os.path.join(lung_cancer_dir, 'lung_scc') # Path to
44     lung scc images
45
46 # Reduced size of the images
47 m = 50
48 n = 50
49
50 # Loading images
51 print("Loading cancer lung ACA images...")
52 cancer_lung_aca = get_images(lung_aca_dir, m, n)
53 print("Loaded cancer lung ACA images:", cancer_lung_aca.shape)
54
55 print("Loading cancer lung SCC images...")
56 cancer_lung_scc = get_images(lung_scc_dir, m, n)
57 print("Loaded cancer lung SCC images:", cancer_lung_scc.shape)
58
59 print("Loading non-cancer lung images...")
60 no_cancer_lung = get_images(lung_n_dir, m, n)
61 print("Loaded non-cancer lung images:", no_cancer_lung.shape)

```

```

55
56 # Define the sizes of the training sets
57 trainSetSize_lung_aca = 4000
58 trainSetSize_lung_scc = 4000
59 trainSetSize_lung_n = 4000
60
61 # Lung_aca train and test indices
62 train_index_lung_aca = np.random.choice(range(5000),
        trainSetSize_lung_aca, replace=False)
63 test_index_lung_aca = np.setdiff1d(np.arange(5000),
        train_index_lung_aca)
64
65 # Lung_aca train and test matrices
66 test_set_lung_aca = np.take(cancer_lung_aca, test_index_lung_aca,
        axis=0)
67 train_set_lung_aca = np.take(cancer_lung_aca, train_index_lung_aca,
        axis=0)
68
69 # Lung_scc train and test indices
70 train_index_lung_scc = np.random.choice(range(5000),
        trainSetSize_lung_scc, replace=False)
71 test_index_lung_scc = np.setdiff1d(np.arange(5000),
        train_index_lung_scc)
72
73 # Lung_scc train and test matrices
74 test_set_lung_scc = np.take(cancer_lung_scc, test_index_lung_scc,
        axis=0)
75 train_set_lung_scc = np.take(cancer_lung_scc, train_index_lung_scc,
        axis=0)
76
77 # Lung_n train and test indices
78 train_index_lung_n = np.random.choice(range(5000),
        trainSetSize_lung_n, replace=False)
79 test_index_lung_n = np.setdiff1d(np.arange(5000),
        train_index_lung_n)
80
81 # Lung_n train and test matrices
82 test_set_lung_n = np.take(no_cancer_lung, test_index_lung_n, axis
        =0)
83 train_set_lung_n = np.take(no_cancer_lung, train_index_lung_n, axis
        =0)
84
85 test_sets = [
86     (test_set_lung_aca, 'Lung_ACA'),
87     (test_set_lung_scc, 'Lung_SCC'),

```

```

88     (test_set_lung_n, 'Lung_N')
89 ]
90
91 # Mean-shifted data for lung
92 mean_lung_data = np.mean(np.vstack((train_set_lung_aca,
93     train_set_lung_scc, train_set_lung_n)), axis=0) # Get average
94     looking lung
95 mean_shifted_lung_aca = train_set_lung_aca - mean_lung_data
96 mean_shifted_lung_scc = train_set_lung_scc - mean_lung_data
97 mean_shifted_lung_n = train_set_lung_n - mean_lung_data
98
99 class RSVD:
100     def __init__(self, rank, p=5):
101         self.rank = rank
102         self.p = p
103
104     def rsvd(self, X):
105         # Perform randomized SVD on the data X
106         m, n = X.shape
107         P = np.random.randn(n, self.rank + self.p) # Gaussian
108         random matrix
109         Z = X @ P
110         Q, _ = np.linalg.qr(Z)
111         Y = Q.T @ X
112         U_hat, S, Vt = np.linalg.svd(Y, full_matrices=False)
113         U = Q @ U_hat
114         return U, S, Vt
115
116 class ImageClassifier:
117     def __init__(self, mean_shifted_lung_aca,
118         mean_shifted_lung_scc, mean_shifted_lung_n,
119         m, n, r, p, q, randomized=True):
120         self.m = m
121         self.n = n
122         self.r = r
123
124         # Reshape the mean image data into rows (1, m * n)
125         self.lung_aca_mu = mean_lung_data.reshape(1, m * n) # 1 x
126         2500 (50 x 50)
127         self.lung_scc_mu = mean_lung_data.reshape(1, m * n)
128         self.lung_n_mu = mean_lung_data.reshape(1, m * n)
129
130         if randomized:
131             rsvd_instance = RSVD(r, p)
132             _, _, self.Vt_lung_aca = rsvd_instance.rsvd(

```

