**DIP THEORY**

**Practical 1**

**🔍 Aim:**

To perform **2D Linear Convolution** and **2D Circular Convolution** between two 2D matrices.

**✅ PART 1: 2D LINEAR CONVOLUTION**

**🔢 Example 1:**

matlab

CopyEdit

clc;

pramod37\_x = [4,5,6;7,8,9];

pramod37\_h = [1;1;1];

disp(pramod37\_x,"pramod37\_x=")

disp(pramod37\_h,"pramod37\_h=")

pramod37\_y = conv2(pramod37\_x, pramod37\_h);

disp(pramod37\_y,"2D Linear Convolution result: y=")

**🧠 Explanation:**

* pramod37\_x is a **2x3** matrix (image or signal).
* pramod37\_h is a **3x1** matrix (kernel/filter).
* conv2(x,h) performs 2D convolution.
* It **slides the filter** over the matrix and performs multiplication + summation (dot product).
* The output will be **size: (2+3-1) x (3+1-1) = 4x3**.
* This is also called **full convolution**.

**🔢 Example 2:**

matlab

CopyEdit

clc;

pramod37\_x = [1,2,3;4,5,6;7,8,9];

pramod37\_h = [1,1;1,1;1,1];

disp(pramod37\_x,"pramod37\_x=")

disp(pramod37\_h,"pramod37\_h=")

pramod37\_y = conv2(pramod37\_x, pramod37\_h);

disp(pramod37\_y,"2D Linear Convolution result: y=")

* pramod37\_x is 3x3, and pramod37\_h is 3x2.
* Output size will be: (3+3−1) x (3+2−1) = 5x4

**✅ PART 2: 2D CIRCULAR CONVOLUTION**

**📌 Theory:**

* **Circular convolution** uses the **Discrete Fourier Transform (DFT)**.
* Convolution in **time/spatial domain** is equivalent to **multiplication in frequency domain**.
* In MATLAB, we use fft2 for 2D FFT (Fast Fourier Transform), and ifft2 for inverse.

**🔢 Example 1:**

matlab

CopyEdit

clc;

pramod37\_x = [1,2;3,4];

pramod37\_h = [5,6;7,8];

disp(pramod37\_x,"pramod37\_x=")

disp(pramod37\_h,"pramod37\_h=")

pramod37\_X = fft2(pramod37\_x);

pramod37\_H = fft2(pramod37\_h);

Y = pramod37\_X .\* pramod37\_H;

pramod37\_y = ifft2(Y);

disp(pramod37\_y,"2D Circular Convolution result: y=")

**🧠 Explanation:**

1. fft2(x) → converts to frequency domain.
2. Multiply both transforms.
3. ifft2(Y) → convert back to spatial domain.
4. This gives **circular convolution result** (size is same as input).

**🔢 Example 2:**

matlab

CopyEdit

clc;

pramod37\_x = [1,2,3;4,5,6;7,8,9];

pramod37\_h = [1,1,1;1,1,1;1,1,1];

...

* Again, same steps as above but with 3x3 matrices.
* Result is also 3x3 due to circular nature.

**👨‍🏫 How to Explain to Invigilator:**

*“In this practical, we demonstrate two types of 2D convolution operations. First,* ***linear convolution****, which gives a bigger output by sliding the filter across the matrix. Second,* ***circular convolution****, which uses Fourier Transform where convolution becomes multiplication in frequency domain, and the result size is same as input. We use conv2() for linear, and fft2() with ifft2() for circular convolution.”*

**🎓 VIVA QUESTIONS WITH ANSWERS:**

| **Question** | **Answer** |
| --- | --- |
| What is convolution? | Convolution is a mathematical operation used to combine two signals or functions. In image processing, it's used to apply filters to images. |
| Difference between linear and circular convolution? | Linear convolution considers the full overlap, leading to larger output. Circular convolution wraps around the edges and the output size is same as input. |
| What does conv2() do in MATLAB? | It performs 2D linear convolution between two matrices. |
| What is fft2() and ifft2()? | fft2() computes the 2D Fourier transform, and ifft2() computes its inverse. They’re used in circular convolution. |
| Why is convolution used in image processing? | It's used for filtering, blurring, edge detection, sharpening, etc. |
| What is the output size of 2D linear convolution? | If input size is m×n and filter size is p×q, output size is (m+p-1) × (n+q-1). |
| Which is faster: linear or circular convolution? | Circular convolution via FFT is usually faster for large matrices. |
| Can convolution be applied on color images? | Yes, but usually it's applied channel-wise on R, G, and B separately. |

Practical 2

**Aim**

To **express Circular Convolution** as **Linear Convolution + Aliasing** using 2D matrices in MATLAB.

**🧠 Concept Overview**

**🔁 Circular Convolution:**

* Circular convolution wraps around the matrix edges.
* It is **periodic** and results in **output size = size of input**.

**➕ Linear Convolution + Aliasing:**

* When performing **linear convolution**, we get a **larger output**.
* But circular convolution assumes **periodic extension**, so extra rows/columns from linear convolution **overlap back (aliasing)** into the original matrix size.

