Amrita Vishwa Vidyapeetham
Centre for Excellence in Computational Engineering and Networking
Amrita School of Engineering, Coimbatore

Copy-Move-Forgery-Detection using DCT

Prepared By:
Batch-B GROUP-9

Kuppala Gowtham Sai Chandra - CB.EN.U4AIE21125
Atmuri Likith Adithya - CB.EN.U4AIE21127
Kode Sri Naga Harsha Vardhan - CB.EN.U4AIE21124
Miriyala Ruthvik Guptha - CB.EN.U4AIE21134

Supervised by:
Dr. Sachin Kumar S
Asst. Professor

An End Semester Project submitted to the CEN department as a partof course evaluations of **Signal and Image Processing** for
B. Tech in **Computer Science Engineering – Artificial Intelligence.**

ACKNOWLEDGMENT

We extend our heartfelt appreciation to the Centre for Excellence in Computational Engineering and Networking (CEN) at Amrita Vishwa Vidyapeetham, Coimbatore, for fostering an exceptional environment that enabled us to delve into our research endeavors. Our gratitude knows no bounds towards Dr. Sachin Kumar S, Assistant Professor in the Department of Centre for Excellence in CEN, whose guidance steered us through the completion of our research. Her insights led us to identify the research area, topic, and problem, shaping our scholarly pursuit.We wish to acknowledge our dedicated team members whose unwavering support and collaboration were instrumental throughout this journey. Additionally, we are profoundly thankful to our esteemed university for providing us with this invaluable opportunity for academic exploration and growth.

TABLE OF CONTENTS

S.NO	TOPIC	PAGE NO.
1	ABSTRACT	4
2	INTRODUCTION	5
3	MOTIVATION OF OUR PROJECT	6
4	LITERATURE SURVEY	7
5	METHODS USED FOR IMPLEMENTATION	8-9
6	CODE IMPLEMENTATION AND RESULTS	10 - 16
7	INFERENCE	17
8	CONCLUSION	17

ABSTRACT

This project targets detecting image forgeries caused by internal copying within images while maintaining resilience against compression like JPEG. Employing 8x8 or 16x16 block segmentation aids in localized forgery detection. Utilizing Discrete Cosine Transform (DCT) and zigzag scanning, low-frequency information is extracted for compression-resistant meaningful data retention.Vector representation of DCT outputs, focusing on the initial 16 elements, undergoes quantization for compression resilience. Euclidean distance assessment with a 3.5 threshold against 8 neighboring vectors is employed, followed by a distance filter of at least 100 pixels between block coordinates, ensuring accurate detection, especially in images with analogous regions.

INTRODUCTION

The proliferation of image manipulation techniques demands robust methods to detect forgeries within images, specifically those involving the duplicating and pasting of areas within the same image. This project delves into developing a resilient algorithm capable of identifying such manipulations while withstanding the effects of compression algorithms like JPEG.

The core strategy involves segmenting images into smaller blocks, typically 8x8 or 16x16, enabling localized scrutiny for potential forgeries. Leveraging Discrete Cosine Transform (DCT) techniques, the project focuses on extracting low-frequency components crucial for retaining significant image information even under compression.

By implementing a series of intricate steps—such as vectorization of DCT outcomes, quantization for resistance against compression, and employing Euclidean distance assessments—the project endeavors to establish a comprehensive methodology for forgery detection. This method aims to discern manipulated regions, even amidst images containing similar or nearly identical areas, enhancing the reliability and accuracy of identifying fraudulent alterations within images.

MOTIVATION OF OUR PROJECT

The motivation behind this project stems from the escalating concern regarding the authenticity and integrity of digital images in today's technologically advanced era. As digital manipulation techniques become more sophisticated, the need for robust and resilient methods to detect image forgeries has become imperative.The prevalence of image tampering, especially the subtle yet deceptive act of copying and pasting areas within the same image, presents a significant challenge. Furthermore, the impact of compression algorithms like JPEG poses additional hurdles in identifying these manipulations.This project is driven by the aspiration to develop an algorithm that not only detects such sophisticated forgeries but also remains resilient against compression techniques. The aim is to provide a reliable and effective solution capable of accurately identifying manipulations within images, thereby contributing to bolstering the authenticity and trustworthiness of digital visual content in various domains, including forensics, journalism, and digital media.

Problem Statement

With the advent of social networking services such as Facebook and Instagram, there has been a huge increase in the volume of image data generated in the last decade. Use of image processing software like GNU Gimp, Adobe Photoshop to create doctored images and videos is a major concern for internet companies like Facebook. These images are prime sources of fake news and are often used in malevolent ways such as for mob incitement.Before action can be taken on basis of a questionable image, we must verify its authenticity.

Literature survey

Reference:-

1. Copy-move forgery detection technique based on discrete cosine transform blocks features

Esteban Alejandro Armas Vega1 • Edgar Gonza´lez Ferna´ndez1 • Ana Lucila Sandoval Orozco1 • Luis Javier Garcı´a Villalba1

Grayscale Transformation:

Converting an image to grayscale reduces color information to luminance, simplifying subsequent analyses. This step is fundamental in enhancing the efficiency of forgery detection algorithms

Block-Based Processing:

Dividing an image into small, overlapping blocks facilitates localized analysis. Such an approach assists in detecting manipulations by examining similarities or discrepancies within these block regions.

Discrete Cosine Transform (DCT):

The DCT is applied to each block, representing the frequency content. Zig-zag scanning and truncating the DCT coefficients help capture essential information while reducing computational complexity.

