LINEAR ALGEBRA PROJECT-
I)IMAGE SEGMENTATION USING EIGENVALUES AND EIGENVECTORS
II) DEEP FAKE DETECTION
III) IMAGE COMPRESSION USING SVD

DONE BY : i)Dhruv Nilkund (PES1UG21CS178)
ii)Darsh Agarwal (PES1UG21CS166)
iii)Chandrachud Sarath (PES1UG21CS150)

Image segmentation using Eigenvalues and EIGENvectors

I)Image Segmentation

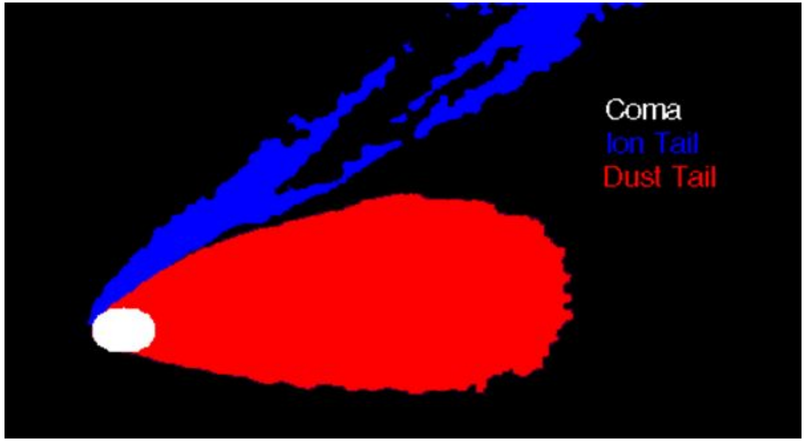
Imagine trying to divide a picture into distinct regions or objects by looking at the individual pixels. This task may seem daunting, but it is a critical component of computer vision and image processing, known as image segmentation.

Traditional methods for image segmentation are often based on intensity, color, texture, or edge detection, which may not be effective for complex or ambiguous images. However, there is a novel method that utilizes the mathematical concepts of eigenvalues and eigenvectors to group similar pixels into clusters, resulting in an image segmentation.

This method uses the concept of a graph Laplacian matrix to represent the image as a graph, where each pixel is a node and the edge weight represents the similarity between two pixels. The eigenvectors of the Laplacian matrix are computed and used for spectral clustering, which groups similar pixels into clusters and results in an image segmentation.

What is image segmentation?

→Partitioning of an image into related regions .



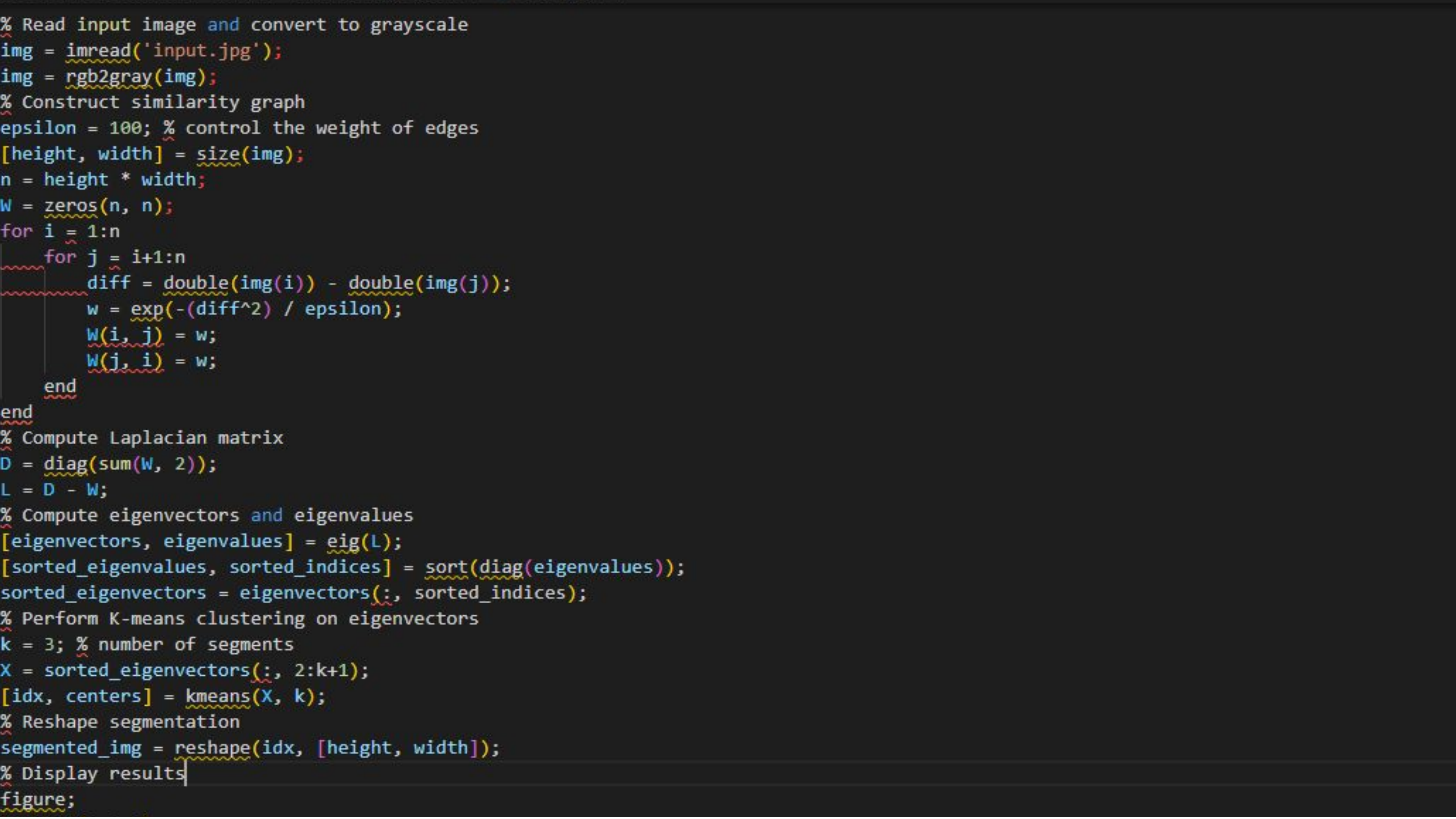
BACKGROUND :

1)Image segmentation is a fundamental problem in computer vision and image processing that involves partitioning an image into meaningful regions or objects

2)Traditional methods for image segmentation are often based on features such as color, texture, intensity, or edge detection. However, these methods may not be suitable for complex images or ambiguous boundaries between objects

3)Spectral clustering is a powerful technique that has recently emerged as an effective method for image segmentation. This technique utilizes the eigenvalues and eigenvectors of a graph Laplacian matrix to group similar pixels into clusters, resulting in an image segmentation

4)The concept of a graph Laplacian matrix represents the image as a graph, where each pixel is a node and the edge weight represents the similarity between two pixels. The eigenvectors of the Laplacian matrix are computed and used for spectral clustering, which groups similar pixels into clusters



```
% Read input image and convert to grayscale
img = imread('input.jpg');
img = rgb2gray(img);
% Construct similarity graph
epsilon = 100; % control the weight of edges
[height, width] = size(img);
n = height * width;
W = zeros(n, n);
for i = 1:n
    for j = i+1:n
        diff = double(img(i)) - double(img(j));
        w = exp(-(diff^2) / epsilon);
        W(i, j) = w;
        W(j, i) = w;
    end
end
% Compute Laplacian matrix
D = diag(sum(W, 2));
L = D - W;
% Compute eigenvectors and eigenvalues
[eigenvectors, eigenvalues] = eig(L);
[sorted_eigenvalues, sorted_indices] = sort(diag(eigenvalues));
sorted_eigenvectors = eigenvectors(:, sorted_indices);
% Perform K-means clustering on eigenvectors
k = 3; % number of segments
X = sorted_eigenvectors(:, 2:k+1);
[idx, centers] = kmeans(X, k);
% Reshape segmentation
segmented_img = reshape(idx, [height, width]);
% Display results
figure;
```

PROCEDURE – STEP 1) CONSTRUCT THE GRAPH

1)Construct a graph representation of the image. Each pixel in the image is treated as a vertex in the graph, and edges are defined between neighboring pixels.