👉 This practical **demonstrates** how circular convolution can be simulated by:

1. Performing linear convolution.
2. Folding (aliasing) the overflow parts back into the original size.

**🧪 MATLAB Code Explained**

**🔢 Code:**

matlab

CopyEdit

clc;

pramod\_x = [1,2;3,4]; % Input matrix

pramod\_h = [5,6;7,8]; % Filter matrix

pramod\_y = conv2(pramod\_x, pramod\_h); % 2D Linear Convolution (gives 3x3 matrix)

% Wrap-around (aliasing) on columns

pramod\_y1 = [pramod\_y(:,1) + pramod\_y(:,3), pramod\_y(:,2)];

% Wrap-around (aliasing) on rows

pramod\_y2 = [pramod\_y1(1,:) + pramod\_y1(3,:); pramod\_y1(2,:)];

% Display results

disp(pramod\_y, "Pramod Joshi 248637 Linear Convolution Result: y=");

disp(pramod\_y2, "Circular Convolution expressed as Linear Convolution=");

**📊 Step-by-step Explanation:**

**1. Linear Convolution:**

You get a **3×3 matrix** from conv2() since:

* Input size: 2×2
* Filter size: 2×2
* Output size: (2+2−1) × (2+2−1) = 3×3

Let’s say output is:

ini

CopyEdit

y = [5 16 12;

22 60 40;

21 52 32];

**2. Aliasing Columns:**

matlab

CopyEdit

pramod\_y1 = [pramod\_y(:,1) + pramod\_y(:,3), pramod\_y(:,2)];

This folds the **1st and 3rd columns** together (wrap around) → simulating periodic boundary.

**3. Aliasing Rows:**

matlab

CopyEdit

pramod\_y2 = [pramod\_y1(1,:) + pramod\_y1(3,:); pramod\_y1(2,:)];

This adds the **1st and 3rd rows** together → simulating circular convolution.

**4. Result:**

Final matrix pramod\_y2 is of size **2x2**, which is the same size as the inputs.  
This simulates **circular convolution** using linear convolution + aliasing.

**📘 How to Explain to Invigilator:**

*"In this practical, we take two 2×2 matrices and perform 2D linear convolution using conv2(). This gives a 3×3 output. Since circular convolution must have the same size as the inputs (2×2), we simulate circular convolution by folding the extra rows and columns back into the result using aliasing. This shows that circular convolution can be expressed as linear convolution plus aliasing (wrapping around overflow terms)."*

**🎓 Viva Questions with Answers**

| **Question** | **Answer** |
| --- | --- |
| What is aliasing in this context? | Aliasing here means wrapping around overflow rows/columns from linear convolution back into the original matrix size to simulate circular convolution. |
| What is the size of output in 2D linear convolution? | If input is m×n and filter is p×q, output = (m+p−1) × (n+q−1). |
| What is the output size of circular convolution? | Same as the size of the input matrices. |
| How is circular convolution related to linear convolution? | Circular convolution is linear convolution with aliasing (wrapping of overflow terms). |
| What is the use of circular convolution in image processing? | It's used in **frequency domain filtering** and **FFT-based processing**, where size consistency is important. |
| Which function in MATLAB gives 2D linear convolution? | conv2() |
| Which function performs circular convolution? | Circular convolution is performed using FFT: ifft2(fft2(x) .\* fft2(h)) |
| Why do we add first and last rows/columns in aliasing? | Because circular convolution assumes the input is periodic, so the overflow parts wrap around to the beginning. |

Practical 3

**🔁 What is Correlation?**

Correlation is a measure of **similarity** between two signals (or matrices).

* **Cross-Correlation** → Compares **two** different signals.
* **Auto-Correlation** → Compares a signal with **itself**.
* **Convolution vs Correlation** → Correlation is like convolution **without flipping** the kernel (but in MATLAB, we simulate correlation using convolution **after flipping** the kernel manually).

**🔷 A) Linear Cross Correlation**

**🔢 Code:**

matlab

CopyEdit

clc;

pramod\_x = [3,1;2,4]; % Input matrix

pramod\_h1 = [1,5;2,3]; % Correlation kernel

% Flip the kernel both horizontally and vertically to simulate correlation using convolution

pramod\_h2 = pramod\_h1(:, end:-1:1);

pramod\_h = pramod\_h2(end:-1:1, :);

pramod\_y = conv2(pramod\_x, pramod\_h); % Perform linear correlation

disp(pramod\_y, "linear Cross Correlation result y=");

**📘 Explanation:**

* conv2() performs convolution, but for **correlation**, we **flip the filter (kernel)** before applying it.
* (:, end:-1:1) flips columns (horizontal flip), and end:-1:1, : flips rows (vertical flip).
* Output shows how well pramod\_x and pramod\_h1 match at different positions.

**🔷 B) Circular Cross Correlation**

**🔢 Code:**

matlab

CopyEdit

clc;

pramod\_x = [1,5;2,4];

pramod\_h = [3,2;4,1];

% Flip the kernel

pramod\_h = pramod\_h(:, end:-1:1);

pramod\_h = pramod\_h(end:-1:1, :);

% Apply FFT-based circular correlation

pramod\_X = fft2(pramod\_x);

pramod\_H = fft2(pramod\_h);

pramod\_Y = pramod\_X .\* pramod\_H;

pramod\_y = ifft2(pramod\_Y);

disp(pramod\_y, "Pramod 248637 Circular Correlation result y=");

**📘 Explanation:**

* Uses **2D FFT** to convert both input and flipped kernel into frequency domain.
* Multiplies them and applies ifft2 to get result.
* Because it's circular, size remains **same as input**.
* You simulate correlation by flipping pramod\_h before FFT.

**🔷 C) Linear Auto Correlation**

**🔢 Code:**

matlab

CopyEdit

clc;

pramod\_x1 = [1,1;1,1];

% Flip the same matrix to auto-correlate

pramod\_x2 = pramod\_x1(:, end:-1:1);

pramod\_x2 = pramod\_x2(end:-1:1, :);

pramod\_x = conv2(pramod\_x1, pramod\_x2); % Perform linear auto-correlation

disp(pramod\_x, "Pramod 248637 linear auto Correlation x=");

**📘 Explanation:**

* Auto-correlation means comparing the signal **with itself**.
* The second version is flipped to simulate correlation.
* Output gives a measure of how much a signal correlates with a shifted version of itself.