DCT Formulation:-

$$c\left(u,v\right)=\alpha\left(u\right)\alpha\left(v\right)\sum_{x=0}^{N-1}\sum_{u=0}^{M-1}f\left(x,y\right)\cos\left(\pi(2x+1)u/2N\right)\cos\left(\pi(2y+1)v/2M\right)$$

Where,

$$\alpha\left(i\right)=\begin{cases}\sqrt{1/N} & for\ \ i=0\\ \sqrt{2/N} & for\ \ i\neq 0\end{cases}$$

u = 0, 1, 2, .., N-1
v= 0, 1, 2, .., M-1
M x N is size of image

Lexicographic Sorting and Similarity Measurement:

Sorting truncated coefficient lists lexicographically enables efficient comparison. Similarity metrics like Euclidean distance or cross-correlation are computed among neighboring blocks' coefficients. A predefined threshold aids in identifying identical or highly similar blocks

Translation Vector Computation:

Identifying identical blocks involves calculating translation vectors between them. A significant number of similar vectors in a specific direction can indicate copy-move tampering, indicating regions within the image that have been duplicated and pasted.

METHODS USED FOR IMPLEMENTATION

RGB to Grayscale Conversion:

- Utilize the rgb2gray function to convert the color image to grayscale.
- Facilitates simplified processing by reducing the image to a single intensity channel.
- Preserves essential image information while simplifying subsequent analysis.

Discrete Cosine Transform(DCT) :-

- The Discrete Cosine Transform (DCT) is a mathematical transform commonly used in image compression
- DCT converts the spatial representation of the block (pixel intensities) into its frequency representation (DCT coefficients).
- In the copy-move forgery detection, After conversion of image blocks DCT in applied for the image blocks

ZIGZAG algorithm:-

- After quantization, the coefficients are arranged in a two-dimensional array.
- The Zigzag algorithm is applied to this array to convert it into a one-dimensional array.
- The zigzag pattern helps to gather the most significant coefficients first, making it more efficient storage

significant_part_extraction:-

- Extracts the significant part of a vector by deleting high significant values.
- A block size of the vector value will be given and higher frequency values will be deleted
- i.e 1x 64 vector in changed to 1x16 vector by deleting high frequencyExtracts the significant part of a vector by deleting high significant values.

Lexicographically_sorting: -

- We are converting the column-vector to Row-vector in counterclockwise direction(anti-clockwise)
- For this we are using the lexisort function to convert the Column to Row vector in such a way the last element in the column comes to the 1st element in the row.

correlation_of_vectors:

- After getting the Block vectors from the lexisort function, we will check correlation threshold value is greater than length of block vector otherwise we may get out of bound.
- Next for comparing the vectors in the block vectors if the Euclidean distance between the vectors is less than Euclidean threshold we will append the those vectors to separate empty array.
- The actual processing of similar vectors is further handled in the elimination_of_weak_vectors method.

Elimination_of_weak_Vectors:-

- After getting the Vectors from correlation function we will check if the Euclidean distance greater that the vector threshold value.
- If this obeys we will call function elimination_of_weak_area

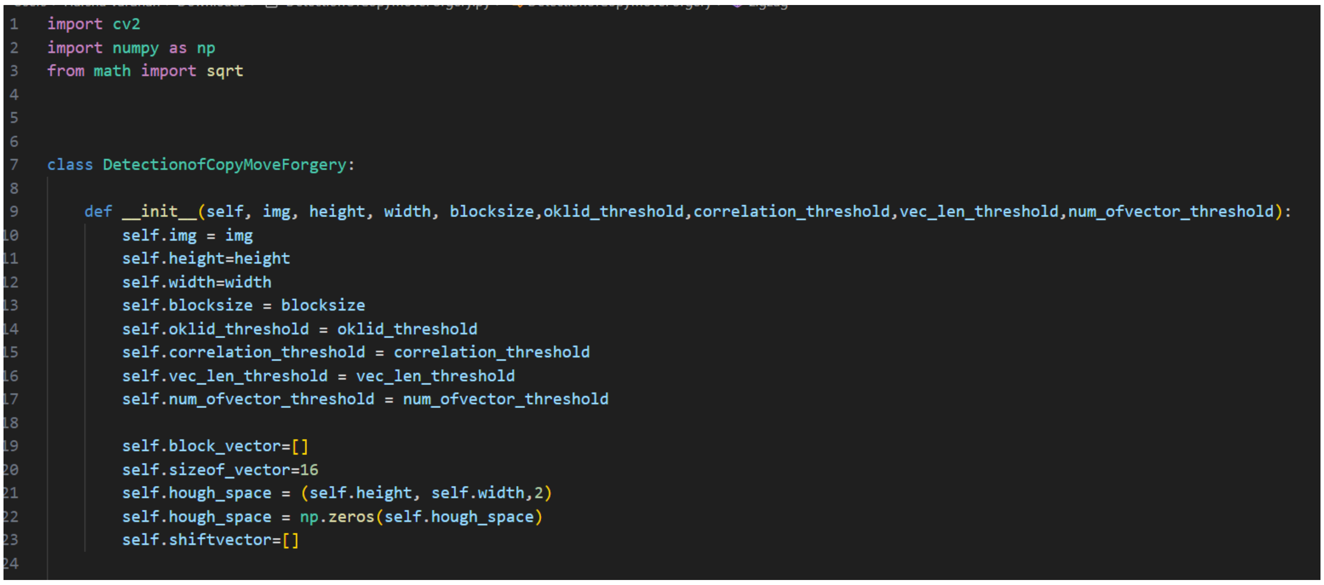
Elimination_of_weak_Area:-

- We will find the displacement and direction of the vectors.
- In order not to lose which block vectors increase which direction, a new vector is created and then the direction is increased.
- the directions of the determined vectors are calculated and the position of this direction in Hough space is increased by one
- Its own coordinates are kept in the vector, and block by block shift in these vectors is assigned to the vector i.e., shift_vectors.