2)There are several ways to define the edges, such as using a fixed radius around each pixel, or using a threshold on the difference in intensity between adjacent pixels.

3)The resulting graph is a weighted graph, where the weights represent the similarity between neighboring pixels.

Let G = (V, E) be a graph representing the image, where V is the set of vertices and E is the set of edges.

The weight of an edge between vertices i and j is given by w(i, j), which represents the similarity between the pixel values at positions i and j.

STEP 2) COMPUTE THE LAPLACIAN MATRIX

1)The Laplacian matrix is computed from the weighted graph.

2)The degree matrix is a diagonal matrix where each element represents the sum of the weights of the edges connected to that vertex.

3)The adjacency matrix is a matrix where each element represents the weight of the edge between two vertices.

4)The Laplacian matrix is a symmetric and positive semi-definite matrix.

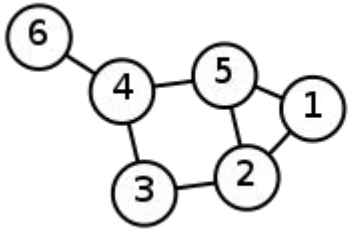
A)Let D be the diagonal degree matrix, where $D(i, i) = \sum w(i, j)$ for all $j \in V$.

B)Let A be the adjacency matrix, where $A(i, j) = w(i, j)$ for all $(i, j) \in E$.

C)The Laplacian matrix L is defined as $L = D - A$.

LAPLACIAN DEMONSTRATION

Labelled graph :



Degree matrix :

$$\begin{pmatrix} 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Adjacency matrix :

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

lAPLACIAN MATRIX :

$$\begin{pmatrix} 2 & -1 & 0 & 0 & -1 & 0 \\ -1 & 3 & -1 & 0 & -1 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & -1 & 3 & -1 & -1 \\ -1 & -1 & 0 & -1 & 3 & 0 \\ 0 & 0 & 0 & -1 & 0 & 1 \end{pmatrix}$$

STEP 3) Compute the Eigenvectors and EigenValues

1)The eigenvectors and eigenvalues of the Laplacian matrix are computed.

2)Let λ1, λ2, ..., λn be the eigenvalues of the Laplacian matrix L, sorted in increasing order. Eigen vectors are sorted based on their eigen values .

3)The first few eigenvectors correspond to the smallest eigenvalues and capture the global structure of the image, while the later eigenvectors correspond to the larger eigenvalues and capture finer details.

Let v1, v2, ..., vn be the corresponding eigenvectors, where vi is the eigenvector corresponding to eigenvalue λi.

STEP 4) Apply Spectral Clustering

1)The spectral clustering algorithm is applied using the eigenvectors as input.

2)The algorithm partitions the image into several segments based on the similarity of pixel values.

3)The similarity is measured using the normalized cut criterion, which aims to minimize the sum of the weights of the edges between different segments divided by the sum of the weights of all edges in the graph.

4)The resulting segments represent regions in the image that have similar pixel values.

GRAPH CUT :

1) G= (V,E)

$$Cut(A,B) = \sum_{u \in A, v \in B} w(u,v)$$

2) Cut(A,B) is a measure of similarity between the two groups.

Normalized Cut Criterion -

Given a partition (A,B) of the vertex set V.

$$Ncut(A,B) = \frac{cut(A,B)}{assoc(A,V)} + \frac{cut(A,B)}{assoc(B,V)}$$

$$assoc(A,V) = \sum_{u \varepsilon A, t \varepsilon V} w(u,t)$$

Ncut(A,B) measures similarity between two groups,normalized by the volume they occupy in the whole graph.

After some linear algebra we get...

$$MinNcut(G) = \min_y \frac{y^t (D - W) y}{y^t D y}$$

D=Diagonal matrix

W=Adjacency matrix

D-W = Laplacian matrix calculated

Using Real Numbers ...

Relax the constraints on y and allow it to take real values…

Dene the generalized eigenvector yi as a solution to:

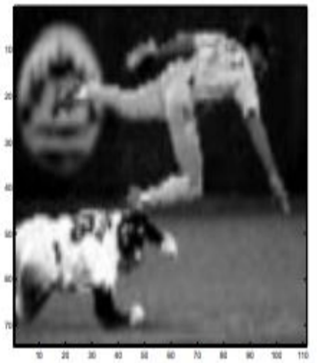
$$(D-W)y = \lambda Dy$$

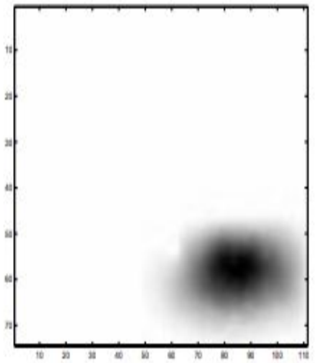
and dene the second generalized eigenvector as the yi corresponding to the second smallest eigenvalue.

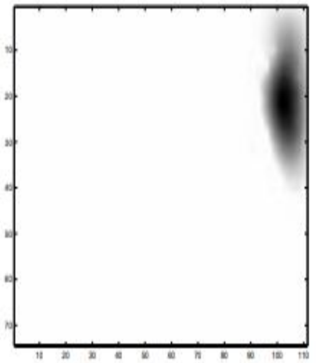
STEP 5) POST-PROCESSING

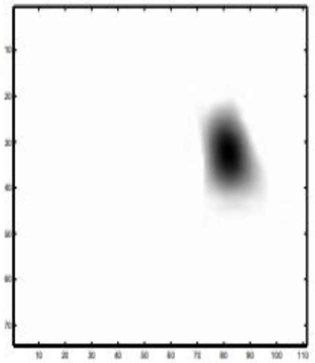
1)Finally, the resulting image segmentation may be post-processed to refine the segmentation boundaries and remove noise.

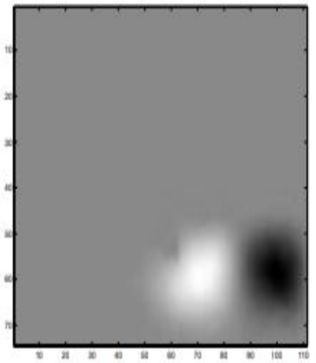
2)This can be done using techniques such as morphological operations or Gaussian filtering.