**🔷 D) Linear Cross Correlation (Another Example)**

**🔢 Code:**

matlab

CopyEdit

clc;

pramod\_x = [1,1;1,1];

pramod\_h1 = [1,2;3,4];

pramod\_h2 = pramod\_h1(:, end:-1:1);

pramod\_h = pramod\_h2(end:-1:1, :);

pramod\_y = conv2(pramod\_x, pramod\_h);

disp(pramod\_y, "Pramod 248637 linear Cross Correlation result y=");

**📘 Explanation:**

* Another example of cross correlation with different kernel.
* Repeats the flipping and convolution logic to simulate correlation.

**🧑‍🏫 How to Explain to Invigilator:**

*"In this practical, I implemented correlation techniques. Cross correlation compares similarity between two matrices, while auto correlation compares a matrix with itself. For linear correlation, I simulate correlation using conv2 after flipping the filter matrix both horizontally and vertically. For circular correlation, I use FFT (fft2) and inverse FFT (ifft2) to perform it efficiently. These operations help in pattern matching, template detection, and feature comparison in image processing."*

**🎓 Viva Questions & Answers**

| **Question** | **Answer** |
| --- | --- |
| What is correlation? | Correlation measures the similarity between two signals. |
| What is the difference between cross and auto correlation? | Cross correlation is between two different signals; auto correlation is with itself. |
| What is the difference between convolution and correlation? | Convolution flips the kernel; correlation does not (but to simulate with conv2, we flip manually). |
| What does conv2() do? | It performs 2D convolution. We flip the kernel to use it for correlation. |
| Why do we flip the kernel in correlation? | We simulate correlation using convolution by flipping both rows and columns of the kernel. |
| What is circular correlation? | It assumes the signal is periodic and wraps around the edges. |
| What functions are used for circular correlation? | fft2() for FFT, ifft2() for inverse FFT, and element-wise multiplication. |
| What's the size of output in linear vs circular correlation? | Linear: size increases (m+p-1, n+q-1), Circular: same size as input. |
| What is the purpose of correlation in image processing? | It's used for **template matching**, **feature detection**, and **measuring similarity**. |

**🔁 What is Correlation?**

Correlation is a measure of **similarity** between two signals (or matrices).

* **Cross-Correlation** → Compares **two** different signals.
* **Auto-Correlation** → Compares a signal with **itself**.
* **Convolution vs Correlation** → Correlation is like convolution **without flipping** the kernel (but in MATLAB, we simulate correlation using convolution **after flipping** the kernel manually).

**🔷 A) Linear Cross Correlation**

**🔢 Code:**

matlab

CopyEdit

clc;

pramod\_x = [3,1;2,4]; % Input matrix

pramod\_h1 = [1,5;2,3]; % Correlation kernel

% Flip the kernel both horizontally and vertically to simulate correlation using convolution

pramod\_h2 = pramod\_h1(:, end:-1:1);

pramod\_h = pramod\_h2(end:-1:1, :);

pramod\_y = conv2(pramod\_x, pramod\_h); % Perform linear correlation

disp(pramod\_y, "linear Cross Correlation result y=");

**📘 Explanation:**

* conv2() performs convolution, but for **correlation**, we **flip the filter (kernel)** before applying it.
* (:, end:-1:1) flips columns (horizontal flip), and end:-1:1, : flips rows (vertical flip).
* Output shows how well pramod\_x and pramod\_h1 match at different positions.

**🔷 B) Circular Cross Correlation**

**🔢 Code:**

matlab

CopyEdit

clc;

pramod\_x = [1,5;2,4];

pramod\_h = [3,2;4,1];

% Flip the kernel

pramod\_h = pramod\_h(:, end:-1:1);

pramod\_h = pramod\_h(end:-1:1, :);

% Apply FFT-based circular correlation

pramod\_X = fft2(pramod\_x);

pramod\_H = fft2(pramod\_h);

pramod\_Y = pramod\_X .\* pramod\_H;

pramod\_y = ifft2(pramod\_Y);

disp(pramod\_y, "Pramod 248637 Circular Correlation result y=");

**📘 Explanation:**

* Uses **2D FFT** to convert both input and flipped kernel into frequency domain.
* Multiplies them and applies ifft2 to get result.
* Because it's circular, size remains **same as input**.
* You simulate correlation by flipping pramod\_h before FFT.

**🔷 C) Linear Auto Correlation**

**🔢 Code:**

matlab

CopyEdit

clc;

pramod\_x1 = [1,1;1,1];

% Flip the same matrix to auto-correlate

pramod\_x2 = pramod\_x1(:, end:-1:1);

pramod\_x2 = pramod\_x2(end:-1:1, :);

pramod\_x = conv2(pramod\_x1, pramod\_x2); % Perform linear auto-correlation

disp(pramod\_x, "Pramod 248637 linear auto Correlation x=");

**📘 Explanation:**

* Auto-correlation means comparing the signal **with itself**.
* The second version is flipped to simulate correlation.
* Output gives a measure of how much a signal correlates with a shifted version of itself.

**🔷 D) Linear Cross Correlation (Another Example)**

**🔢 Code:**

matlab

CopyEdit

clc;

pramod\_x = [1,1;1,1];

pramod\_h1 = [1,2;3,4];

pramod\_h2 = pramod\_h1(:, end:-1:1);

pramod\_h = pramod\_h2(end:-1:1, :);

pramod\_y = conv2(pramod\_x, pramod\_h);

disp(pramod\_y, "Pramod 248637 linear Cross Correlation result y=");

**📘 Explanation:**

* Another example of cross correlation with different kernel.
* Repeats the flipping and convolution logic to simulate correlation.

