SCHOOL OF ARTIFICIAL INTELLIGENCE

21AIE303

SIGNAL AND IMAGE PROCESSING

TERM PROJECT

B.TECH

CSE-AI(Semester-5)

Detecting Copy-move Forgery using DCT

Under the supervision of: **Dr. Sachin Kumar S**

SUBMITTED BY

GROUP-8

SYKAM SUMANJALI(CB.EN.U4AIE21068)

SUDA HARI PRIYA(CB.EN.U4AIE21067)

R HEMA RADHIKA(CB.EN.U4AIE21050)

SOUVIK GORAIN(CB.EN.U4AIE21065)

ACKNOWLEDGEMENT:

We would like to express our sincere gratitude to our esteemed faculty for their invaluable guidance and support throughout our SIP project on **Detecting Copy-move Forgery using DCT**

Your expertise and mentorship were instrumental in shaping our project's success. Your dedication to our learning and development has been a source of inspiration. Thank you for your unwavering commitment to our academic growth. We deeply appreciate your contributions to our team's journey.

TABLE OF CONTENTS

Detecting Copy move Forgery using DCT

Abstract:

Copy-move forgery is a prevalent and challenging form of image tampering where a portion of an image is duplicated and pasted within the same image to conceal or replicate certain content. Detecting copy-move forgery is essential for maintaining the integrity of digital content. This paper presents a method for detecting copy-move forgery using Discrete Cosine Transform (DCT). The proposed approach involves converting the color image into a grayscale format, dividing it into overlapping blocks, and performing feature extraction using DCT on different feature sets. The features are then subjected to block clustering using the K-means algorithm, and feature matching is accomplished using radix sort. The algorithm effectively identifies duplicated regions, providing a robust solution for detecting copy-move forgery in digital images. Experimental results demonstrate the efficacy of the proposed method in accurately detecting and localizing instances of copy-move forgery. The presented approach contributes to the field of digital forensics by enhancing the ability to identify and mitigate the impact of image tampering.

INTRODUCTION:

The copy move forgery is one of the difficult forgery.

Copy-Move is a special type of image manipulation technique in which a part of the image itself is copied and pasted into another part of the same image.