Detection_forgery:-

- At First we will find the Max Hough_space Value for the find the Threshold value.
- Next we are setting img array by setting all pixel values to 0.
- max - (max*self.num_ofvector_threshold/100) using this formula we will calculate the threshold value
- We will check if the coordinates in the Hough space correspond to a shift vector then we will a filled rectangles in the particular coordinates.
- So we will get the output image with white blocks. These blocks are the forged part of the original image.

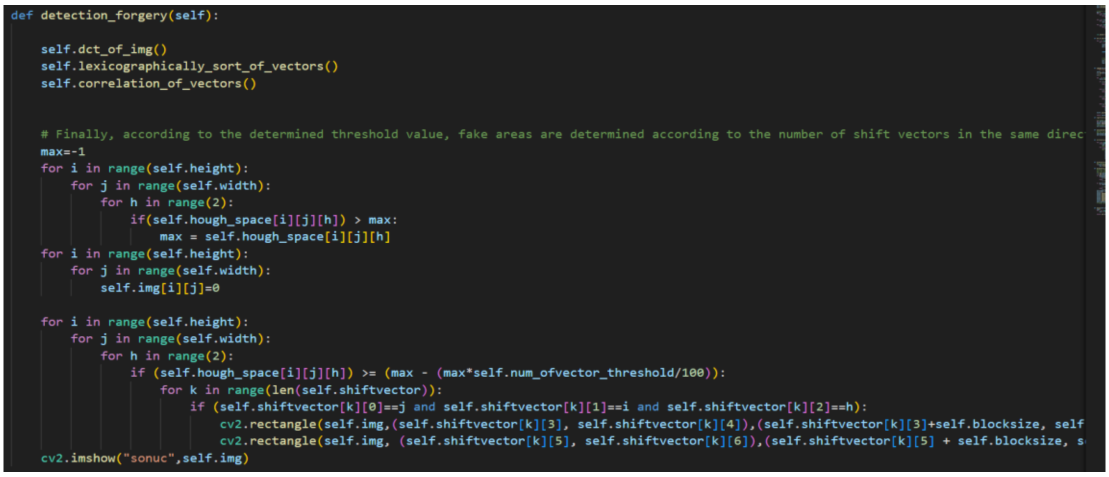
CODE IMPLEMENTATION



```python
import cv2
import numpy as np
from math import sqrt



class DetectionofCopyMoveForgery:

    def __init__(self, img, height, width, blocksize,oklid_threshold,correlation_threshold,vec_len_threshold,num_ofvector_threshold):
        self.img = img
        self.height=height
        self.width=width
        self.blocksize = blocksize
        self.oklid_threshold = oklid_threshold
        self.correlation_threshold = correlation_threshold
        self.vec_len_threshold = vec_len_threshold
        self.num_ofvector_threshold = num_ofvector_threshold

        self.block_vector=[]
        self.sizeof_vector=16
        self.hough_space = (self.height, self.width,2)
        self.hough_space = np.zeros(self.hough_space)
        self.shiftvector=[]

```

Explanation :-

The code defines a class DetectionofCopyMoveForgery for identifying copy-move forgery in an image. It initializes parameters and variables for forgery detection. Using methods such as DCT transformation, lexicographical sorting of vectors, and correlation analysis, it processes image blocks, identifies similarities, and marks potential forged regions in the image based on predefined thresholds. The code executes a step-by-step forgery detection process by transforming blocks, sorting vectors, measuring similarities, and analyzing Hough space to identify and mark potential forged regions within the image.



```python
def detection_forgery(self):

    self.dct_of_img()
    self.lexicographically_sort_of_vectors()
    self.correlation_of_vectors()


    # Finally, according to the determined threshold value, fake areas are determined according to the number of shift vectors in the same direc
    max=-1
    for i in range(self.height):
        for j in range(self.width):
            for h in range(2):
                if(self.hough_space[i][j][h]) > max:
                    max = self.hough_space[i][j][h]
    for i in range(self.height):
        for j in range(self.width):
            self.img[i][j]=0

    for i in range(self.height):
        for j in range(self.width):
            for h in range(2):
                if (self.hough_space[i][j][h]) >= (max - (max*self.num_ofvector_threshold/100)):
                    for k in range(len(self.shiftvector)):
                        if (self.shiftvector[k][0]==j and self.shiftvector[k][1]==i and self.shiftvector[k][2]==h):
                            cv2.rectangle(self.img,(self.shiftvector[k][3], self.shiftvector[k][4]),(self.shiftvector[k][3]+self.blocksize, self
                            cv2.rectangle(self.img, (self.shiftvector[k][5], self.shiftvector[k][6]),(self.shiftvector[k][5] + self.blocksize, s
    cv2.imshow("sonuc",self.img)
```