**🧑‍🏫 How to Explain to Invigilator:**

*"In this practical, I implemented correlation techniques. Cross correlation compares similarity between two matrices, while auto correlation compares a matrix with itself. For linear correlation, I simulate correlation using conv2 after flipping the filter matrix both horizontally and vertically. For circular correlation, I use FFT (fft2) and inverse FFT (ifft2) to perform it efficiently. These operations help in pattern matching, template detection, and feature comparison in image processing."*

**🎓 Viva Questions & Answers**

| **Question** | **Answer** |
| --- | --- |
| What is correlation? | Correlation measures the similarity between two signals. |
| What is the difference between cross and auto correlation? | Cross correlation is between two different signals; auto correlation is with itself. |
| What is the difference between convolution and correlation? | Convolution flips the kernel; correlation does not (but to simulate with conv2, we flip manually). |
| What does conv2() do? | It performs 2D convolution. We flip the kernel to use it for correlation. |
| Why do we flip the kernel in correlation? | We simulate correlation using convolution by flipping both rows and columns of the kernel. |
| What is circular correlation? | It assumes the signal is periodic and wraps around the edges. |
| What functions are used for circular correlation? | fft2() for FFT, ifft2() for inverse FFT, and element-wise multiplication. |
| What's the size of output in linear vs circular correlation? | Linear: size increases (m+p-1, n+q-1), Circular: same size as input. |
| What is the purpose of correlation in image processing? | It's used for **template matching**, **feature detection**, and **measuring similarity**. |

Prac4

**✅ Aim:**

To perform the **Discrete Fourier Transform (DFT)** of a **4x4 grayscale image** using MATLAB.

**🔢 Code:**

matlab

CopyEdit

clc;

x = [1,1,1,1;

1,1,1,1;

1,1,1,1;

1,1,1,1];

X = fft2(x, -1); % Perform 2D DFT

disp(X, "X[k]=");

🔁 You may have a typo in your code. It should be fft2(x) instead of fft(x,-1) for 2D DFT on an image.

**🧠 Theory Behind the Code**

**📘 What is DFT?**

* The **Discrete Fourier Transform (DFT)** converts a signal (or image) from the **spatial domain** to the **frequency domain**.
* In the **frequency domain**, we can analyze **patterns, periodicity, and frequency content** of the image.

**🖼️ In context of an image:**

* Each pixel represents **intensity** in the spatial domain.
* Applying DFT helps us see how the image is composed of **different frequency components** (low, mid, high).

**⚙️ Explanation of Code**

1. **x = [1,1,1,1; ... ]**
   * This is a **4×4 grayscale image**, where all pixel values are **1** (uniform image).
2. **X = fft2(x)**
   * fft2 is used for **2D DFT**, commonly used in image processing.
   * It converts the 2D image into its **frequency representation**.
3. **disp(X)**
   * Displays the output of DFT.
   * Since all pixels are the same (1), the result will show:
     + One **non-zero frequency component** (DC component), and
     + All other frequency components as **zero**.

**🧪 Expected Output**

For a 4x4 matrix filled with 1s, the output of fft2(x) will be:

matlab

CopyEdit

X =

16 0 0 0

0 0 0 0

0 0 0 0

0 0 0 0

**✨ Why?**

* Because the image is **completely uniform**.
* So the only frequency present is the **DC component** (X(1,1) = 16, which is the sum of all pixel values).
* All other frequencies are zero (no changes in intensity across rows or columns).

**👨‍🏫 How to Explain to Invigilator:**

*"In this practical, we create a 4x4 grayscale image with all pixels having value 1. We apply 2D DFT using the fft2() function. Since the image is uniform, its frequency spectrum will contain only the DC component, which is the total energy of the image (1×16 = 16). The rest of the frequencies will be zero because there are no changes or patterns in the image. This shows that DFT captures how fast intensities change across an image."*

**🎓 Viva Questions with Answers**

| **Question** | **Answer** |
| --- | --- |
| What is DFT? | DFT stands for Discrete Fourier Transform. It converts a signal/image from the time/spatial domain into the frequency domain. |
| Why use DFT in image processing? | To analyze the frequency content of an image, perform filtering, compression, and feature extraction. |
| What is the difference between fft() and fft2()? | fft() is for 1D signals, while fft2() is for 2D matrices like images. |
| What does fft2(x) return? | It returns the 2D Fourier transform of matrix x, showing frequency components. |
| What is the value of X(1,1) in DFT? | It is the **DC component**, which represents the average or total intensity (sum of all pixels). |
| What is the output of fft2() for a uniform image? | The DC component is non-zero; all other frequency components are zero. |
| What is the inverse of DFT? | Inverse DFT (IDFT) is used to convert data back to the spatial domain. In MATLAB, use ifft2(). |
| Is DFT output complex? | Yes, the DFT output is usually complex, containing both magnitude and phase. |
| How do you visualize DFT of an image? | Use abs(fft2(x)) to get magnitude spectrum and angle(fft2(x)) for phase. |

Prac5

**✅ Aim:**

**To compute the Discrete Cosine Transform (DCT)** and perform the **Karhunen–Loève Transform (KL Transform)** for a given 2D matrix.

**🔢 MATLAB Code Explanation:**

matlab

CopyEdit

clear;

clc;

X = [4,3,5,6;

4,2,7,7;

5,5,6,7];

[m,n] = size(X);

A = [];

E = [];

**🧠 Step-by-step Logic:**

**📌 Step 1: Compute Mean and Covariance Matrix**

matlab

CopyEdit

for i = 1:n

A = A + X(:,i); % Sum of each column vector (for mean)

E = E + X(:,i) \* X(:,i)'; % Outer product for Covariance calculation

end

mx = A / n; % Mean vector

E = E / n; % Mean of the outer products

C = E - mx \* mx'; % Covariance Matrix

* A accumulates all columns → to compute **mean**.
* E accumulates **outer products** → to compute covariance.
* C is the **covariance matrix** of the data.