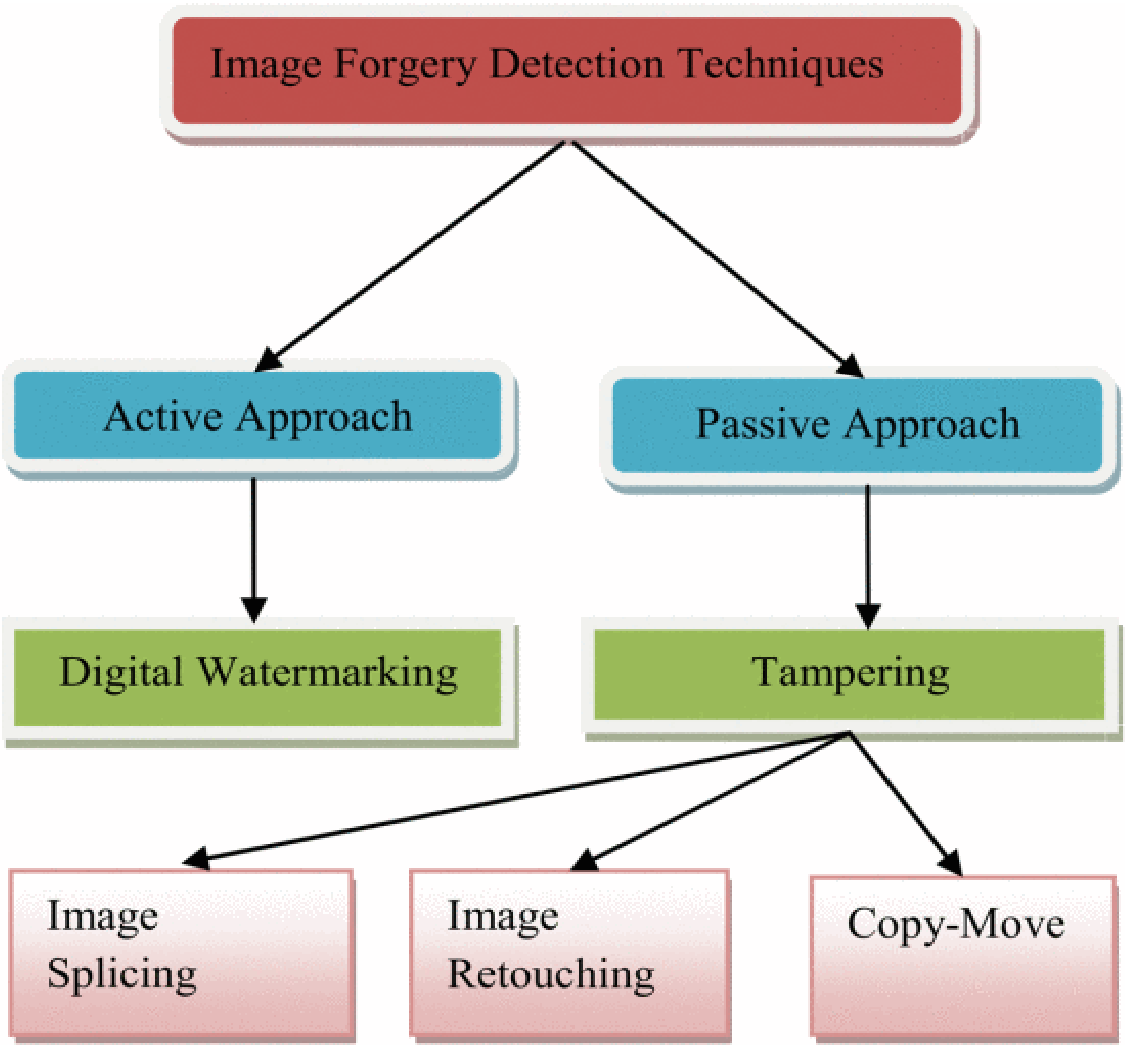


Fig. 1. Forgery detection classification

- **Discrete Cosine Transform(DCT)**

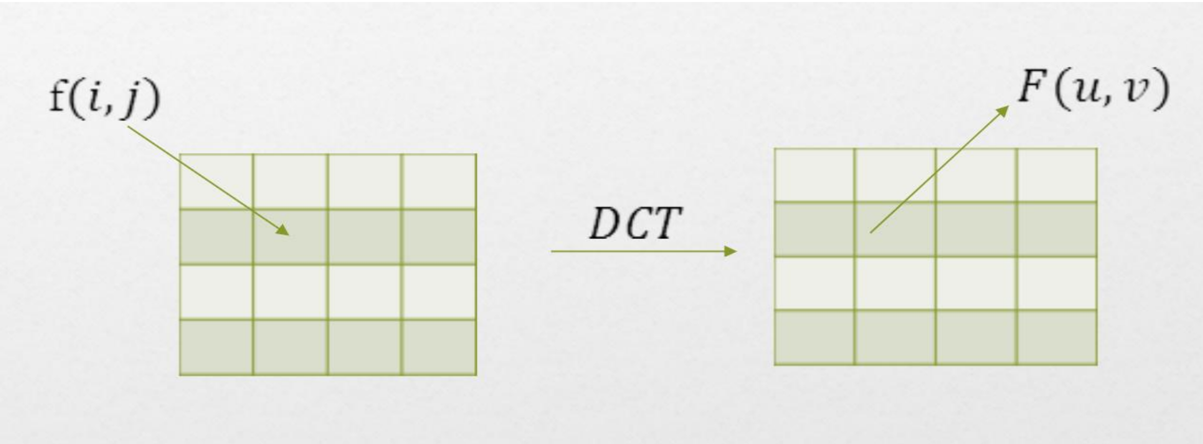
Important to numerous applications in science and engineering from lossy compression of audio and images, to spectral methods for the numerical solution of partial differential equations.

Fourier-related transform is similar to the Discrete Fourier transform (DFT), but using only real numbers.

Transforms an image from the spatial domain to the frequency domain.

Helps separate the image into parts of differing importance.

Expresses a finite sequence of data points in terms of a sum of cosine oscillating at different frequencies.



$$D(u,v) = \frac{2}{b} C(u) C(v) \sum_{x=0}^{b-1} \sum_{y=0}^{b-1} I(x,y) \cos \frac{\pi u(2x+1)}{2b} \cos \frac{\pi v(2y+1)}{2b}$$

$$Where\ C(u) = \left\{ \begin{array}{ll} \frac{1}{\sqrt{2}} & if\ u = 0 \\ 1 & otherwise \end{array} \right\}$$

METHODOLOGY:

Different methods have been developed to detect the image forgery in digital images.