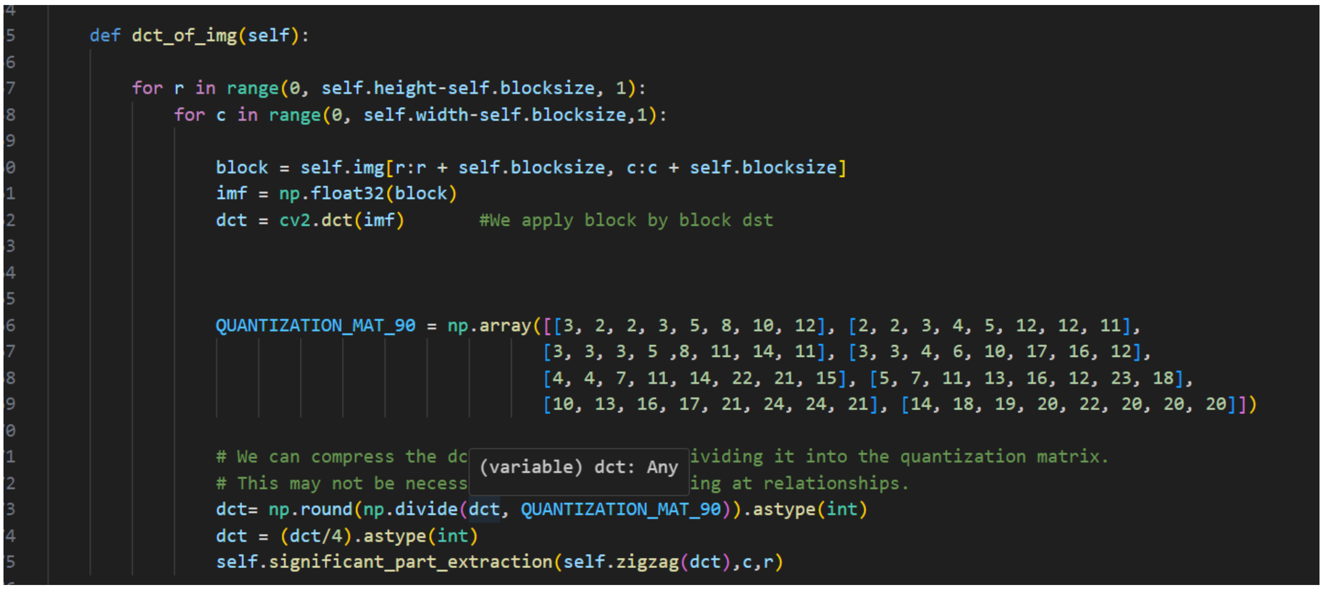
Explanation:-

The detection_forgery() function performs post-processing by marking potential forged areas based on the Hough space analysis:

Find Maximum in Hough Space: It determines the maximum value in the Hough space array.

Thresholding and Area Marking: Using a threshold derived from the maximum value, it identifies regions with a significant number of shift vectors in the same direction and marks these areas as potential forgeries on the self.img.

Visualization: Displays the image with marked potential forged areas using OpenCV's cv2.imshow() function.



```
    def dct_of_img(self):

        for r in range(0, self.height-self.blocksize, 1):
            for c in range(0, self.width-self.blocksize,1):

                block = self.img[r:r + self.blocksize, c:c + self.blocksize]
                imf = np.float32(block)
                dct = cv2.dct(imf)       #We apply block by block dst



                QUANTIZATION_MAT_90 = 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]])

                # We can compress the dc (variable) dct: Any ividing it into the quantization matrix.
                # This may not be necess                    ing at relationships.
                dct= np.round(np.divide(dct, QUANTIZATION_MAT_90)).astype(int)
                dct = (dct/4).astype(int)
                self.significant_part_extraction(self.zigzag(dct),c,r)
```

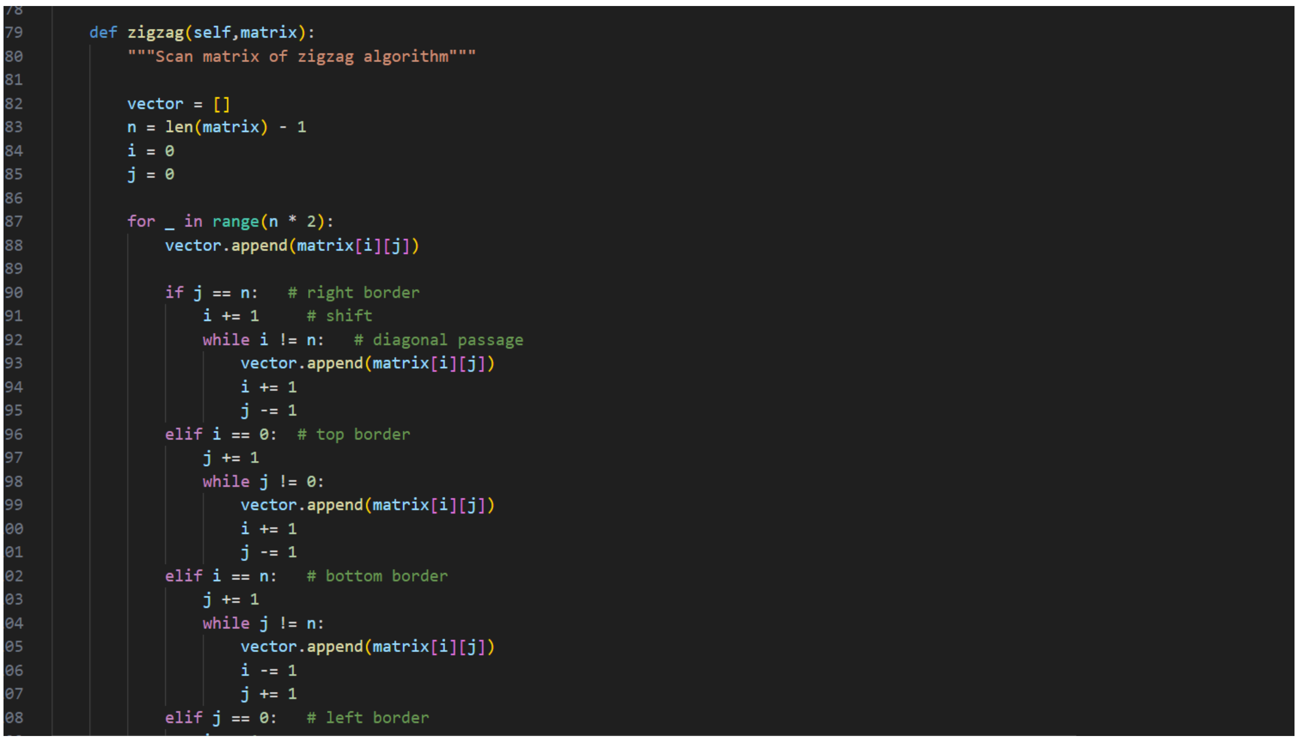
Explanation:-

The dct_of_img() function applies Discrete Cosine Transform (DCT) to image blocks and extracts significant parts:

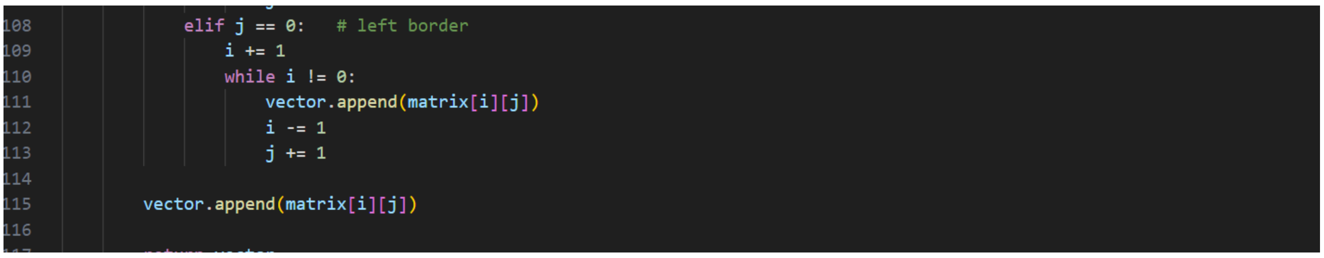
Block-wise DCT: Iterates through the image in block-sized steps, computes DCT for each block using cv2.dct().

Quantization: Divides the DCT coefficients by a predefined quantization matrix for compression.

Extracts Significant Parts: Performs Zigzag scanning on the quantized DCT coefficients, extracts the significant part, and generates block vectors for further processing.



```
    def zigzag(self,matrix):
        """Scan matrix of zigzag algorithm"""

        vector = []
        n = len(matrix) - 1
        i = 0
        j = 0

        for _ in range(n * 2):
            vector.append(matrix[i][j])

            if j == n:   # right border
                i += 1     # shift
                while i != n:   # diagonal passage
                    vector.append(matrix[i][j])
                    i += 1
                    j -= 1
            elif i == 0:  # top border
                j += 1
                while j != 0:
                    vector.append(matrix[i][j])
                    i += 1
                    j -= 1
            elif i == n:   # bottom border
                j += 1
                while j != n:
                    vector.append(matrix[i][j])
                    i -= 1
                    j += 1
            elif j == 0:   # left border
```



```
            elif j == 0:   # left border
                i += 1
                while i != 0:
                    vector.append(matrix[i][j])
                    i -= 1
                    j += 1

        vector.append(matrix[i][j])

```

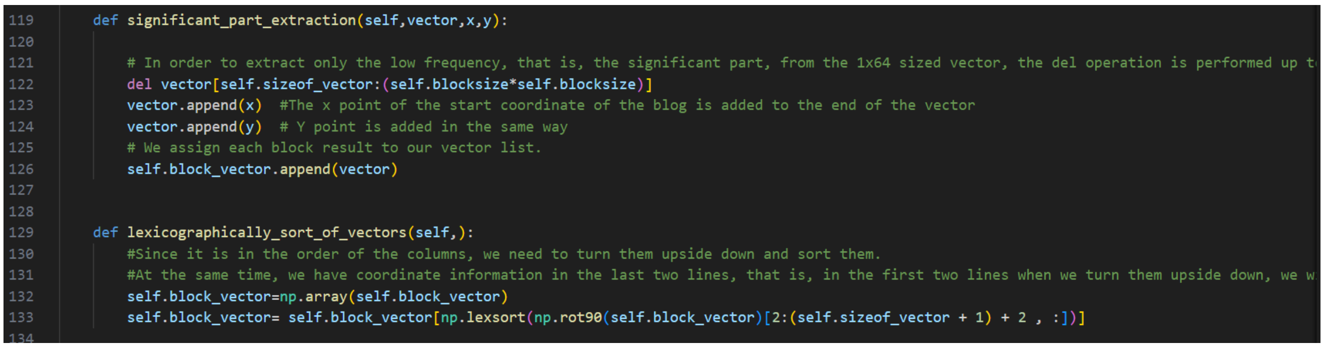
Explanation:-

The zigzag() function implements the Zigzag scanning algorithm on a matrix:

Scanning Approach: Traverses the matrix in a zigzag pattern, appending elements to a vector in a specific order.

Directional Movements: Handles movements based on matrix borders (right, top, bottom, left) by updating indices i and j.

Vector Generation: Constructs and returns a vector containing matrix elements arranged in the zigzag pattern.



```python
    def significant_part_extraction(self,vector,x,y):

        # In order to extract only the low frequency, that is, the significant part, from the 1x64 sized vector, the del operation is performed up t
        del vector[self.sizeof_vector:(self.blocksize*self.blocksize)]
        vector.append(x)  #The x point of the start coordinate of the blog is added to the end of the vector
        vector.append(y)  # Y point is added in the same way
        # We assign each block result to our vector list.
        self.block_vector.append(vector)


    def lexicographically_sort_of_vectors(self,):
        #Since it is in the order of the columns, we need to turn them upside down and sort them.
        #At the same time, we have coordinate information in the last two lines, that is, in the first two lines when we turn them upside down, we w
        self.block_vector=np.array(self.block_vector)
        self.block_vector= self.block_vector[np.lexsort(np.rot90(self.block_vector)[2:(self.sizeof_vector + 1) + 2 , :])]
```

Explanation:-

Method significant_part_extraction()

Vector Truncation: Deletes elements from the vector beyond the specified limit (up to index 16), extracting the significant part.

