# Linear Algebra

Team members:

Chandrachud Sarath- PES1UG21CS150

Dhruv Nilkund- PES1UG21CS178

Darsh Agarwal-PES1UG21S166

## Image segmentation:

```
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));
```

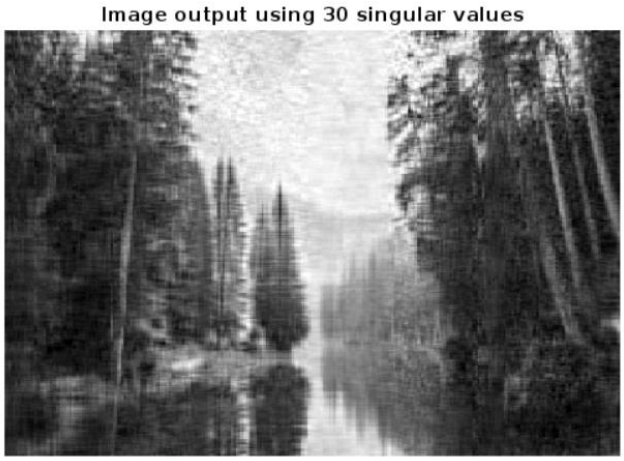
```
L = D - W;

% 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;

subplot(1,2,1);

imshow(img);

title('Original Image');

subplot(1,2,2);

imshow(segmented_img, []);

title(sprintf('Segmented Image (%d segments)', k));
```

**Image compression:**

```
inImage=imread('test1.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
        figure;
        buffer = sprintf('Image output using %d singular values', N)
   imshow(uint8(D));
```

```
    title(buffer);
end
```

## OUTPUT


**Deepfake Detection:**

```
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
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)
real_images = real_images / 255.0
fake_images = fake_images / 255.0
# reshape the images to 4D arrays for CNN
real_images = real_images.reshape(-1, image_size, image_size, 3)
fake_images = fake_images.reshape(-1, image_size, image_size, 3)
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)
#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))
# 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)
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
```