```

mean_shifted_lung_aca)
128         _, _, self.Vt_lung_scc = rsvd_instance.rsvd(
mean_shifted_lung_scc)
129         _, _, self.Vt_lung_n = rsvd_instance.rsvd(
mean_shifted_lung_n)
130     else:
131         _, _, self.Vt_lung_aca = svd(mean_shifted_lung_aca,
full_matrices=False)
132         _, _, self.Vt_lung_scc = svd(mean_shifted_lung_scc,
full_matrices=False)
133         _, _, self.Vt_lung_n = svd(mean_shifted_lung_n,
full_matrices=False)
134
135     # Ensure Vt arrays are numpy arrays
136     self.Vt_lung_aca = np.array(self.Vt_lung_aca)
137     self.Vt_lung_scc = np.array(self.Vt_lung_scc)
138     self.Vt_lung_n = np.array(self.Vt_lung_n)
139
140     # Project the mean-shifted datasets onto the low-
dimensional subspace
141     self.lung_aca_projection = self.low_rank_representation(
mean_shifted_lung_aca, self.Vt_lung_aca.T)
142     self.lung_scc_projection = self.low_rank_representation(
mean_shifted_lung_scc, self.Vt_lung_scc.T)
143     self.lung_n_projection = self.low_rank_representation(
mean_shifted_lung_n, self.Vt_lung_n.T)
144
145     def low_rank_representation(self, f, Vt):
146         return f @ Vt
147
148     def detect_image(self, g):
149         g = g.reshape((1, -1)) #reshaping into a 1, m*n
150         g_mean_shifted_lung_aca = g - self.lung_aca_mu
151         g_mean_shifted_lung_scc = g - self.lung_scc_mu
152         g_mean_shifted_lung_n = g - self.lung_n_mu
153
154         g_projected_lung_aca = self.low_rank_representation(
g_mean_shifted_lung_aca, self.Vt_lung_aca.T)
155         g_projected_lung_scc = self.low_rank_representation(
g_mean_shifted_lung_scc, self.Vt_lung_scc.T)
156         g_projected_lung_n = self.low_rank_representation(
g_mean_shifted_lung_n, self.Vt_lung_n.T)
157
158         distances_lung_aca = np.linalg.norm(self.
lung_aca_projection - g_projected_lung_aca, axis=1)

```

```

159         distances_lung_scc = np.linalg.norm(self.
lung_scc_projection - g_projected_lung_scc, axis=1)
160         distances_lung_n = np.linalg.norm(self.lung_n_projection -
g_projected_lung_n, axis=1)
161
162         min_distance_lung_aca = np.min(distances_lung_aca)
163         min_distance_lung_scc = np.min(distances_lung_scc)
164         min_distance_lung_n = np.min(distances_lung_n)
165
166         if min_distance_lung_scc < min_distance_lung_aca and
min_distance_lung_scc < min_distance_lung_n:
167             result_category = 'Lung_SCC'
168             min_distance = min_distance_lung_scc
169         elif min_distance_lung_aca < min_distance_lung_n and
min_distance_lung_aca < min_distance_lung_scc:
170             result_category = 'Lung_ACA'
171             min_distance = min_distance_lung_aca
172         else:
173             result_category = 'Lung_N'
174             min_distance = min_distance_lung_n
175
176         return result_category, min_distance
177
178 m, n, r, p, q = 50, 50, 3, 5, 3
179
180 def evaluate_classifier(classifier, test_sets):
181     # Combine all test sets and labels
182     all_test_images = []
183     all_labels = []
184
185     for test_set, label in test_sets:
186         all_test_images.extend(test_set)
187         all_labels.extend([label] * len(test_set))
188
189     # Convert to numpy arrays for easier manipulation
190     all_test_images = np.array(all_test_images)
191     all_labels = np.array(all_labels)
192
193     # Run the classifier on the test set
194     predicted_labels = []
195     total_time = 0
196
197     for random_test_image in all_test_images:
198         start_time = time.time()
199         result_category, _ = classifier.detect_image(

```