The proposed method uses the DCT coefficients to represent the overlapping block. The DCT coefficients are ordered in zigzag manner to keep the low frequency coefficients together and before the high frequency coefficients in the row vector.

1) **Convert the Color Image into Gray-Scale Image:**

Color image, it can be converted to a grayscale image using the standard formula, $I=0.299R+0.587G+0.114B$. R,G,B represents the three-color components of RGB color model.

2) **Divide the Gray-Scale Image into Overlapping Blocks:**

Divide the gray-scale image into blocks of size 8×8.

3) **Feature Extraction Using DCT:**

Apply DCT to each block for feature extraction.

4) **Block Clustering Using K-Means Algorithm:**

Group the feature vectors (DCT coefficients) using the K-Means clustering algorithm.

5) **Radix Sort for Feature Matching:**

Use Radix Sort for feature matching, likely in the context of comparing and sorting feature vectors for similarity detection.

6) **Duplicate Detection:**
    1. Identify groups of similar blocks within each cluster.
    2. Compare DCT coefficients of neighboring blocks to find similarities.
    3. Calculate the Euclidean distance between similar blocks.
    4. Mark duplicate regions in the result image with red rectangles.
    5.

7) **Display Original and Duplicate Markers:**
    1. Display a new figure showing the original and duplicate markers.
    2. Original markers are shown in green, and duplicate markers are shown in red.

8) **Print Block Size and Execution Time:**
    1. Print the selected block size to the console.
    2. Measure the execution time of the entire process.
    3. Display the execution time on the console.

METHOD ANALYSIS :

- The method demonstrates a comprehensive approach, utilizing image preprocessing, feature extraction, clustering, and visualization to identify regions of potential manipulation.
- The implemented method successfully identifies copy-move forgeries by analyzing DCT coefficients and utilizing clustering techniques.
- Visual markers aid in distinguishing original and duplicate regions within the image.
- The inclusion of execution time information provides insights into the algorithm's efficiency.

PCA Algorithm:

1. Given image convert into grey scale
   Let N be total number of pixels and b denote number of square pixels in square block.

Using PCA

2. Find eigenvectors and eigenvalues.
3. Sort them in decreasing order.
4. Compute projections of centred testing gram matrix on ordered eigenvectors.
5. Lexicographically sort projected version of centred testing gram matrix denoted by S.
6. For every i^{th} rows i in S, select no. of subsequent rows, sj such that $|i-j|<=R^{th}$ and place all pairs of coordinates (xi,yi) and (xj,yj) on to list P_in.
7. Compute offset for each row of P_in.
8. Compute frequency offset.
9. Those rows in P_in which have high frequency offsets are duplicated regions.

PCA Efficiency:

- Due to the inherent characteristics of Principal Component Analysis (PCA), the number of features required to represent a block is significantly reduced compared to Fridrich's method.
- The reduction in the number of features contributes to improved time complexity, making the PCA-based method computationally more efficient.

Limitation of PCA:

- Despite its efficiency, the PCA-based method lacks robustness against small rotations of copy-moved regions.
- This limitation may impact the accuracy of detection, particularly in scenarios involving slight rotations of duplicated content.

Advantages of DCT:

- Discrete Cosine Transform (DCT) is widely utilized for representing images in the frequency domain.
- DCT exhibits the capability to represent most of the intensity distribution details with a reduced number of coefficients, making it a valuable tool in image processing.

DCT vs. PCA Time Complexity:

- DCT, as an algorithm, demonstrates quicker processing times due to its inherent dimension reduction properties.
- The modified matching algorithm employed with DCT enhances the overall efficiency without compromising on robustness.

JPEG Image Considerations:

- DCT is found to be superior to PCA, especially in the context of detecting forgeries in JPEG images.
- The efficiency of DCT makes it a preferable choice for identifying and mitigating tampering in JPEG images compared to using a predefined method like PCA.

Algorithm Modification:

- The switch to DCT in the approach aims to address the inefficiency of PCA in detecting forgeries in JPEG images effectively.
- The modification enhances the overall program efficiency, making it capable of detecting forgeries in diverse image formats, including JPEG.