**📌 Step 2: Compute Eigenvalues and Eigenvectors**

matlab

CopyEdit

[V,D] = spec(C); % Eigen decomposition

d = diag(D); % Eigenvalues

[d, i] = gsort(d); % Sort eigenvalues descending

* spec(C) gives eigenvalues D and eigenvectors V.
* gsort(d) sorts eigenvalues from largest to smallest (for energy order).

**📌 Step 3: Sort Eigenvectors Accordingly**

matlab

CopyEdit

for j = 1:length(d)

T(:,j) = V(:,i(j)); % Sorted Eigenvectors

end

T = T';

* T becomes the **transformation matrix** (eigenvectors arranged by importance).
* Each row of T is a **basis vector** in KL space.

**📌 Step 4: KL Transformation (Projection)**

matlab

CopyEdit

for i = 1:n

Y(:,i) = T \* X(:,i); % Project X onto KL basis

end

disp(Y, 'KL transformation of the input matrix Y=')

* Projects original data onto **KL basis** → this is the **KL Transformed Data** Y.

**📌 Step 5: Reconstruction**

matlab

CopyEdit

for i = 1:n

x(:,i) = T' \* Y(:,i); % Reconstruct original data

end

disp(x, 'Reconstructed matrix of the given sample matrix X=')

* By multiplying with T', you can reconstruct the original matrix X.

**👨‍🏫 How to Explain to Invigilator**

*"In this practical, we perform the KL transform (also known as PCA for 2D matrices). We start with a 3×4 matrix X. First, we compute the mean and the covariance matrix. Then we find the eigenvalues and eigenvectors of the covariance matrix, which give us the KL transform basis. We sort the eigenvectors based on eigenvalues and use them as our transformation matrix T. The KL transform of X is obtained by projecting X onto these basis vectors. Finally, we reconstruct the original matrix using the inverse transform. This is useful in image compression and pattern recognition, where we want to reduce redundancy and keep important information."*

**🎓 Viva Questions with Answers**

| **Question** | **Answer** |
| --- | --- |
| What is the KL Transform? | KL Transform is a linear transform that converts data to a space where variables are uncorrelated. It’s like PCA. |
| What is the goal of KL Transform? | To reduce data dimensionality while preserving the most significant features or patterns in the data. |
| How is KL Transform computed? | By finding eigenvalues and eigenvectors of the covariance matrix of the data. |
| What is the role of eigenvectors? | Eigenvectors define the new axis or basis directions for the transformed data. |
| Why do we sort eigenvalues? | Sorting eigenvalues helps in choosing the directions with the most information (energy). |
| What is the benefit of KL Transform in image processing? | It helps in **compression**, **feature extraction**, and **noise reduction**. |
| Is KL Transform reversible? | Yes, if all eigenvectors are used, the transformation is reversible (we can reconstruct the original matrix). |
| What is covariance matrix? | It shows the linear relationship (correlation) between all pairs of features (or signals). |
| What if we use only top 1 or 2 eigenvectors? | It results in data compression by reducing dimensions but may lose some information. |
| How is KL Transform different from DFT/DCT? | DFT/DCT are **fixed basis transforms**, while KL Transform has a **data-dependent basis** calculated using covariance. |

Prac6

**✅ Aim:**

Perform **Brightness Enhancement**, **Contrast Manipulation**, and generate the **Negative of an Image** using image processing techniques.

**🔍 1. Brightness Enhancement**

**🧠 Concept:**

Brightness is related to how light or dark an image is. By increasing or decreasing the intensity values of each pixel, we can brighten or darken an image.

**⚙️ What Happens in the Code:**

* The image is first converted to grayscale.
* Brightness is increased by **adding a constant value (e.g., 50)** to all pixel intensities.
* A darker version is also created by **subtracting the same constant**.

**✅ What to Explain to Invigilator:**

"We enhance the brightness by increasing each pixel’s intensity by 50. This makes the image appear lighter. Similarly, decreasing intensity darkens the image. We ensure the result stays in valid image format by converting it back to uint8 after modification."

**🎯 2. Contrast Manipulation**

**🧠 Concept:**

Contrast refers to the **difference between the darkest and brightest parts** of the image. Higher contrast makes features more distinct, while lower contrast makes it look dull.

**⚙️ What Happens in the Code:**

* Pixel values are **multiplied**:
  + 0.5 × pixel value → **Decreases contrast**
  + 2 × pixel value (after reduction) → **Increases contrast**

**✅ What to Explain to Invigilator:**

"In this part, we modify the contrast by scaling the pixel intensities. Reducing all pixel values compresses the intensity range (low contrast), while stretching them back increases it. This simulates how image contrast can be controlled."

**🌓 3. Image Negative**

**🧠 Concept:**

Negative images invert brightness – **dark areas become light** and **light areas become dark**. This is useful in **medical imaging** and **feature detection**.

**⚙️ What Happens in the Code:**

* Every pixel intensity is **subtracted from 255** (the maximum for 8-bit images).
* This flips the brightness scale:  
  New Pixel = 255 - Old Pixel

**✅ What to Explain to Invigilator:**

"To create the negative, we subtract each pixel from 255. This inverts the image – dark regions become bright and vice versa. This technique is helpful in highlighting features that are not easily visible in the original image."