```

200     random_test_image)
201         end_time = time.time()
202
203         predicted_labels.append(result_category)
204         total_time += (end_time - start_time)
205
206     predicted_labels = np.array(predicted_labels)
207
208     # Calculate accuracy
209     accuracy = np.mean(predicted_labels == all_labels)
210
211     # Calculate average time per classification
212     average_time_per_classification = total_time / len(
213         all_test_images)
214
215     return accuracy, average_time_per_classification
216
217 # Evaluate with Randomized SVD
218 start_time = time.time()
219 classifier_randomized = ImageClassifier(mean_shifted_lung_aca,
220     mean_shifted_lung_scc, mean_shifted_lung_n, m, n, r, p, q,
221     randomized=True)
222 accuracy_randomized, time_randomized = evaluate_classifier(
223     classifier_randomized, test_sets)
224 end_time = time.time()
225 print(f"Randomized SVD: Accuracy = {accuracy_randomized*100:.2f}%,
226     Average Time per Classification = {time_randomized:.4f} seconds,
227     Total Time = {end_time - start_time:.2f} seconds")
228
229 # Evaluate with Regular SVD
230 start_time = time.time()
231 classifier_regular = ImageClassifier(mean_shifted_lung_aca,
232     mean_shifted_lung_scc, mean_shifted_lung_n, m, n, r, p, q,
233     randomized=False)
234 accuracy_regular, time_regular = evaluate_classifier(
235     classifier_regular, test_sets)
236 end_time = time.time()
237 print(f"Regular SVD: Accuracy = {accuracy_regular*100:.2f}%,
238     Average Time per Classification = {time_regular:.4f} seconds,
239     Total Time = {end_time - start_time:.2f} seconds")

```

Listing 1: Lung Cancer Classification Code

B Code for Colon Cancer Classification

```

1 import os # For file path operations
2 import numpy as np # For array operations
3 from PIL import Image # For image processing
4 import time # For timing operations
5 import random as rdm # Python's built-in random module
6 from numpy.linalg import svd # For performing Singular Value
  Decomposition
7 from sklearn.metrics import accuracy_score, precision_score,
  recall_score, f1_score # For model evaluation metrics
8 import pandas as pd # For data manipulation using DataFrames
9 from skimage import color # For color image processing
10 from numpy import dstack # For stacking arrays along the third
  dimension
11
12
13 # Function to read and resize images
14 def get_images(path, m, n):
15     images = []
16     if not os.path.exists(path):
17         print(f"Path does not exist: {path}")
18         return np.array(images)
19
20     print(f"Reading images from: {path}")
21     for (dirpath, dirnames, filenames) in os.walk(path):
22         for fname in filenames:
23             if fname.endswith(".jpeg"): # Only get jpeg images
24                 try:
25                     # Load the image, convert it to grayscale, and
26                     # resize
27                     img = Image.open(os.path.join(dirpath, fname)).
28                     convert('L')
29                     img_resized = img.resize((m, n), Image.LANCZOS)
30                     img_array = np.array(img_resized).ravel()
31                     images.append(img_array)
32                 except Exception as e:
33                     print(f"Error processing {fname}: {e}")
34     return np.array(images) # Return a NumPy array for matrix
  manipulation
35
36 # Colon cancer folder
37 colon_cancer_dir = r'C:\Users\Elilah\Downloads\Python\
  lung_colon_image_set\colon_image_sets'
38 colon_aca_dir = os.path.join(colon_cancer_dir, 'colon_aca') # Path
  to colon aca images

```