In summary, the adoption of DCT in forgery detection proves to be advantageous, providing a balance between computational efficiency and robust detection across various image scenarios, including those involving small rotations and JPEG compression.

Table 1: Comparative study of existing techniques.

S. No.	Paper title	Method used	Tampering detection type	Pros/cons	Publication year
1.	Detection of copy-move forgery in digital image [13]	DCT	Copy-move region is detected	Will not work in noisy image	2003
2.	Exposing digital forgeries by detecting duplicated image regions [14]	PCA	Exact copy-move region is detected automatically	Time complexity is high	2004
3.	Robust detection of region duplication in digital image [16]	Similarity matching	Copy-move region detected in noisy conditions	Time complexity is reduced [14]	2006
4.	A sorted neighbourhood approach for detecting duplicate reason based on DWT and SVD [10]	DWT-SVD	Efficiently detects forged region	Time complexity is less compared to other algorithms [14]	2007
5.	A new approach for detecting copy-move forgery detection in digital image [17]	DWT	Exact copy-move region is detected	Works well in noisy and compressed image	2008
6.	Detection of copy-move forgery in digital images using SIFT algorithm [9]	SIFT	Copy-move region is detected	Detects false result also	2008
7.	Identifying tampered regions using singular value decomposition in Digital image forensics [8]	SVD	Copy-Move region is detected accurately	Will not work in highly noised & compressed image	2008
8.	Fast copy-move forgery detection [15]	Improved PCA	Exact Copy-Move region is detected	Works well in noisy, compressed image	2009
9.	Detect digital image splicing with visual cues [6]	DW-VAM	In spliced image, forged region is detected	Work only in the Splicing	2009
10.	Fast, automatic and fine-grained tempered JPEG image detection via DCT coefficient analysis [19]	Double Quantization – DCT	Tampered region is detected accurately	Works only in JPEG Format	2009
11.	Copy-move forgery detection in digital image [18]	SVD	Forged region is detected	Will not work well in noisy image	2010
12.	DWT-DCT based Copy-Move image forgery detection [11]	DCT-DWT	Forged region is detected accurately	Will not work in highly compressed image	2011
13.	An integrated technique for splicing and copy-move image forgery detection [7]	DCT-SURF	Copy-Move and spliced both region detected	Works well for both copy-move and splicing	2011
14.	Improved DCT-based detection of copy-move forgery in digital image [22]	DCT	Copy-move region detected accurately	Works well if the image blurred & compressed	2011
15.	A robust detection algorithm for copy move forgery in a digital image [23]	DCT	Exact copy-move region detected	Works well if the image is noisy or blurred	2012

RESULTS:

COPY-MOVE FORGERY DETECTION IN PYTHON

```
import numpy as np
from scipy import fft
import cv2
from operator import itemgetter
from google.colab.patches import cv2_imshow

class QuantizationMatrix():
    Q50 = np.array([[16, 11, 10, 16, 24, 40, 51, 61],
                    [12, 12, 14, 19, 26, 58, 60, 55],
                    [14, 13, 16, 24, 40, 57, 69, 56],
                    [14, 17, 22, 29, 51, 87, 80, 62],
                    [18, 22, 37, 56, 68, 109, 103, 77],
                    [24, 35, 55, 64, 81, 104, 113, 92],
                    [49, 64, 78, 87, 103, 121, 120, 101],
                    [72, 92, 95, 98, 112, 100, 103, 99]])

    Q75 = np.array([[8, 6, 5, 8, 12, 20, 26, 31],
                    [6, 6, 7, 10, 13, 29, 30, 28],
                    [7, 7, 8, 12, 20, 29, 35, 28],
                    [7, 9, 11, 15, 26, 44, 40, 31],
                    [9, 1, 19, 28, 34, 55, 52, 39],
                    [12, 18, 28, 32, 41, 52, 57, 46],
                    [25, 32, 39, 44, 52, 61, 60, 52],
                    [36, 46, 48, 49, 56, 50, 52, 50]])

    Q90 = np.array([[3, 2, 2, 3, 5, 8, 10, 12],
                    [2, 2, 3, 4, 5, 12, 12, 11],
                    [3, 3, 3, 5, 8, 11, 14, 11],
                    [3, 3, 4, 6, 10, 17, 16, 12],
                    [4, 4, 7, 11, 14, 22, 21, 15],
                    [5, 7, 11, 13, 16, 12, 23, 18],
                    [10, 13, 16, 17, 21, 24, 24, 21],
                    [14, 18, 19, 20, 22, 20, 20, 20]])

    Qrand = np.array([[4, 4, 6, 11, 24, 24, 24, 24],
                      [4, 5, 6, 16, 24, 24, 24, 24],
                      [6, 6, 14, 24, 24, 24, 24, 24],
                      [11, 16, 24, 24, 24, 24, 24, 24],
                      [24, 24, 24, 24, 24, 24, 24, 24],
                      [24, 24, 24, 24, 24, 24, 24, 24],
                      [24, 24, 24, 24, 24, 24, 24, 24],
                      [24, 24, 24, 24, 24, 24, 24, 24]])

    def get_qm(self, qf=0.75):
        if qf == 0.5:
            return self.Q50
```

```python
        elif qf == 0.75:
            return self.Q75
        elif qf == 0:
            return self.Qrand
        elif qf == 0.9:
            return self.Q90


def read_img(img_path):
    # cv2.imread directly to read the image
    original_image = cv2.imread(img_path, cv2.IMREAD_COLOR)

    # To Check if the image is loaded successfully
    if original_image is None:
        raise ValueError(f"Error: Unable to load image from
{img_path}")

    # Convert to grayscale
    image = cv2.cvtColor(original_image, cv2.COLOR_BGR2GRAY)

    overlay = original_image.copy()

    img = np.array(image)
    height, width = img.shape

    return img, original_image, overlay, width, height

def create_quantize_dct(img, width, height, block_size, stride, Q_8x8):
    quant_row_matrices = []

    for i in range(0, height - block_size, stride):
        for j in range(0, width - block_size, stride):
            block = img[i: i + block_size, j: j + block_size]

            # DCT
            dct_matrix = fft.dct(block)

            # Quantization of DCT coefficients
            quant_block = np.round(np.divide(dct_matrix, Q_8x8))
            block_row = list(quant_block.flatten())

            # Left-corner pixel coordinates and block
            quant_row_matrices.append([(i, j), block_row])

    return quant_row_matrices

def lexographic_sort(quant_row_matrices):
    sorted_blocks = sorted(quant_row_matrices, key=itemgetter(1))
```

```
    # FORMAT: [[block1], [block2], (pos1), (pos2), shift vector]
    matched_blocks = []

    # To keep track of shift count
    shift_vec_count = {}

    for i in range(len(sorted_blocks) - 1):
        if sorted_blocks[i][1] == sorted_blocks[i + 1][1]:
            point1 = sorted_blocks[i][0]
            point2 = sorted_blocks[i + 1][0]

            # Shift vector
            s = np.linalg.norm(np.array(point1) - np.array(point2))

            # Increment count for s
            shift_vec_count[s] = shift_vec_count.get(s, 0) + 1
            matched_blocks.append([sorted_blocks[i][1], sorted_blocks[i
+ 1][1],
                                   point1, point2, s])

    return shift_vec_count, matched_blocks

def shift_vector_thresh(shift_vec_count, matched_blocks, shift_thresh):
    matched_pixels_start = []
    for sf in shift_vec_count:
        if shift_vec_count[sf] > shift_thresh:
            for row in matched_blocks:
                if sf == row[4]:
                    matched_pixels_start.append([row[2], row[3]])

    return matched_pixels_start

def display_results(overlay, original_image, matched_pixels_start,
block_size):
    alpha = 0.5
    orig = original_image.copy()

    for starting_points in matched_pixels_start:
        p1 = starting_points[0]
        p2 = starting_points[1]

        overlay[p1[0]: p1[0] + block_size, p1[1]: p1[1] + block_size] =
(0, 0, 255)
        overlay[p2[0]: p2[0] + block_size, p2[1]: p2[1] + block_size] =
(0, 255, 0)

    cv2.addWeighted(overlay, alpha, original_image, 1, 0,
original_image)
```

```
    cv2_imshow(orig)
    print("Input Image: Original Image")
    cv2_imshow(original_image)
    print("Output Image: Forged regions marked in red, Original regions
marked in green")
    cv2.waitKey(0)
    cv2.destroyAllWindows()


# Input image path
img_path = '/content/forged2.png' # Change this to your image path

# User-defined parameters
block_size = int(input("Enter the block size: "))
qf = float(input("Enter the quality factor: "))
shift_thresh = int(input("Enter the shift vector threshold: "))
stride = int(input("Enter the sliding window stride: "))

# 8x8 quantization matrix based on QF
Q_8x8 = QuantizationMatrix().get_qm(qf)

# Read image
img, original_image, overlay, width, height = read_img(img_path)

# DCT
quant_row_matrices = create_quantize_dct(img, width, height,
block_size, stride, Q_8x8)

# Lexicographic sort
shift_vec_count, matched_blocks = lexographic_sort(quant_row_matrices)

# Shift vector thresholding
matched_pixels_start = shift_vector_thresh(shift_vec_count,
matched_blocks, shift_thresh)

# Displaying output
display_results(overlay, original_image, matched_pixels_start,
block_size)
```