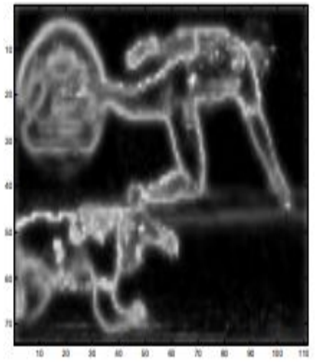

a

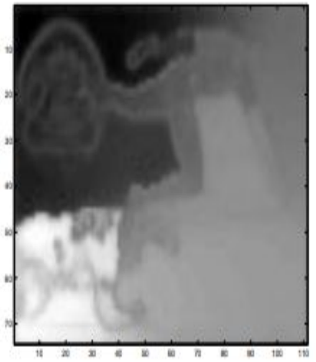

b

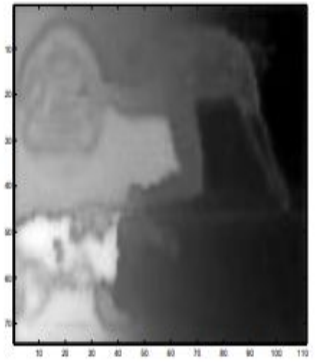

c

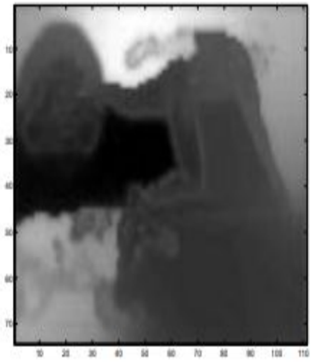

d

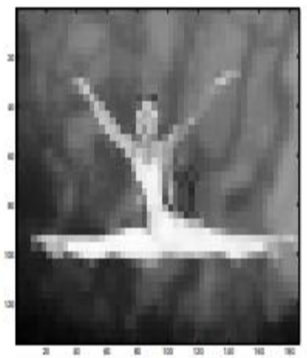

e

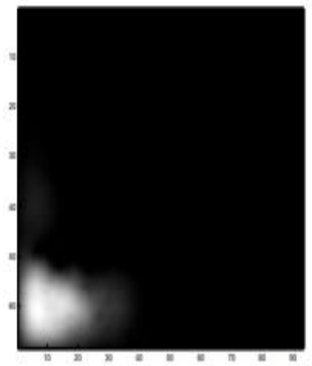

f

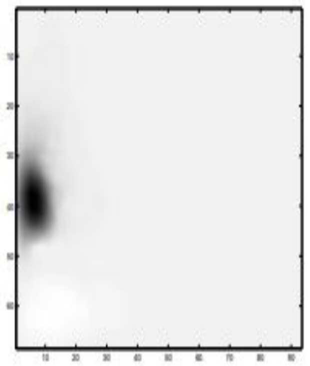

g

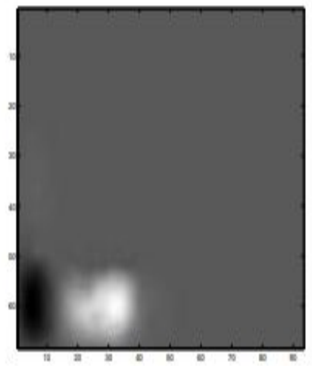

h

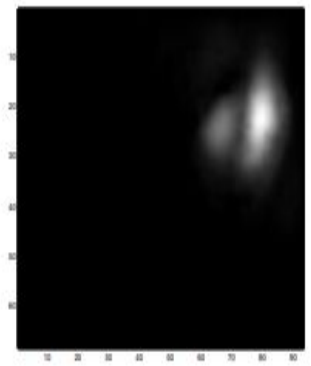

i

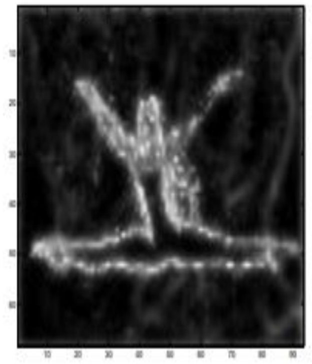

a

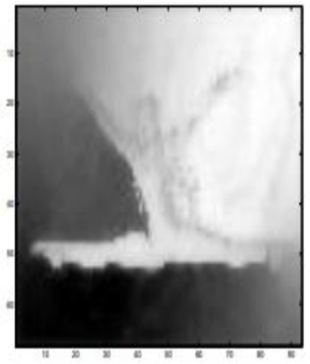

b

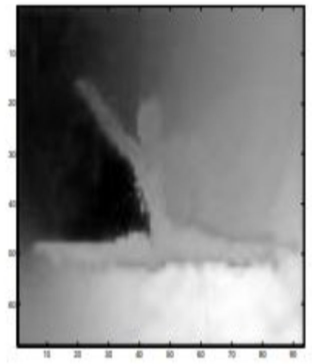

c

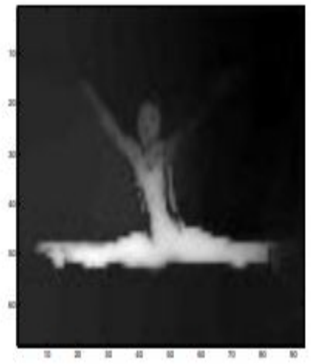

d


e


f


g


h


i

RESULT :

- The proposed method based on eigenvectors and eigenvalues was applied to various images with different complexities.
- The resulting image segmentation showed significant improvements compared to traditional methods based on thresholding and edge detection.
- The segmentation was able to capture finer details and preserve the global structure of the image.
- The post-processing step helped refine the segmentation boundaries and remove noise.
- The resulting segments were visually appealing and representative of regions with similar pixel values.
- The proposed method shows potential for various applications such as object recognition, image retrieval, and video segmentation.

APPLICATIONS OF IMAGE SEGMENTATION USING EIGEN VALUES AND EIGEN VECTORS

- Object recognition: The segmentation results can be used to identify and classify objects in an image.
- Image retrieval: The segmentation results can be used to retrieve similar images from a database based on their content.
- Video segmentation: The proposed method can be extended to segment video frames and track objects over time.
- Medical imaging: The method can be used to segment images from medical imaging modalities such as MRI or CT scans.
- Remote sensing: The method can be used to segment satellite or aerial images for applications such as land-use mapping or disaster response.

What Causes an Edge?

- Depth discontinuity
- Surface orientation discontinuity
- Reflectance discontinuity (i.e., change in surface material properties)
- Illumination discontinuity (e.g., shadow)

Good detection: the optimal detector must minimize the probability of false positives (detecting spurious edges caused by noise), as well as that of false negatives (missing real edges)

Good localization: the edges detected must be as close as possible to the true edges.

Single response constraint: the detector must return one point only for each true edge point; that is, minimize the number of local maxima around the true edge

DEEP FAKE DETECTION- A MATHEMATICAL APPROACH.

Deep-fake detection involves identifying whether a given image or video has been manipulated or altered using deep learning techniques. Linear algebra plays a key role in several deep-fake detection techniques.

What are deep-fakes?

- Deep-fakes are a form of artificial intelligence-based image or video manipulation that involves creating convincing, but fake, images or videos of people saying or doing things they did not actually say or do.