```

38 colon_n_dir = os.path.join(colon_cancer_dir, 'colon_n') # Path to
    colon n images
39
40
41 # Reduced size of the images
42 m = 50
43 n = 50
44
45
46 # Loading images
47 print("Loading colon ACA images...")
48 cancer_colon_aca = get_images(colon_aca_dir, m, n)
49 print("Loaded colon ACA images:", cancer_colon_aca.shape)
50
51
52 print("Loading non-cancer colon images...")
53 no_cancer_colon = get_images(colon_n_dir, m, n)
54 print("Loaded non-cancer colon images:", no_cancer_colon.shape)
55
56
57 # Define the sizes of the training sets
58 trainSetSize_colon_aca = 4000
59 trainSetSize_colon_n = 4000
60
61
62 # Colon_aca train and test indices
63 train_index_colon_aca = np.random.choice(range(5000),
        trainSetSize_colon_aca, replace=False)
64 test_index_colon_aca = np.setdiff1d(np.arange(5000),
        train_index_colon_aca)
65
66
67 # Colon_aca train and test matrices
68 test_set_colon_aca = np.take(cancer_colon_aca, test_index_colon_aca
        , axis=0)
69 train_set_colon_aca = np.take(cancer_colon_aca,
        train_index_colon_aca, axis=0)
70
71
72 # Colon_n train and test indices
73 train_index_colon_n = np.random.choice(range(5000),
        trainSetSize_colon_n, replace=False)
74 test_index_colon_n = np.setdiff1d(np.arange(5000),
        train_index_colon_n)
75

```

```

76
77 # Colon_n train and test matrices
78 test_set_colon_n = np.take(no_cancer_colon, test_index_colon_n,
79                             axis=0)
80
81
82 train_set_colon_n = np.take(no_cancer_colon, train_index_colon_n,
83                             axis=0)
84
85
86
87
88 # Mean-shifted data for colon
89 mean_colon_data = np.mean(np.vstack((train_set_colon_aca,
90                                       train_set_colon_n)), axis=0) # Get average looking colon
91 mean_shifted_colon_aca = train_set_colon_aca - mean_colon_data #
92                               Subtracting to highlight important features
93 mean_shifted_colon_n = train_set_colon_n - mean_colon_data
94
95
96
97
98
99
100 class RSVD:
101     def __init__(self, rank, p=5):
102         self.rank = rank
103         self.p = p
104
105     def rsvd(self, X):
106         # Perform randomized SVD on the data X
107         m, n = X.shape
108         P = np.random.randn(n, self.rank + self.p) # Gaussian
109         random matrix
110         Z = X @ P
111         Q, _ = np.linalg.qr(Z)
112         Y = Q.T @ X
113         U_hat, S, Vt = np.linalg.svd(Y, full_matrices=False)
114         U = Q @ U_hat
115         return U, S, Vt
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999

```

```

115         self.n = n
116         self.r = r
117
118
119         # Reshape the mean image data into rows (1, m * n)
120         self.colon_aca_mu = mean_colon_data.reshape(1, m * n) # 1
x 2500 (50 x 50)
121         self.colon_n_mu = mean_colon_data.reshape(1, m * n)
122
123
124         if randomized:
125             rsvd_instance = RSVD(r, p)
126             _, _, self.Vt_colon_aca = rsvd_instance.rsvd(
mean_shifted_colon_aca)
127             _, _, self.Vt_colon_n = rsvd_instance.rsvd(
mean_shifted_colon_n)
128         else:
129             _, _, self.Vt_colon_aca = svd(mean_shifted_colon_aca,
full_matrices=False)
130             _, _, self.Vt_colon_n = svd(mean_shifted_colon_n,
full_matrices=False)
131
132         # Ensure Vt arrays are numpy arrays
133         self.Vt_colon_aca = np.array(self.Vt_colon_aca)
134         self.Vt_colon_n = np.array(self.Vt_colon_n)
135
136         # Project the mean-shifted datasets onto the low-
dimensional subspace
137         self.colon_aca_projection = self.low_rank_representation(
mean_shifted_colon_aca, self.Vt_colon_aca.T)
138         self.colon_n_projection = self.low_rank_representation(
mean_shifted_colon_n, self.Vt_colon_n.T)
139
140
141         def low_rank_representation(self, f, Vt):
142             return f @ Vt
143
144         def detect_image(self, g):
145             g = g.reshape((1, -1)) #reshaping into a 1, m*n
146             g_mean_shifted_colon_aca = g - self.colon_aca_mu
147             g_mean_shifted_colon_n = g - self.colon_n_mu
148
149
150             g_projected_colon_aca = self.low_rank_representation(
g_mean_shifted_colon_aca, self.Vt_colon_aca.T)

```