OUTPUT:

Original Image

Forged regions : Red ; Original : Green

MATLAB :

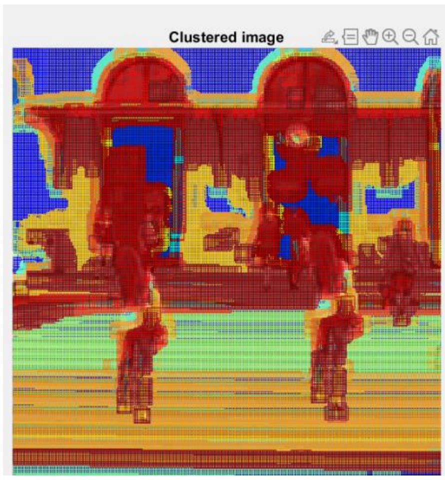
```
function DetectCopyMoveForgery
    clc
    clear
    blocksize = 8;
    overlap = 1;
    Nd = 16;
    Th = 0.9999;
    s_threshold = 2;

    % Get input image
    [filename, path] = uigetfile('*.*', "Select an Image");
    img = imread(fullfile(path, filename));
    imshow(img);
    title('Original image');
    [r, c, n] = size(img);
    if n > 1
        im = rgb2gray(img);
    else
        im = img;
    end
    figure;
    imshow(im), title('Gray image');

    % Display block size
    disp(['Block Size: ', num2str(blocksize)]);

    % Divide into overlapping blocks
    a = 1;
    for j = 1:overlap:(c - blocksize) + 1
        for i = 1:overlap:(r - blocksize) + 1
            sondos(a).block = im(i:i + blocksize - 1, j:j + blocksize - 1);
            sondos(a).position = [i, j];
            sondos(a).index = a;
```

```
            a = a + 1;
        end
    end

    % Measure execution time
    tic;

    % Apply DCT for each block
    sz = numel(sondos);
    DC = zeros(sz, 1);
    FDCT = zeros(sz, blocksize^2);  % Assuming blocksize x blocksize DCT
    for a = 1:sz
        [feature, ~] = featureExtraction(sondos(a).block);
        DC(a) = feature(1);
        FDCT(a, :) = feature(:)';
    end

    % Divide into groups
    numloss = 20;
    try
        % Increase the maximum number of iterations
        maxIterations = 500;
        opts = statset('MaxIter', maxIterations);
        [idx, centers] = kmeans(FDCT, numloss, 'Options', opts);

    catch
        error('Failed to converge. Consider adjusting parameters or preprocessing
the data.');
    end

    G = cell(numloss, 1);
    for n = 1:numloss
        G{n} = find(idx == n);
    end

    % Draw segmentation
    col = jet(numloss);
    figure;
    imshow(im), title('Clustered image');
    for e = 1:numloss
        color = col(e, :);
        for ee = 1:numel(G{e})
            idx = G{e}(ee);
            rectangle('Position', [sondos(idx).position(2),
sondos(idx).position(1), blocksize, blocksize], 'EdgeColor', color);
        end
    end

    % Detect CM and identify duplicates
    figure;
    imshow(im), title('Result image');
    detectedDuplicates = false(sz, 1);
    for n0 = 1:numloss
        emp = find(G{n0} == 0);
        if ~isempty(emp)
            A = zeros(numel(emp), blocksize^2 + 2);
            for a = 1:numel(emp)
                idx = G{n0}(emp(a));
                [f, ~] = featureExtraction(sondos(idx).block);
```

```
                A(a, 1:blocksize^2) = f;
                A(a, end-1) = sondos(idx).position(1);
                A(a, end) = sondos(idx).position(2);
            end
        else
            A = zeros(numel(G{n0}), blocksize^2 + 2);
            for a = 1:numel(G{n0})
                idx = G{n0}(a);
                [f, ~] = featureExtraction(sondos(idx).block);
                A(a, 1:blocksize^2) = f;
                A(a, end-1) = sondos(idx).position(1);
                A(a, end) = sondos(idx).position(2);
            end
        end

        Asorted = sortrows(A, 1:9);

        for i = 1:size(Asorted, 1) - 1
            similar = abs(Asorted(i + 1, 1:9) - Asorted(i, 1:9)) < s_threshold;
            if all(similar)
                x1 = Asorted(i, end-1);
                x2 = Asorted(i + 1, end-1);
                y1 = Asorted(i, end);
                y2 = Asorted(i + 1, end);
                D = sqrt((x1 - x2)^2 + (y1 - y2)^2);
                if D > Nd
                    detectedDuplicates(G{n0}(i)) = true;
                    detectedDuplicates(G{n0}(i + 1)) = true;
                    rectangle('Position', [y1, x1, blocksize, blocksize],
'EdgeColor', 'r');
                    rectangle('Position', [y2, x2, blocksize, blocksize],
'EdgeColor', 'r');
                end
            end
        end
    end

    % Display original and duplicate markers
    figure;
    imshow(im), title('Original and Duplicate Markers');
    hold on;

    originalIndices = find(~detectedDuplicates);
    duplicateIndices = find(detectedDuplicates);

    % Extract coordinates for original markers
    originalX = zeros(1, numel(originalIndices));
    originalY = zeros(1, numel(originalIndices));
    for i = 1:numel(originalIndices)
        idx = originalIndices(i);
        originalX(i) = sondos(idx).position(2) + blocksize / 2;
        originalY(i) = sondos(idx).position(1) + blocksize / 2;
    end

    % Extract coordinates for duplicate markers
    duplicateX = zeros(1, numel(duplicateIndices));
    duplicateY = zeros(1, numel(duplicateIndices));
    for i = 1:numel(duplicateIndices)
        idx = duplicateIndices(i);
```

```
        duplicateX(i) = sondos(idx).position(2) + blocksize / 2;
        duplicateY(i) = sondos(idx).position(1) + blocksize / 2;
    end

    % Plot original markers in green
    scatter(originalX, originalY, 50, 'g', 'filled');

    % Plot duplicate markers in red
    scatter(duplicateX, duplicateY, 50, 'r', 'filled');

    hold off;

    % Display execution time
    elapsedTime = toc;
    disp(['Execution Time: ', num2str(elapsedTime), ' seconds']);
end

function [feature, vector] = featureExtraction(block)
    dctCoefficients = dct2(block);
    feature = dctCoefficients(:)';
    vector = feature;
end
```