Coordinate Appending: Appends x and y coordinates to the vector representing the block's starting point.

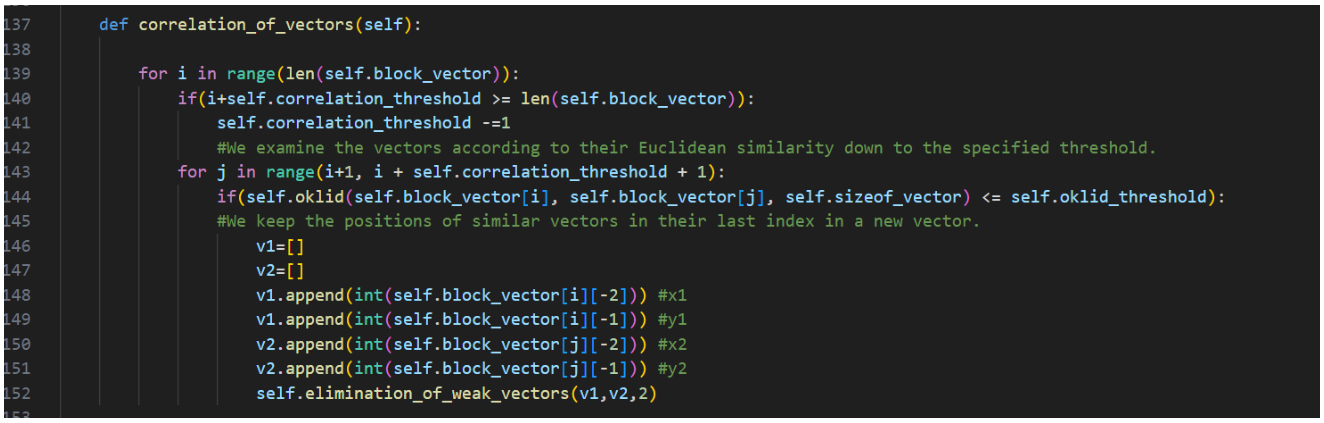
Storage: Adds the modified vector to block_vector for further processing.

Method lexicographically_sort_of_vectors()

Array Transformation: Converts block_vector to a NumPy array for sorting operations.

Sorting Logic: Applies lexicographical sort on the array, organizing vectors based on specific criteria such as relationships and coordinates.

Sorting Considerations: Sorts the array columns while considering coordinate information (excluded from sorting) placed in the last two lines during rotation, ensuring correct sorting based on vectors' characteristics.



```python
    def correlation_of_vectors(self):

        for i in range(len(self.block_vector)):
            if(i+self.correlation_threshold >= len(self.block_vector)):
                self.correlation_threshold -=1
                #We examine the vectors according to their Euclidean similarity down to the specified threshold.
            for j in range(i+1, i + self.correlation_threshold + 1):
                if(self.oklid(self.block_vector[i], self.block_vector[j], self.sizeof_vector) <= self.oklid_threshold):
                #We keep the positions of similar vectors in their last index in a new vector.
                    v1=[]
                    v2=[]
                    v1.append(int(self.block_vector[i][-2])) #x1
                    v1.append(int(self.block_vector[i][-1])) #y1
                    v2.append(int(self.block_vector[j][-2])) #x2
                    v2.append(int(self.block_vector[j][-1])) #y2
                    self.elimination_of_weak_vectors(v1,v2,2)
```

Explanation:-

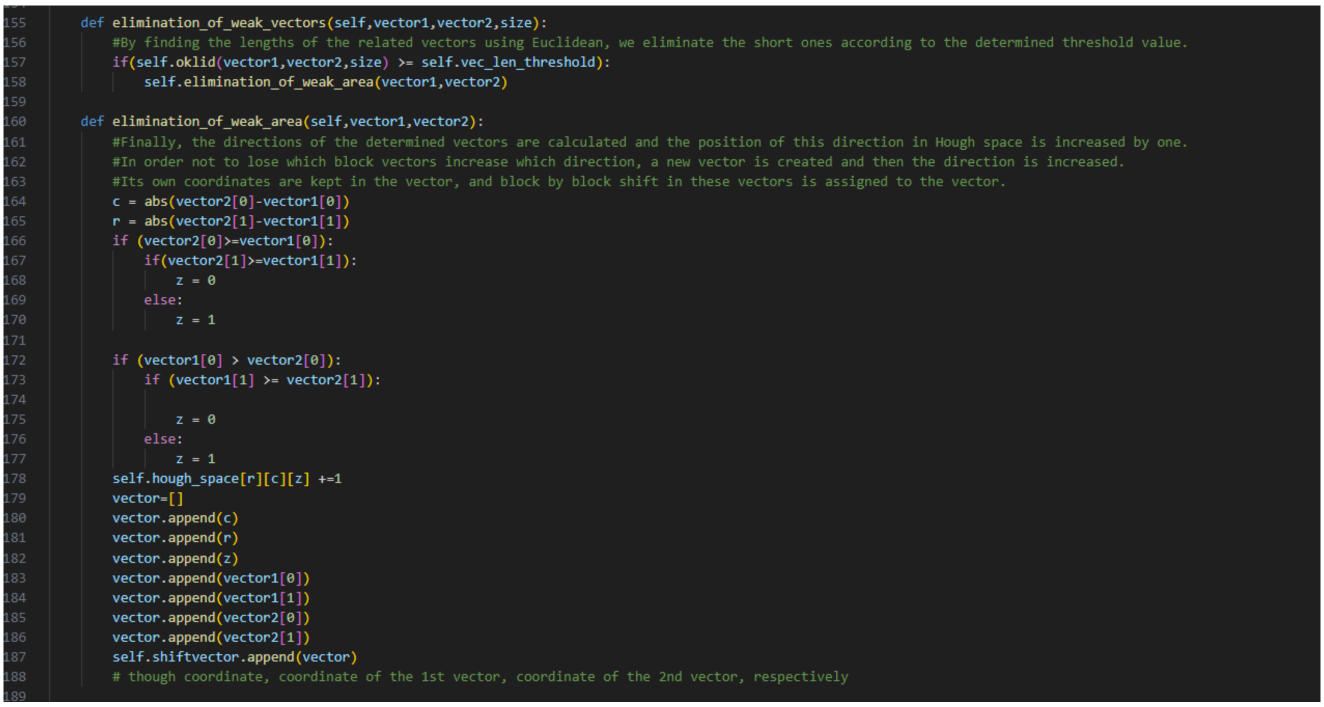
Method correlation_of_vectors()