**🎓 Viva Questions and Answers**

| **Viva Question** | **Answer** |
| --- | --- |
| What is brightness in an image? | Brightness refers to how light or dark an image is, controlled by pixel intensity values. |
| How is brightness increased or decreased? | By adding or subtracting a constant value to/from each pixel’s intensity. |
| What is contrast? | Contrast is the difference in intensity between the darkest and brightest parts of the image. |
| How do you increase contrast? | By multiplying pixel values to stretch the intensity range. |
| What happens if pixel values go beyond 255? | They are clipped back to 255 if not handled properly. That's why we convert back to uint8. |
| What is the purpose of image negative? | To invert the brightness levels, often used in X-rays and other medical images for better analysis. |
| Why convert images to grayscale first? | Grayscale simplifies processing by reducing the image to one channel of intensity values. |
| What is the pixel range of grayscale image? | 0 to 255 for 8-bit images. |
| What is the datatype of images in Scilab/Matlab? | Typically uint8 for standard images, which stores values from 0 to 255. |
| Why use double() before operations? | To avoid overflow/underflow during mathematical operations. |

Prac7

**✅ Aim:**

To perform **Thresholding Operation** and **Gray Level Slicing** on a grayscale image using Scilab.

**🔍 1. Threshold Operation**

**🧠 Concept:**

Thresholding is a **segmentation technique** used to convert a grayscale image into a **binary image**. It is widely used in separating objects from the background.

**⚙️ What Happens in the Code:**

1. The input image is **converted to grayscale**.
2. A **threshold value t is taken as input**.
3. The program checks each pixel:
   * If pixel intensity < t → set to 0 (black).
   * Else → set to 255 (white).
4. The result is a **binary image** highlighting areas above and below the threshold.

**✅ What to Say to Invigilator:**

"In thresholding, we compare each pixel with a threshold value. If it's less than the threshold, we set it to black (0), otherwise to white (255). This creates a binary image, helping in object detection, like identifying text or shapes."

**🎯 2. Gray Level Slicing**

**🧠 Concept:**

Gray level slicing is used to **highlight a specific range** of intensities (gray levels) in an image. It’s useful in medical and satellite images where you want to enhance certain features.

**⚙️ What Happens in the Code:**

1. The image is converted to grayscale and typecast to double for processing.
2. The maximum pixel intensity L is calculated.
3. A slicing range is selected from a = L/2 to b = L.
4. Each pixel is checked:
   * If it falls within [a, b], set to **maximum intensity (highlighted)**.
   * Else, set to **0 (black)**.
5. The resulting image shows only the region of interest in white.

**✅ What to Say to Invigilator:**

"In gray level slicing, we isolate a specific range of pixel values and make them white (highlight), setting all others to black. It helps bring out regions of interest without preserving the background."

**🧠 Viva Questions & Answers**

| **Viva Question** | **Answer** |
| --- | --- |
| What is thresholding? | It is a segmentation technique to convert grayscale images into binary based on intensity comparison with a threshold value. |
| What is the use of thresholding? | To extract objects or regions from the background, such as text from a scanned document. |
| What is a binary image? | An image where each pixel is either 0 (black) or 255 (white). |
| What are the different types of thresholding? | Simple, Adaptive, and Otsu’s thresholding. |
| What is gray level slicing? | A method to highlight a range of gray levels in an image and suppress others. |
| What’s the difference between slicing and thresholding? | Thresholding gives binary output (black or white), while slicing highlights a specific intensity range. |
| What is the typical use of gray level slicing? | Medical imaging (e.g., highlighting tumors), satellite images, or feature enhancement. |
| Why convert images to double for processing? | To perform arithmetic operations without overflow, as uint8 is limited to 0–255. |
| What happens if we don’t convert back to uint8? | The image might not display correctly as it won’t match the expected data type. |

Prac8

**🧪 Aim:**

To study **image segmentation** techniques using:

1. Differentiation of Gaussian (DoG) function
2. DoG filter function
3. Edge detection with different operators

**🧩 Part A: Differentiation of Gaussian Function**

**✅ Concept:**

* The **Gaussian function** smooths an image (reduces noise).
* The **Differentiation of Gaussian (DoG)** is the second derivative of the Gaussian.
* It helps in detecting **edges** by identifying areas where intensity changes rapidly.

**🔍 Explanation of Code Steps:**

1. You input a value for **σ (sigma)** – which controls the smoothness.
2. A range -10 to 10 is created to simulate 1D space.
3. The **DoG formula** is applied to generate a curve.
4. This curve shows where edges would be strongest in a real image.

**🎯 What to tell Invigilator:**

"We take the second derivative of the Gaussian function to highlight areas in the image where intensity changes quickly, which is useful for edge detection. The shape of the function is like a wave — it goes up, then down — capturing transitions in pixel values."

**🧩 Part B: DoG Filter Function**

**✅ Concept:**

* This simulates **Difference of Gaussians** using two different sigma values.
* It acts like a **band-pass filter**, enhancing features within a specific frequency range.

**🔍 Explanation of Code Steps:**

1. You input **two sigma values**, σ1 and σ2.
2. Two Gaussian derivatives are calculated using these sigmas.
3. Subtracting the two results gives a **DoG filter curve**.
4. This visually shows how **feature enhancement** or edge detection can change depending on the sigma values.

**🎯 What to tell Invigilator:**

"Here we subtract two Gaussian derivatives with different sigma values. This is called the Difference of Gaussians. It emphasizes edges more clearly and can be used for detecting structures of specific sizes in images."

**🧩 Part C: Edge Detection using Operators**

**✅ Concept:**

This part uses 4 standard **edge detection algorithms**:

* **Sobel**
* **Prewitt**
* **LoG (Laplacian of Gaussian)**
* **Canny**

**🔍 Explanation of Code Steps:**

1. The original image is **converted to grayscale** for simplicity.
2. The edge() function is applied using each method:
   * **Sobel**: Detects edges using a horizontal and vertical mask.
   * **Prewitt**: Similar to Sobel but uses a simpler averaging kernel.
   * **LoG**: Combines Gaussian smoothing and Laplacian edge detection.
   * **Canny**: Most advanced — uses gradient calculation + non-maximum suppression + double thresholding.
3. Each result is displayed in a separate figure.