OUTPUT :



LITERATURE REVIEW :


```
Command Window
Block Size: 8
Execution Time: 85.2072 seconds
```

ASPECT	PAPER 1	PAPER 2
Research Focus	Detection of copy move forgery in images	Detection of copy move forgery in digital images
Main Technique	Block Matching Algorithm	Discrete Cosine Transform
Challenges Addressed	Time complexity of block matching algorithms	
Proposed Solution	Use of Discrete Cosine Transform (DCT)	Detection of region duplication forgery
Automation of Threshold	Effort made to automate threshold selection	Not specified
Feature Representation	DCT used for representing features	DCT used for detecting duplicated blocks
Image division Strategy	Overlapping blocks	Overlapping blocks
Perfomance Improvement	Addressed time complexity issues	Not explicitly mentioned
Scope for Improvement	Some issues remain unsolved or need improvement	Detection of tampering is a challenging task

REFERENCES :

https://ieeexplore.ieee.org/document/6707675
https://www.researchgate.net/publication/236632995_Detecting_Copy_move_Forgery_using_DCT
https://link.springer.com/article/10.1007/s42044-019-00029-y#:~:text=Proposed%20method%20includes%20the%20following,5)%20radix%20sort%20for%20feature
https://www.tandfonline.com/doi/pdf/10.1080/09747338.2014.921415