Vector Comparison Loop: Iterates through each vector in block_vector for correlation analysis.

Threshold Adjustment: If nearing the end of the list, adjusts the correlation threshold to prevent exceeding the list length.

Similarity Calculation: Measures Euclidean similarity between vectors and identifies pairs meeting the threshold (oklid_threshold).

Coordinate Extraction: Extracts x and y coordinates from similar vectors and initiates elimination of weak vectors based on a specified threshold (vec_len_threshold) using elimination_of_weak_vectors() method.



```
    def elimination_of_weak_vectors(self,vector1,vector2,size):
        #By finding the lengths of the related vectors using Euclidean, we eliminate the short ones according to the determined threshold value.
        if(self.oklid(vector1,vector2,size) >= self.vec_len_threshold):
            self.elimination_of_weak_area(vector1,vector2)

    def elimination_of_weak_area(self,vector1,vector2):
        #Finally, the directions of the determined vectors are calculated and the position of this direction in Hough space is increased by one.
        #In order not to lose which block vectors increase which direction, a new vector is created and then the direction is increased.
        #Its own coordinates are kept in the vector, and block by block shift in these vectors is assigned to the vector.
        c = abs(vector2[0]-vector1[0])
        r = abs(vector2[1]-vector1[1])
        if (vector2[0]>=vector1[0]):
            if(vector2[1]>=vector1[1]):
                z = 0
            else:
                z = 1

        if (vector1[0] > vector2[0]):
            if (vector1[1] >= vector2[1]):

                z = 0
            else:
                z = 1
        self.hough_space[r][c][z] +=1
        vector=[]
        vector.append(c)
        vector.append(r)
        vector.append(z)
        vector.append(vector1[0])
        vector.append(vector1[1])
        vector.append(vector2[0])
        vector.append(vector2[1])
        self.shiftvector.append(vector)
        # though coordinate, coordinate of the 1st vector, coordinate of the 2nd vector, respectively
```

Explanation:-

Method elimination_of_weak_vectors()

Vector Length Check: Compares the lengths of vectors using Euclidean distance and eliminates those shorter than the specified threshold (vec_len_threshold).

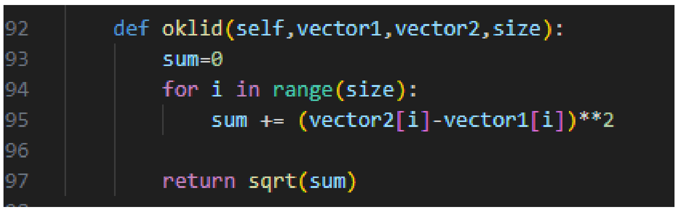
Elimination Trigger: Initiates the elimination process for weak vectors using elimination_of_weak_area() if the vector length meets the threshold.

Method elimination_of_weak_area()

Direction Calculation: Computes the directions based on the coordinates of two related vectors.

Hough Space Update: Updates the Hough space by increasing the count in the corresponding direction using coordinates obtained from the vectors.

Vector Formation: Creates a new vector storing directional information, including coordinates and shift positions, and appends it to shiftvector for further analysis.



```python
def oklid(self,vector1,vector2,size):
    sum=0
    for i in range(size):
        sum += (vector2[i]-vector1[i])**2

    return sqrt(sum)
```

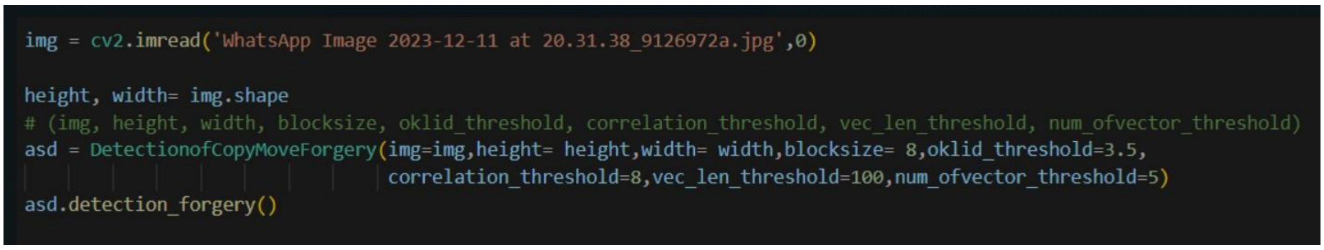
Explanation:-

The oklid() method calculates the Euclidean distance between two vectors, vector1 and vector2, based on a specified size.

Initialization: Initializes the sum variable to zero to accumulate the squared differences between corresponding elements of the vectors.