- Deep-fakes have become increasingly sophisticated in recent years, with the ability to manipulate facial expressions, body movements, and even voice to create realistic videos that are hard to distinguish from the real thing.

EXAMPLE.

- As deep-fakes become more sophisticated and easier to create, it is important to have effective tools for detecting and preventing them.
- This has led to a growing field of research and development in deep-fake detection, which involves using various techniques from computer vision, machine learning, and deep learning to identify and distinguish between real and fake images or videos.

DEEP-FAKE APPLICATIONS OF LINEAR ALGEBRA.

DEEP FAKE DETECTION- IDENTIFYING THE MOST RELEVANT COMPONENTS.

- Principal Component Analysis (PCA): PCA is a linear algebra technique used to identify patterns and correlations in high-dimensional data.
  - In deep-fake detection, PCA can be used to identify the features that are most important for distinguishing between real and fake images. By reducing the dimensionality of the image data using PCA, it is possible to identify the features that are most informative for detecting deep-fakes.
- Singular value decomposition (SVD): in the case of deep-fake detection, we can use SVD to extract the most important features from an image or video and use them to distinguish between real and fake content.
  - For example, we can apply SVD to a set of facial images or videos and extract the most important facial features, such as eye position, mouth shape, and facial expression. We can then use these features to train a machine learning model to classify new images or videos as real or fake.

DEEP FAKE DETECTION-TRAINING A CLASSIFIER/ IDENTIFIER.

- Convolutional Neural Networks (CNNs): CNNs are a type of deep learning model that are widely used for image recognition and classification tasks.
  - In deep-fake detection, CNNs can be used to identify the features that are most important for distinguishing between real and fake frames. By training a CNN on a large dataset of real and fake frames, it is possible to learn the features that are most indicative of deepfake manipulation.
  - Better for video.
- Generative Adversarial Networks (GANs): GANs are a type of deep learning model that are widely used for image generation and manipulation tasks.
  - In deep-fake detection, GANs can be used to generate fake images that mimic the characteristics of real images. By comparing the generated images to real images, it is possible to identify the features that are most indicative of deepfake manipulation.
  - Better for Image.

THE METHOD AND APPLICATIVE FLOW.

THE STEPS

1. Data preprocessing: The first step is to prepare the image data for analysis. This involves converting the images into a standardized format, such as grayscale, and resizing them to a consistent size. The images are then flattened into a high-dimensional matrix, where each row represents an image and each column represents a pixel.
2. Singular value decomposition (SVD): The next step is to apply SVD to the matrix of flattened images. SVD decomposes the matrix into three components: a left singular matrix, a diagonal matrix, and a right singular matrix. The diagonal matrix contains the singular values, which represent the amount of variation in the data that is explained by each principal component.
3. principal component analysis (PCA): The next step is to perform PCA on the matrix of singular values. PCA identifies the principal components that capture the most variation in the data. By selecting the top principal components, we can reduce the dimensionality of the data while still retaining the most important information.

Alternatively either SVD or PCA can be used as well.

THE STEPS

4.Training a classifier: Once the dimensionality of the data has been reduced using PCA, the next step is to train a classifier on the reduced data. Both CNN and GAN are good options.

- CNN: we have a CNN model that takes in an image and outputs a probability of it being real or fake.
    - The input image is transformed through a series of convolutional and activation layers, resulting in a low-dimensional representation of the image. The weights in each layer are learned through backpropagation, where gradients are calculated using matrix operations such as matrix multiplication and element-wise multiplication.
    - From a linear algebra perspective, CNNs can be viewed as a series of matrix multiplications and nonlinear activations. Each convolutional layer can be represented as a matrix multiplication between the input image and a set of learnable filters, followed by a nonlinear activation function such as ReLU. The output of the layer is then passed to the next layer for further processing

THE STEPS

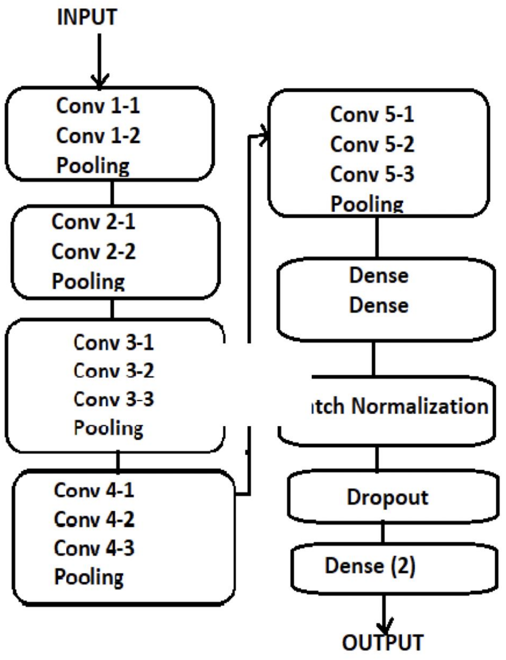


Figure Architecture of the fine-tuned VGG model used

- For example, let's say we have a CNN model that takes in an image and outputs a probability of it being real or fake. The input image is transformed through a series of convolutional and activation layers, resulting in a low-dimensional representation of the image. The weights in each layer are learned through backpropagation, where gradients are calculated using matrix operations such as matrix multiplication and element-wise multiplication.

$$f(t) * g(t) := \underbrace{\int_{-\infty}^{\infty} f(\tau) g(t-\tau)\, d\tau}_{(f*g)(t)},$$

THE STEPS

- GAN: are a type of generative model that can be used to generate realistic images by learning the underlying distribution of a dataset.
  - They consist of two neural networks: a generator and a discriminator. The generator network is trained to generate images that resemble the real images in the dataset, while the discriminator network is trained to distinguish between the real and fake images.
  - From a linear algebra perspective, GANs can be viewed as a game between the generator and the discriminator, where the generator tries to minimize the loss function of the discriminator by generating realistic images, while the discriminator tries to maximize the loss function by correctly classifying the real and fake images.

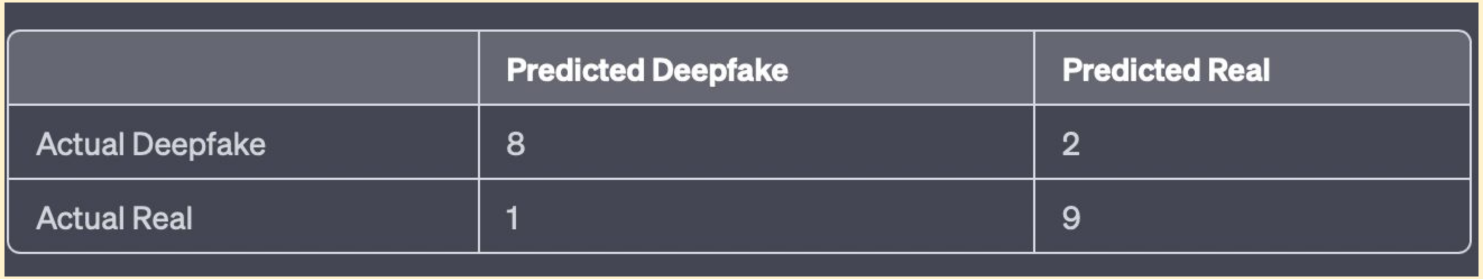
$$\begin{aligned}\min_G \max_D V(D, G) = {} & \mathbb{E}_{x \sim p_{data}(x)}[\log D(x)] \\ & + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))]\end{aligned}$$

THE STEPS

5.Evaluation: The final step is to evaluate the performance of the classifier on the test data. This involves computing metrics such as accuracy, precision, recall, and F1 score, which measure the performance of the classifier in detecting deep-fakes.