**🎯 What to tell Invigilator:**

"We apply multiple edge detection algorithms to compare their effectiveness. Sobel and Prewitt detect basic vertical and horizontal edges. LoG uses smoothing before detecting changes, and Canny is more precise and accurate, detecting even faint edges with better noise suppression."

**🧠 Viva Questions & Answers**

| **Question** | **Answer** |
| --- | --- |
| What is image segmentation? | It is the process of dividing an image into meaningful regions or objects for easier analysis. |
| What is the role of sigma in Gaussian? | Sigma controls the spread of the Gaussian. Higher sigma means more smoothing. |
| Why do we use DoG? | It enhances edges and features by subtracting Gaussian-blurred images with different sigmas. |
| What is the main difference between Sobel and Prewitt? | Sobel gives more weight to center pixels, making it slightly more accurate than Prewitt. |
| What is the best edge detector? | Canny is considered the best because it’s accurate, detects weak edges, and reduces noise. |
| What is LoG in edge detection? | LoG stands for Laplacian of Gaussian. It first smooths the image with a Gaussian and then applies a Laplacian filter. |
| What happens if sigma is too low or too high? | Too low = less smoothing (noise preserved), Too high = oversmoothing (edges may be lost). |
| Why convert image to grayscale? | Edge detection works on intensity values, which are present in grayscale images. |

Prac9

**🧪 Aim:**

To implement a **simple image compression** technique using **block-wise processing**, **mean**, **standard deviation**, and **binary thresholding**.

**📌 What This Practical Does (Summary):**

This practical:

1. Takes a **4×4 matrix** as an image block (gray values).
2. Divides it into smaller **sub-blocks** (user-defined size).
3. Computes **mean and standard deviation** for each block.
4. Uses this info to create a **binary allocation matrix**.
5. Based on that, calculates two representative values: **a** (low) and **b** (high).
6. Replaces block values using **only a and b** → this compresses the block.

**🧩 Main Concepts Explained:**

**✅ 1. Block Division**

* The image is divided into blk x blk sub-blocks.
* This allows processing in small chunks → like JPEG does in 8x8 blocks.

**✅ 2. Mean (m) and Standard Deviation (σ)**

* **Mean (m)**: Average pixel intensity in the block.
* **Standard Deviation (σ)**: How much pixel values vary from the mean.

Used to estimate brightness and contrast in that block.

**✅ 3. Binary Allocation Matrix (B)**

* Compares each pixel in the block with the mean:
  + If pixel > mean → 1
  + Else → 0
* Result is a binary matrix of the same block size.

This helps identify which pixels are “brighter” than average.

**✅ 4. Values a and b (ml and mu)**

* These are the two pixel values we will keep for compression.
* ml (low value) = for 0s in B (darker pixels)
* mu (high value) = for 1s in B (brighter pixels)
* These are computed using the **mean, standard deviation, and ratio of 1s/0s** in B.

This helps recreate the block using only 2 values instead of many.

**✅ 5. Reconstruction**

* Replace every 1 in B with mu, every 0 with ml.
* That gives a compressed approximation of the original block.

So now, each block is represented with just two intensity levels instead of all the original values.

**🧠 How to Explain to Invigilator:**

"We divide the image matrix into smaller blocks and compress each block using only two values: one for the brighter pixels and one for the darker pixels. These values are calculated using the block's mean and standard deviation. The binary matrix helps us identify which pixel gets which value. This helps us reduce data and is a basic form of image compression."

**💬 Viva Questions and Answers**

| **Question** | **Answer** |
| --- | --- |
| What is image compression? | Reducing the amount of data needed to represent an image without significantly affecting its quality. |
| Why use blocks in compression? | Processing smaller blocks reduces complexity and helps preserve local features, similar to JPEG compression. |
| What is the role of the mean in this practical? | It's used as a threshold to separate bright and dark pixels. |
| What is the use of the binary matrix B? | It marks which pixels are above (1) or below (0) the mean, used to reconstruct the image with fewer values. |
| Why do we use only two values (a and b)? | To reduce the data size — it's a form of **lossy compression** using just two intensity levels per block. |
| What happens if all values are above or below the mean? | No compression is applied — we skip that block because it already has uniform intensity. |
| Is this lossless or lossy compression? | Lossy — because we lose the exact original pixel values and use approximated ones. |
| Can we improve this method? | Yes, by using more levels or adaptive block sizes or transform-based methods like DCT. |

Prac10

**🧪 Aim:**

To perform:

1. **Binary Image Processing** – using **dilation, erosion, opening, and closing**
2. **Color Image Processing** – extracting and modifying **RGB components** of a color image

**🧩 1. Dilation and Erosion (Binary Image Processing)**

**✅ What is Dilation?**

* Dilation **expands** white regions in a binary image.
* Useful to **fill gaps, connect objects**, or **make shapes thicker**.
* It adds pixels to object boundaries.

**✅ What is Erosion?**

* Erosion **shrinks** white regions.
* Useful to **remove noise** or **disconnect objects**.
* It removes pixels from object boundaries.

**✅ What is the structuring element?**

* A small shape (like a 7×7 rectangle here) used to define **how the operation is applied**.
* Acts like a kernel or mask.

**✅ Explanation:**

* You read an image (usually a binary or grayscale).
* Apply **imdilate()** and **imerode()** with the structuring element.
* Show the result using imshow().

**🧩 2. Opening and Closing (Morphological Operations)**

**✅ What is Opening?**

* **Erosion followed by Dilation**.
* Removes **small white noise** or thin protrusions.
* Good for **smoothing object contours**.

**✅ What is Closing?**

* **Dilation followed by Erosion**.
* Fills **small black holes** or gaps inside objects.
* Good for **joining broken parts** of an object.