Distance Calculation: Iterates through each element in the vectors up to the specified size. For each element at index i, computes the squared difference between the corresponding elements of vector2 and vector1, adding this squared difference to the sum.

Euclidean Distance: Computes the square root of the accumulated sum of squared differences, returning the Euclidean distance as the final result.



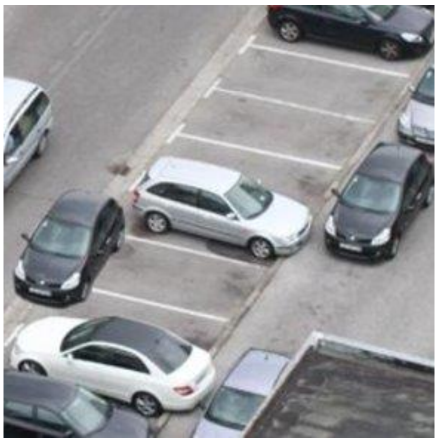
```python
img = cv2.imread('WhatsApp Image 2023-12-11 at 20.31.38_9126972a.jpg',0)

height, width= img.shape
# (img, height, width, blocksize, oklid_threshold, correlation_threshold, vec_len_threshold, num_ofvector_threshold)
asd = DetectionofCopyMoveForgery(img=img,height= height,width= width,blocksize= 8,oklid_threshold=3.5,
                                 correlation_threshold=8,vec_len_threshold=100,num_ofvector_threshold=5)
asd.detection_forgery()
```

Explanation:-

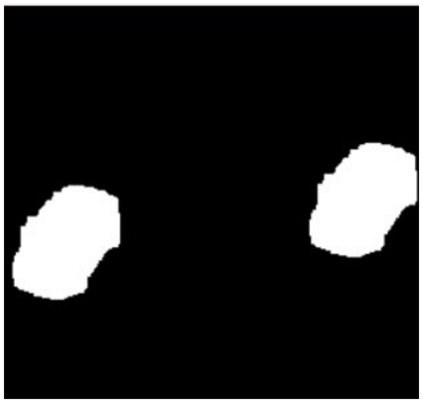
This part will reads the our image will process and gives the output.

OUTPUT:-

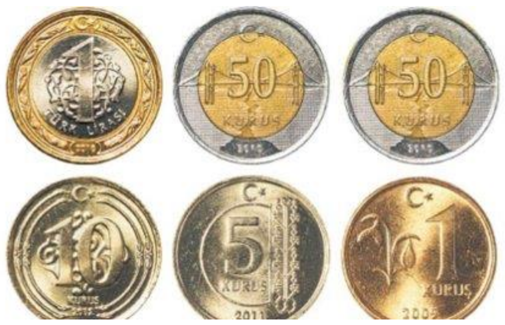
Forged image



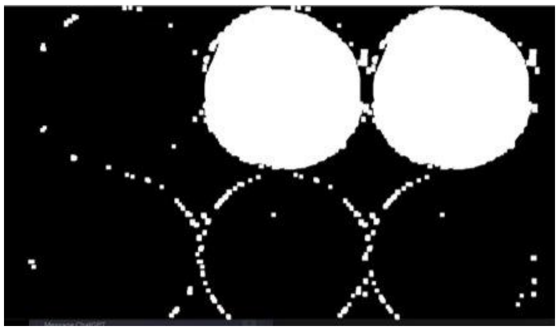
Detection



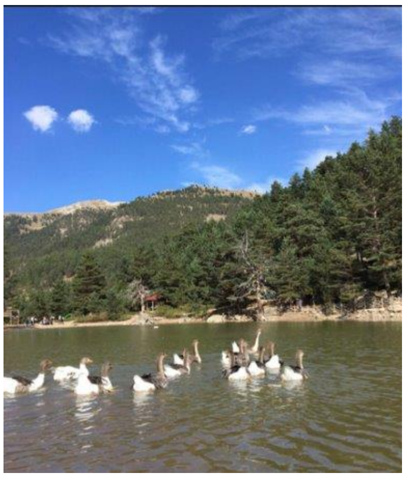
Forged Image



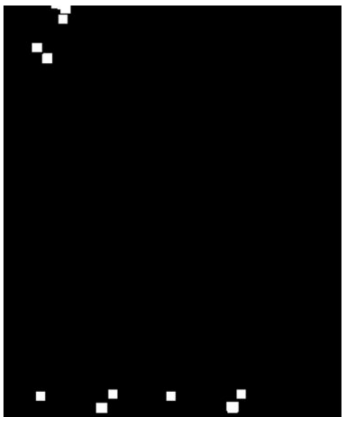
Detected



Forged Image



Detected



INFERENCE

The project detects potential copy-move forgeries within images by analyzing block-wise relationships, employing Discrete Cosine Transform (DCT) and lexicographical sorting for robustness against compression. It identifies similarities between blocks, tracks shift vectors, and highlights suspected tampered regions, offering a method to detect potential image manipulation, specifically copy-move forgeries, through vector analysis and correlation

CONCLUSION

In conclusion, the project demonstrates an effective method for detecting copy-move forgeries within images, leveraging block-wise analysis and vector comparisons to identify tampered regions while resisting image compression effects. By employing techniques like DCT, lexicographical sorting, and Euclidean similarity measures, it offers a robust approach to pinpoint potential areas of image manipulation, aiding in forensic analysis and ensuring image authenticity.

REFERENCES :-

https://infotec.repositorioinstitucional.mx/jspui/bitstream/1027/523/1/ArmasVega2021_Article_Copy-moveForgeryDetectionTechn.pdf

https://www.irjet.net/archives/V3/i6/IRJET-V3I6407.pdf

https://sci-hub.se/10.1007/s11042-019-08354-x

https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=e3c9c7d735c1fd8319e34fd3db37973c44802a68#page=693