This can be represented using a confusion matrix

- A confusion matrix is a table used to evaluate the performance of a classifier by showing the number of true positives, false positives, true negatives, and false negatives.



	Predicted Deepfake	Predicted Real
Actual Deepfake	8	2
Actual Real	1	9

CODING APPROACH.

Importing all the necessary libraries.

```python
import numpy as np
import cv2
import tensorflow as tf
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from keras.preprocessing.image import ImageDataGenerator
from keras.optimizers import Adam
from keras.utils import to_categorical
from keras.layers import LeakyReLU, Dropout, BatchNormalization,
Reshape, UpSampling2D
from keras.layers import Conv2DTranspose, Activation
from keras.models import Model, Input
from sklearn.decomposition import PCA, TruncatedSVD
```

LOADING DATA SET, PREPROCESSING.

```
real_images = []
fake_images = []

# load real images
for i in range(1, 1001):
    img =
cv2.imread("path/to/real/images/real_image_{}.jpg".fo
rmat(i))
    real_images.append(img)

# load fake images
for i in range(1, 1001):
    img =
cv2.imread("path/to/fake/images/fake_image_{}.jpg".fo
rmat(i))
    fake_images.append(img)

# convert lists to numpy arrays
real_images = np.array(real_images)
fake_images = np.array(fake_images)
```

```
real_images = real_images / 255.0

fake_images = fake_images / 255.0




real_images = real_images.reshape(-1,
image_size, image_size, 3)

fake_images = fake_images.reshape(-1,
image_size, image_size, 3)
```

Building data set.

```
real images = []
fake_images = []

# load real images
for i in range(1, 1001):
    img =
cv2.imread("path/to/real/images/real_image_{}.jpg".format(i))
    real_images.append(img)

# load fake images
for i in range(1, 1001):
    img =
cv2.imread("path/to/fake/images/fake_image_{}.jpg".format(i))
    fake_images.append(img)

# convert lists to numpy arrays
real_images = np.array(real_images)
fake_images = np.array(fake_images)
```

Building data set.

```
real images = []
fake_images = []

# load real images
for i in range(1, 1001):
    img =
cv2.imread("path/to/real/images/real_image_{}.jpg".format(i))
    real_images.append(img)

# load fake images
for i in range(1, 1001):
    img =
cv2.imread("path/to/fake/images/fake_image_{}.jpg".format(i))
    fake_images.append(img)

# convert lists to numpy arrays
real_images = np.array(real_images)
fake_images = np.array(fake_images)
```

APPLYING SVD/ PCA

```
#SVD
svd = TruncatedSVD(n_components=50)
X_real = svd.fit_transform(real_images.reshape(len(real_images),
-1))
X_fake = svd.transform(fake_images.reshape(len(fake_images), -1))

#PCA
pca = PCA(n_components=50)
X_real = pca.fit_transform(real_images.reshape(len(real_images),
-1))
X_fake = pca.transform(fake_images.reshape(len(fake_images), -1))
```

APPLYING SVD/ PCA

```
#SVD
svd = TruncatedSVD(n_components=50)
X real = svd.fit_transform(real_images.reshape(len(real_images),
-1))
X_fake = svd.transform(fake_images.reshape(len(fake_images), -1))

#PCA
pca = PCA(n components=50)
X real = pca.fit_transform(real_images.reshape(len(real_images),
-1))
X_fake = pca.transform(fake_images.reshape(len(fake_images), -1))
```

Training CNN

```
# Define the CNN architecture
model = Sequential([
    Conv2D(32, (3, 3), activation='relu',
input_shape=X_real.shape[1:]),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(64, activation='relu'),
    Dropout(0.5),
    Dense(1, activation='sigmoid')
])

# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])

# Train the model
model.fit(X_real, y_real, validation_data=(X_fake, y_fake),
epochs=10)
```

Training GAN

```
from tensorflow.keras.layers import Reshape, Conv2DTranspose
# Define the generator model
generator = Sequential([
    Dense(64 * 7 * 7, activation='relu', input_shape=(latent_dim,)),
    Reshape((7, 7, 64)),
    Conv2DTranspose(64, (3, 3), strides=(2, 2), padding='same', activation='relu'),
    Conv2DTranspose(32, (3, 3), strides=(2, 2), padding='same', activation='relu'),
    Conv2DTranspose(1, (3, 3), activation='sigmoid', padding='same')
])

# Define the discriminator model
discriminator = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(64, activation='relu'),
    Dropout(0.5),
    Dense(1, activation='sigmoid')
])

# Compile the discriminator model
discriminator.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Freeze the weights of the discriminator model during GAN training
discriminator.trainable = False

# Combine the generator and discriminator models into a GAN model
gan = Sequential([
    generator,
    discriminator
])
# Compile the GAN model
gan.compile(optimizer='adam', loss='binary_crossentropy
```

Summary

CNN

The fitted CNN model can be run through with testing data set, and its predictions can be converted into a Confusion matrix.

GAN

The fitted GAN model's discriminator can be run through with testing data set, and its predictions can be converted into a Confusion matrix. The generator can be used to generate synthetic samples after introducing some noise for variation.

SVD Image Compression

Why do we need to compression ?

There is a growing demand for digital multimedia, comprising of images or video frames, the storage and transmission of these images require larger memory space and bandwidth.

We can reduce memory usage, bandwidth size and cost for storing and transporting such files if we can compress the data, such that not much meaningful data is lost.

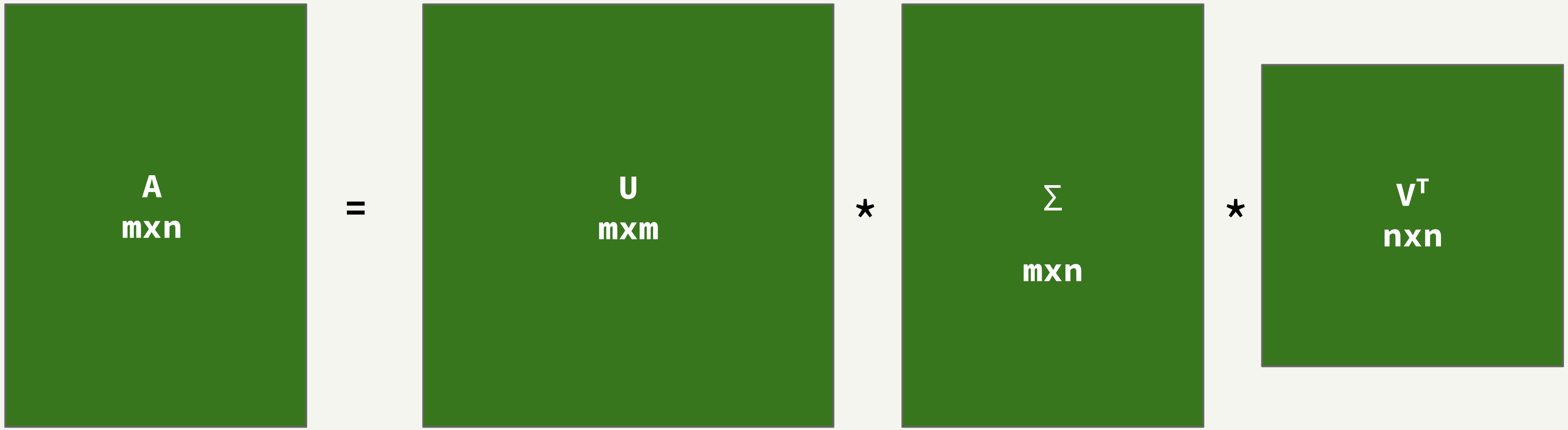
What is SVD?