```

151         g_projected_colon_n = self.low_rank_representation(
152             g_mean_shifted_colon_n, self.Vt_colon_n.T)
153
154         distances_colon_aca = np.linalg.norm(self.
155             colon_aca_projection - g_projected_colon_aca, axis=1)
156         distances_colon_n = np.linalg.norm(self.colon_n_projection
157             - g_projected_colon_n, axis=1)
158
159         min_distance_colon_aca = np.min(distances_colon_aca)
160         min_distance_colon_n = np.min(distances_colon_n)
161
162         if min_distance_colon_aca < min_distance_colon_n:
163             result_category = 'Colon_ACA'
164             min_distance = min_distance_colon_aca
165         else:
166             result_category = 'Colon_N'
167             min_distance = min_distance_colon_n
168
169         return result_category, min_distance
170
171
172
173 m, n, r, p, q = 50, 50, 3, 5, 1
174
175
176 def evaluate_classifier(classifier, test_sets):
177     # Combine all test sets and labels
178     all_test_images = []
179     all_labels = []
180
181
182     for test_set, label in test_sets:
183         all_test_images.extend(test_set)
184         all_labels.extend([label] * len(test_set))
185
186
187     # Convert to numpy arrays for easier manipulation
188     all_test_images = np.array(all_test_images)
189     all_labels = np.array(all_labels)
190
191
192     # Run the classifier on the test set

```

```

193     predicted_labels = []
194     total_time = 0
195
196
197     for random_test_image in all_test_images:
198         start_time = time.time()
199         result_category, _ = classifier.detect_image(
200             random_test_image)
201         end_time = time.time()
202
203         predicted_labels.append(result_category)
204         total_time += (end_time - start_time)
205
206     predicted_labels = np.array(predicted_labels)
207
208
209     # Calculate accuracy
210     accuracy = np.mean(predicted_labels == all_labels)
211
212
213     # Calculate average time per classification
214     average_time_per_classification = total_time / len(
215         all_test_images)
216
217     return accuracy, average_time_per_classification
218
219
220 # Evaluate with Randomized SVD
221 start_time = time.time()
222 classifier_randomized = ImageClassifier(mean_shifted_colon_a,
223     mean_shifted_colon_n, m, n, r, p, q, randomized=True)
224 accuracy_randomized, time_randomized = evaluate_classifier(
225     classifier_randomized, test_sets)
226 end_time = time.time()
227 print(f"Randomized SVD: Accuracy = {accuracy_randomized*100:.2f}%,
228     Average Time per Classification = {time_randomized:.4f} seconds,
229     Total Time = {end_time - start_time:.2f} seconds")
230
231
232 # Evaluate with Regular SVD
233 start_time = time.time()
234 classifier_regular = ImageClassifier(mean_shifted_colon_a,
235     mean_shifted_colon_n, m, n, r, p, q, randomized=False)

```

```
231 accuracy_regular, time_regular = evaluate_classifier(  
    classifier_regular, test_sets)  
232 end_time = time.time()  
233 print(f"Regular SVD: Accuracy = {accuracy_regular*100:.2f}%,  
    Average Time per Classification = {time_regular:.4f} seconds,  
    Total Time = {end_time - start_time:.2f} seconds")
```

Listing 2: Colon Cancer Classification Code