**✅ Explanation:**

* Image is processed using **imopen()** and **imclose()** with a structuring element.
* The result is smoother and cleaner objects.

**🧩 3. Extracting RGB Components (Color Image Processing)**

**✅ How does an RGB image work?**

* A color image is made of **Red, Green, and Blue channels**, each 2D matrix of pixel values.
* These are stacked into a 3D matrix (height × width × 3).

**✅ Component Extraction:**

* To isolate one color:
  + Set the **other two channels to zero**.
* For example:
  + To show only **Red**, set Green and Blue to 0.
* This helps you **visualize individual color contributions**.

**🧩 4. Removing RGB Planes (Color Separation)**

**✅ Concept:**

* Instead of showing **only one component**, this shows how the image looks when **one color component is removed**.
* This affects the overall image color:
  + Removing red makes the image appear **bluish-green**, and so on.

**✅ Why is it useful?**

* Helps in understanding how each component affects the final image.
* Important in color correction and processing tasks.

**🧠 How to Explain to Invigilator:**

"In the first part, we apply morphological operations like dilation and erosion to enhance or clean up binary images. Dilation expands objects, and erosion shrinks them. We also use opening and closing to remove noise and fill small gaps. In the second part, we work with RGB images. We isolate each color component to understand its contribution, and also observe the effect of removing any one component from the image."

**💬 Viva Questions and Answers**

| **Question** | **Answer** |
| --- | --- |
| What is morphological image processing? | It’s a technique based on shapes. It uses structuring elements to process images, commonly for binary images. |
| What is the use of dilation? | To expand object boundaries, fill small holes, or connect broken parts. |
| What is erosion used for? | To remove noise, shrink objects, or disconnect connected parts. |
| What is the difference between opening and closing? | Opening = erosion followed by dilation (removes noise), Closing = dilation followed by erosion (fills gaps). |
| What is a structuring element? | A small matrix (shape) used to probe the image during morphological operations. |
| What are RGB components? | Red, Green, and Blue channels that make up a color image. |
| What happens when we remove a color channel? | The image’s color balance changes, allowing us to study the impact of that component. |
| Why is color component separation useful? | For tasks like color analysis, correction, or segmentation in image processing. |
| Are these techniques used in real-world applications? | Yes — in medical imaging, document scanning, satellite image processing, and object detection |

Prac11

**🧪 Aim:**

To perform **image arithmetic operations**:

* Addition
* Subtraction
* Multiplication
* Division

on two images.

**🔍 What is Image Arithmetic?**

Image arithmetic involves performing **mathematical operations pixel-by-pixel** on two images of the **same size**. It is useful in:

* Enhancing or comparing images
* Removing background or noise
* Masking parts of images
* Creating special effects

**📌 Main Points for Each Operation:**

**🟢 1. Image Addition**

* **What it does:**  
  Adds the pixel values of two images.

c(i,j)=a(i,j)+b(i,j)c(i,j) = a(i,j) + b(i,j)c(i,j)=a(i,j)+b(i,j)

* **Use cases:**
  + Brighten an image
  + Combine details from two different sources (e.g., CT + MRI)
  + Create overlay effects
* **Important step:**  
  Both images must be resized to the **same dimensions**.

**🔴 2. Image Subtraction**

* **What it does:**  
  Subtracts one image from another:

c(i,j)=a(i,j)−b(i,j)c(i,j) = a(i,j) - b(i,j)c(i,j)=a(i,j)−b(i,j)

* **Use cases:**
  + Background subtraction
  + Motion detection
  + Spotting differences between two frames or images
* **Note:**  
  Negative pixel values may get clipped to 0 (black), depending on data type.

**🔵 3. Image Multiplication**

* **What it does:**  
  Multiplies the corresponding pixels of two images:

c(i,j)=a(i,j)×b(i,j)c(i,j) = a(i,j) \times b(i,j)c(i,j)=a(i,j)×b(i,j)

* **Use cases:**
  + Masking: multiply with a binary mask
  + Enhancing certain regions
  + Pixel-wise blending
* **Caution:**  
  Result can go beyond the 0–255 range, so image might appear too bright or dark if not normalized.

**🟣 4. Image Division**

* **What it does:**  
  Divides pixels of one image by another:

c(i,j)=a(i,j)b(i,j)c(i,j) = \frac{a(i,j)}{b(i,j)}c(i,j)=b(i,j)a(i,j)​

* **Use cases:**
  + Normalize an image
  + Compare intensities
  + Highlight structural differences
* **Important:**
  + Handle division by zero (black pixels = 0) — this can create errors or undefined pixels.

**💬 How to Explain to Invigilator:**

"In this practical, we performed basic arithmetic operations on images. We first resize both images to the same size to ensure pixel-by-pixel operations work correctly. Then we applied addition to enhance brightness or combine details, subtraction to detect differences, multiplication to mask or emphasize parts, and division to compare or normalize intensity values. These operations are common in preprocessing and analysis in medical imaging, object tracking, and satellite image analysis."

**🧠 Viva Questions and Answers**

| **Question** | **Answer** |
| --- | --- |
| Why do we resize images before arithmetic operations? | To ensure both images are of the same size, so pixel-wise operations can be performed. |
| What is the use of image addition? | To brighten an image, blend two images, or highlight features. |
| What is image subtraction used for? | To detect changes, perform background removal, or compare two images. |
| What is the risk in image multiplication? | Pixel values may exceed 255, leading to overflow or display distortion. |
| What is the main concern in image division? | Division by zero, which can lead to undefined or noisy output. |
| Can we apply image arithmetic on color images? | Yes, but it’s usually done channel-wise (separately for R, G, and B). |
| What are real-world uses of these operations? | In medical imaging, remote sensing, forensics, and video surveillance. |
| What is imadd()? | A built-in function to perform safe addition between two images in MATLAB or Scilab. |