SVD stands for Singular Value Decomposition, which is a mathematical technique used to decompose a matrix into three parts:

$$U \times \Sigma \times V^T$$

where U and V are orthogonal matrices, and Σ is a diagonal matrix with non-negative real numbers (singular values) on the diagonal.

SVD is a very useful tool in linear algebra, signal processing, and machine learning.


A
mxn
=
U
mxm
*
Σ
mxn
*
Vᵀ
nxn

Calculate U,V and Σ

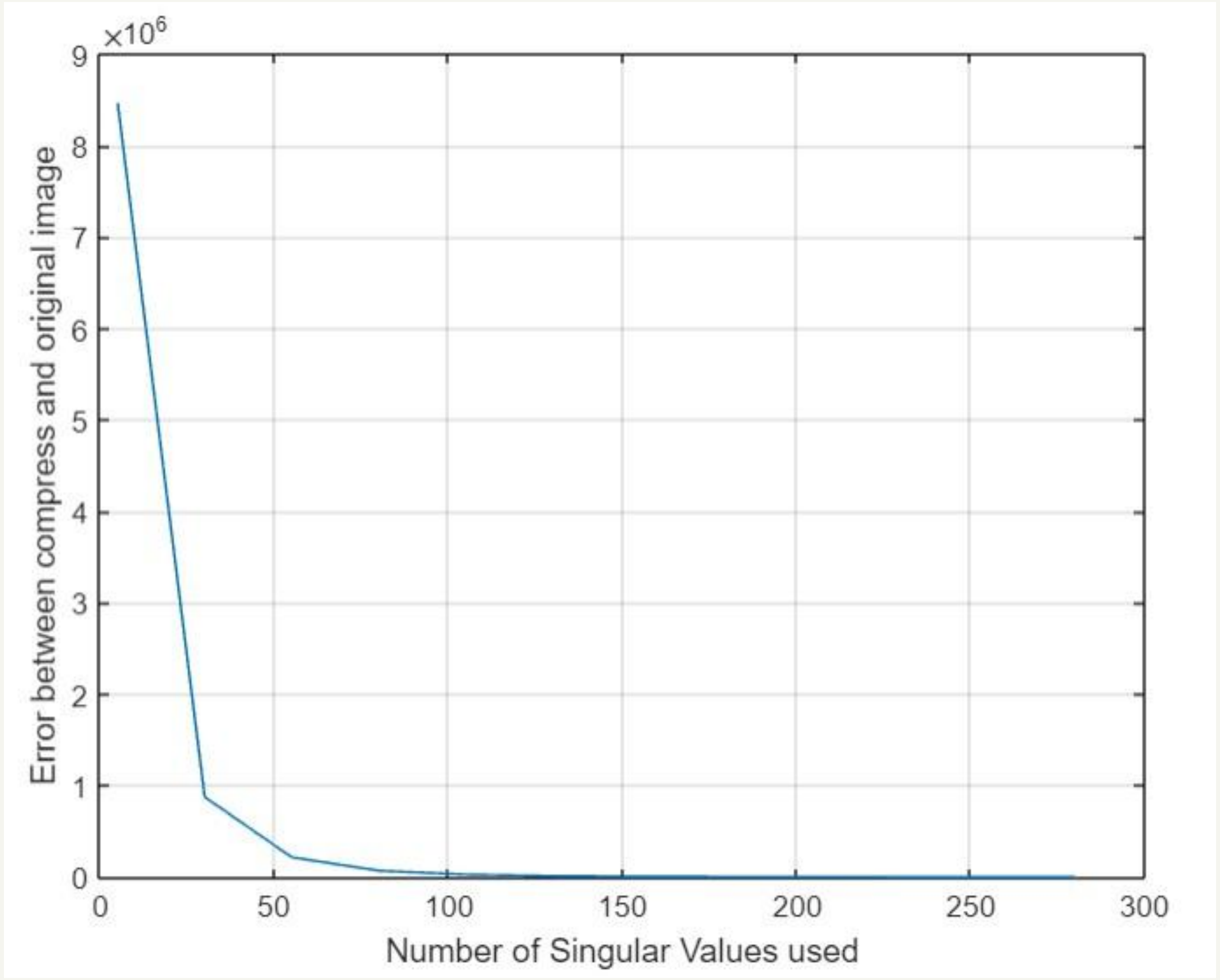
- From a given matrix M, calculate AA^T and A^TA.
- Use AA^T to form U, which is calculated by calculating eigen values and eigen vectors of AA^T.
- In the same way, V can be formed by calculating the eigen values and eigen vectors of A^TA.
- Columns of U and V are formed by dividing each eigen vector by its magnitude.
- Singular values are computed by taking the square root of eigen values. They are arranged in descending order in the diagonal matrix and the corresponding singular vectors in U and V are rearranged accordingly.

How SVD helps in image compression

SVD alone does not help in image compression. After a matrix is decomposed into its 3 component matrices, we start discarding some of the singular values.

The first value of the diagonal matrix contains the most information, and all the further values in the matrix contain decreasing amount of information about the image.

Hence discarding such values reduces the size of the image while avoiding noticeable distortion from the original picture.


×10^6
Error between compress and original image
Number of Singular Values used
0
1
2
3
4
5
6
7
8
9
0
50
100
150
200
250
300

SVD-based image compression works by decomposing an image matrix into its singular values and eigenvectors. The singular values represent the importance of each eigenvector in the matrix, and they can be sorted in descending order to determine which ones to keep and which ones to discard. The eigenvectors are the directions in which the image varies the most, and they can be used to reconstruct the original image with some loss of information.

By discarding the singular values and corresponding eigenvectors that are least important, it is possible to reduce the storage requirements for the image matrix. This is because the compressed image can be represented using a smaller number of singular values and eigenvectors than the original image, resulting in a smaller file size.

The level of compression achieved with SVD-based image compression depends on the number of singular values and eigenvectors kept. Generally, the more singular values and eigenvectors kept, the better the quality of the compressed image, but the larger the file size. Conversely, reducing the number of singular values and eigenvectors leads to greater compression, but also to a loss of image quality.

