Copy_Move_forgery_detection using DCT and Tkinter GUI

Premsai D 2022bcy0059
Kamalakar B 2022bcy0050
Sai Prithvi S 2022bcy0058

Abstract

This project implements a block-based copy–move forgery detection algorithm using Discrete Cosine Transform (DCT) features. A Tkinter-based GUI is integrated to allow users to upload images, run detection, and view both the original and processed results side by side. The system highlights duplicated regions in tampered images and saves the output automatically.

Introduction

Digital image forgery is a growing concern in multimedia security. One common technique is **copy–move forgery**, where a region of an image is copied and pasted elsewhere within the same image. Detecting such manipulations is crucial for image forensics. This project demonstrates a practical implementation of a DCT-based detection algorithm with a user-friendly GUI.

Methodology

1. **Preprocessing**: Input image is converted to grayscale.
   Block Division: The image is divided into overlapping blocks of fixed size.
2. **Feature Extraction**: DCT is applied to each block, and significant coefficients are retained.

3. **Vector Sorting**: Feature vectors are lexicographically sorted for efficient comparison.
4. **Correlation & Matching**: Similar blocks are identified using Euclidean distance thresholds.
5. **Forgery Localization**: Shift vectors are accumulated in a Hough space to identify consistent duplicated regions.
6. **Visualization**: Detected regions are highlighted with rectangles.

GUI Integration :

- **Tkinter** provides the graphical interface. **File Dialog** allows users to select images.
- **Side-by-Side Display** shows the original image and the detected forgery output simultaneously.
- **Dynamic saving images** stores the processed image as user choice of name

Results

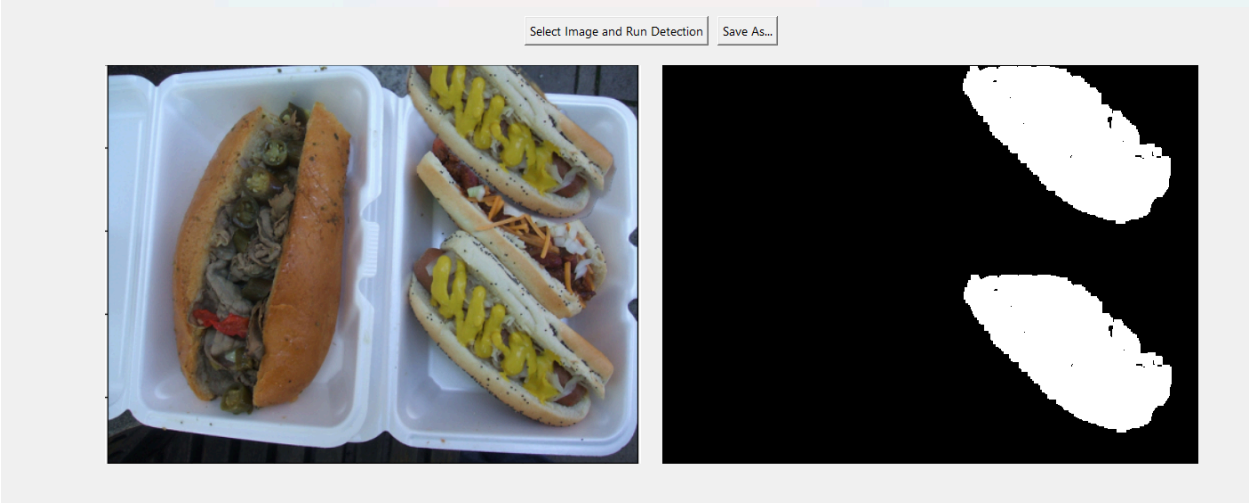
- The system successfully detects duplicated regions in tampered images. The GUI makes the tool accessible to non-technical users.
- Output images are saved for further analysis