Code

```
%reading and converting the image

inImage=imread('details.jpg');

inImage=rgb2gray(inImage);

inImageD=double(inImage);

% decomposing the image using singular value decomposition

[U,S,V]=svd(inImageD);

% Using different number of singular values (diagonal of S) to compress and%
reconstruct the image

dispEr = [];

numSVals = [];
```

```
for N=5:25:300% store the singular values in a temporary var
C = S;
% discard the diagonal values not required for compression
C(N+1:end,:)=0;
C(:,N+1:end)=0;
% Construct an Image using the selected singular values
D=U*C*V';
% display
buffer = sprintf('Image output using %d singular values', N)
imshow(uint8(D));
title(buffer);
end
```

Deciding Which Eigen values to discard

The decision of which eigenvalues to discard when compressing an image using SVD depends on the desired level of compression and the acceptable level of image quality loss. Generally, the more eigenvalues that are retained, the better the image quality will be, but the larger the file size will be as well. A common approach to determining the number of eigenvalues to retain is to calculate the total energy of the matrix using the sum of the squares of the singular values. This is called the matrix's total energy, and it represents the amount of information in the matrix. By selecting a threshold percentage of the total energy, we can determine the number of singular values to keep.

Another approach is to use heuristics based on the characteristics of the image being compressed. For example, images with high contrast and sharp edges may require more singular values to be retained to preserve their quality, while images with smooth gradients and low contrast may be more forgiving of lossy compression.

OUTPUT

Image output using 5 singular values

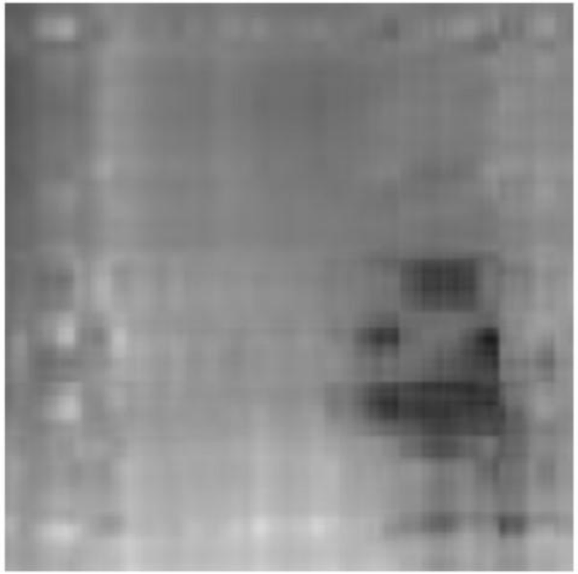


Image output using 155 singular values

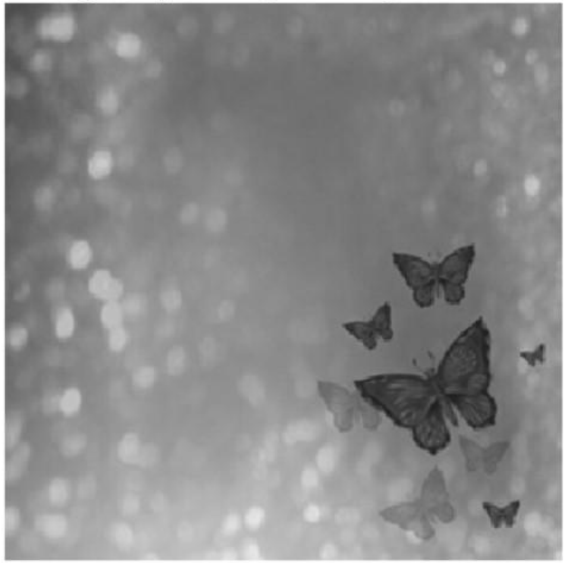
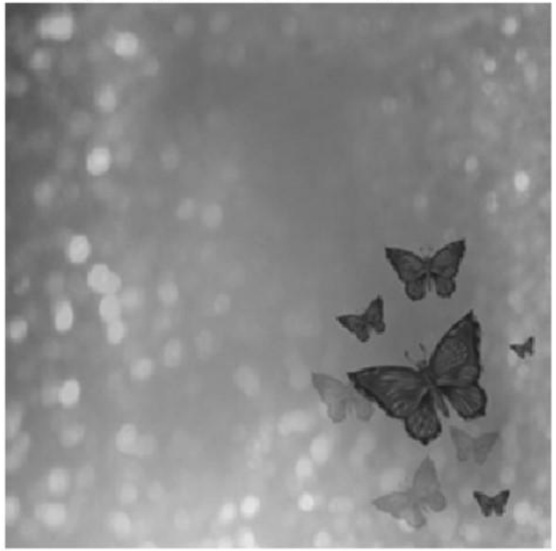
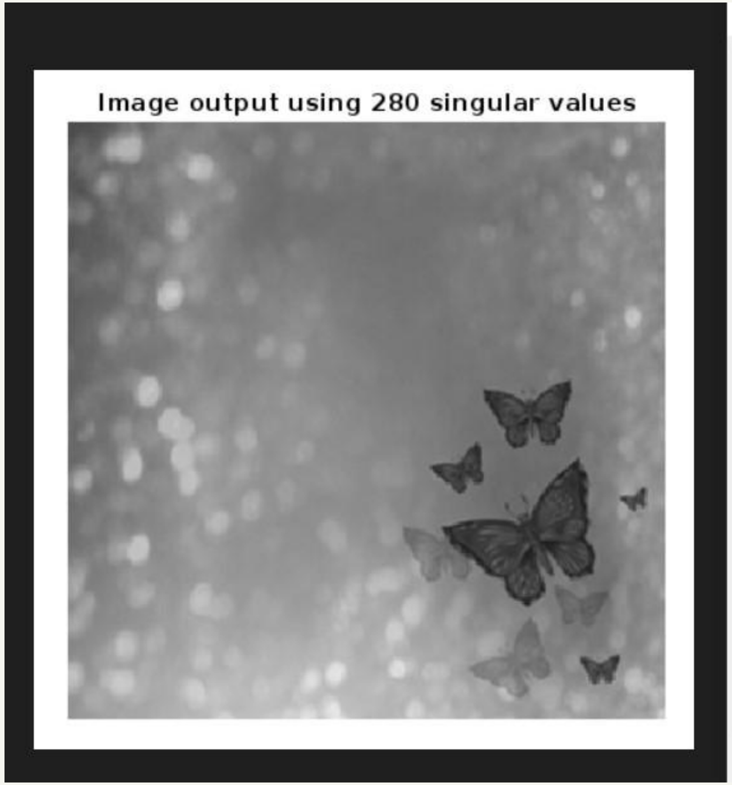
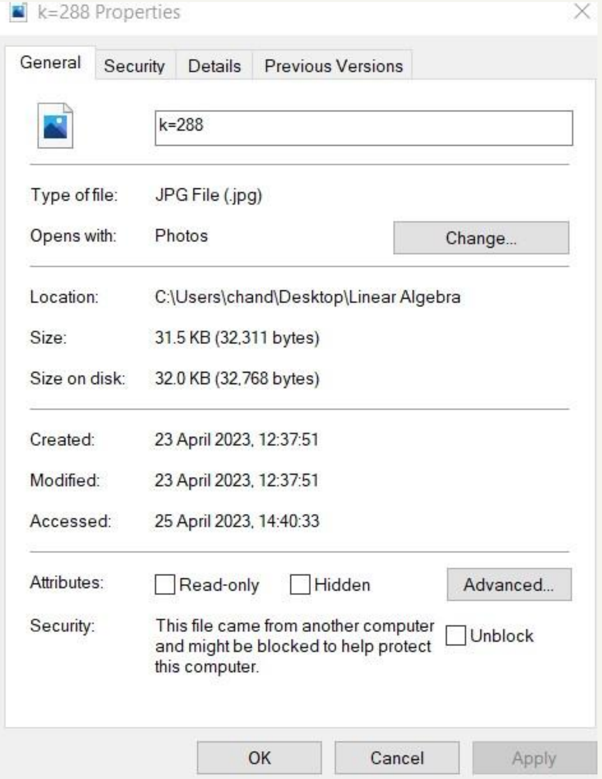


Image output using 280 singular values



OUTPUT




k=288 Properties

General | Security | Details | Previous Versions

k=288

Type of file: JPG File (.jpg)

Opens with: Photos

Location: C:\Users\chand\Desktop\Linear Algebra

Size: 31.5 KB (32,311 bytes)

Size on disk: 32.0 KB (32,768 bytes)

Created: 23 April 2023, 12:37:51

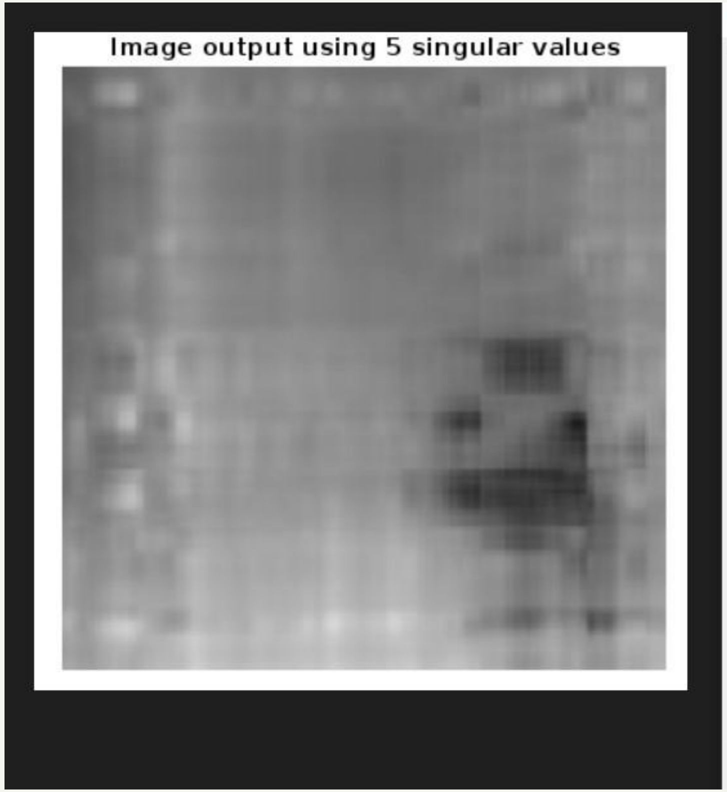
Modified: 23 April 2023, 12:37:51

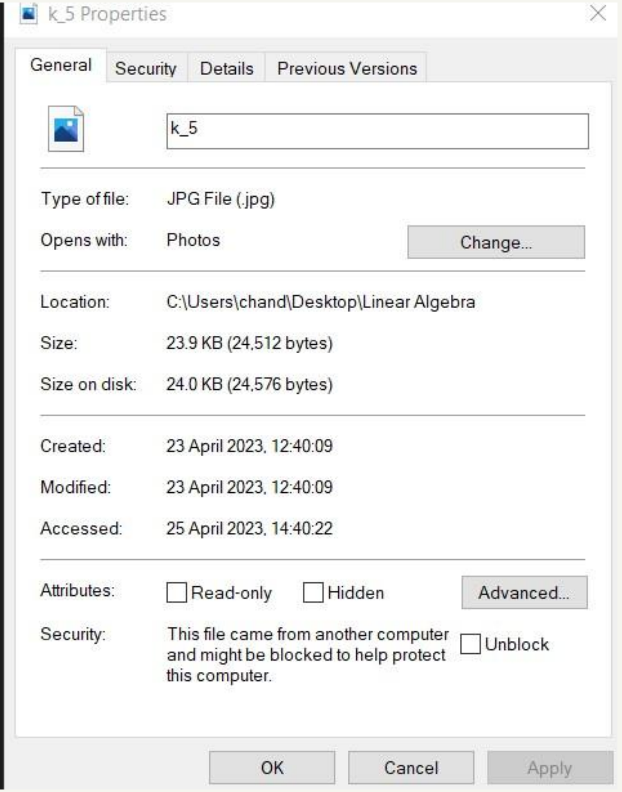
Accessed: 25 April 2023, 14:40:33

Attributes: ☐ Read-only ☐ Hidden

Security: This file came from another computer and might be blocked to help protect this computer. ☐ Unblock

OUTPUT




k_5 Properties

General | Security | Details | Previous Versions

k_5

Type of file: JPG File (.jpg)
Opens with: Photos [Change...]

Location: C:\Users\chand\Desktop\Linear Algebra
Size: 23.9 KB (24,512 bytes)
Size on disk: 24.0 KB (24,576 bytes)

Created: 23 April 2023, 12:40:09
Modified: 23 April 2023, 12:40:09
Accessed: 25 April 2023, 14:40:22

Attributes: ☐ Read-only ☐ Hidden [Advanced...]
Security: This file came from another computer and might be blocked to help protect this computer. ☐ Unblock

OK | Cancel | Apply

Bibliography

https://in.mathworks.com/help/matlab/math/image-compression-with-low-rank-svd.html

http://www.ijmttjournal.org/Volume-65/Issue-8/IJMTT-V65I8P507.pdf

https://stackoverflow.com/questions/13614886/using-svd-to-compress-an-image-in-matlab

https://www.researchgate.net/publication/321479958_Image_compression_using_singular_value_decomposition https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=0c3e11d0cd0a09038981ffe67e2c8ee7ae43581f

https://www.cs.princeton.edu/courses/archive/fall08/cos429/lecture_linear_filters_edge_detection.pdf

https://d1wqtxts1xzle7.cloudfront.net/31159717/Maini-libre.pdf?1392250617=&response-content-disposition=inline%3B+filename%3DStudy_and_comparison_of_various_image_ed.pdf&Expires=1681800864&Signature=K0qmTV7tDbArc4FMfRu3rsKXgxQKEDQKiMy1lRvWZMoVieSx~bdueGgaZoOzZ1YJGMZ~-tHhSEZ0X8hkKL~J1rAKTcI0XanB2uXUgYXowMOtjZz4psg-hSeLQpmTC9eXPFOep9T0CVOd6tqpIGwroxr8IilOUxk0AbYsyNVt1UuWorTukMmZTIFI-w2mzaGHAmT5WtP6nKT3cNtFHq4sSs5GUPvist2Ebua6BaSxIKZ27rgAdcjlyLdpDibKHZTfMESnQCoizQ3aucsP0pKuHLuoNOcxpXOL9RCj3oX4vAsNTMwnCO67N2TSEMM7kbOeIINu3Q1oc0y7DlnxC-wVhA__&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA

https://arxiv.org/abs/1812.04948

https://www.warse.org/IJATCSE/static/pdf/file/ijatcse62922020.pdf

https://proceedings.neurips.cc/paper_files/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf

https://towardsdatascience.com/vgg-neural-networks-the-next-step-after-alexnet-3f91fa9ffe2c#:~:text=VGG%20is%20an%20innovative%20object,most%20used%20image%2Drecognition%20architectures.