Conclusion

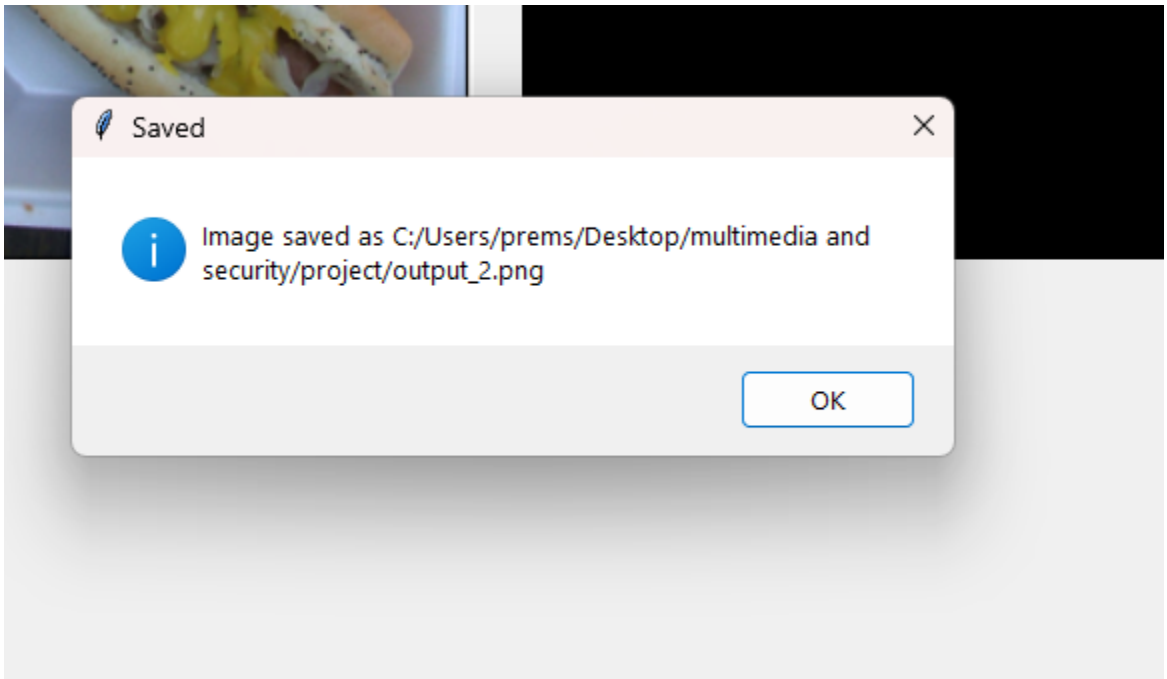
This project demonstrates a practical approach to detecting copy–move forgeries using DCT features and provides a simple GUI for usability. It bridges the gap between theoretical algorithms and real-world applications in multimedia security.

Example demo :

Input :



Output :
Saved as output_2.png



CODE :

```
import cv2
import numpy as np
from math import sqrt
import tkinter as tk
from tkinter import filedialog, messagebox
from PIL import Image, ImageTk

# ---------------- Detection Class ----------------
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 = np.zeros((self.height, self.width, 2))
        self.shiftvector = []

    def detection_forgery(self):
        self.dct_of_img()
        self.lexicographically_sort_of_vectors()
        self.correlation_of_vectors()
        max_val = -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_val:
                        max_val = 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_val - (max_val *
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.shiftvector[k][4] +
self.blocksize), (255), -1)
                                cv2.rectangle(self.img,
(self.shiftvector[k][5], self.shiftvector[k][6]),
                                              (self.shiftvector[k][5] +
self.blocksize,
                                               self.shiftvector[k][6] +
self.blocksize), (255), -1)
        return self.img

    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)
                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]])
                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)

    def zigzag(self, matrix):
        vector = []
        n = len(matrix) - 1
        i = 0
        j = 0
        for _ in range(n * 2):
            vector.append(matrix[i][j])
            if j == n:
                i += 1
                while i != n:
                    vector.append(matrix[i][j])
                    i += 1
                    j -= 1
            elif i == 0:
                j += 1
                while j != 0:
                    vector.append(matrix[i][j])
                    i += 1
                    j -= 1
            elif i == n:
                j += 1
                while j != n:
                    vector.append(matrix[i][j])
                    i -= 1
                    j += 1
            elif j == 0:
                i += 1
                while i != 0:
                    vector.append(matrix[i][j])
                    i -= 1
                    j += 1
                vector.append(matrix[i][j])
        return vector

    def significant_part_extraction(self, vector, x, y):
        del vector[self.sizeof_vector:(self.blocksize * self.blocksize)]
        vector.append(x)
```

```
        vector.append(y)
        self.block_vector.append(vector)

    def lexicographically_sort_of_vectors(self):
        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, :])]

    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
            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):
                    v1 = []
                    v2 = []
                    v1.append(int(self.block_vector[i][-2]))
                    v1.append(int(self.block_vector[i][-1]))
                    v2.append(int(self.block_vector[j][-2]))
                    v2.append(int(self.block_vector[j][-1]))
                    self.elimination_of_weak_vectors(v1, v2, 2)

    def elimination_of_weak_vectors(self, vector1, vector2, size):
        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):
        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)

    def oklid(self, vector1, vector2, size):
        sum_val = 0
        for i in range(size):
            sum_val += (vector2[i] - vector1[i]) ** 2
        return sqrt(sum_val)

# ---------------- GUI ----------------
def run_detection():
    filepath = filedialog.askopenfilename(filetypes=[("Image files",
"*.png;*.jpg;*.jpeg")])
    if not filepath:
        return
    img = cv2.imread(filepath, 0)
    if img is None:
        messagebox.showerror("Error", "Could not load image.")
        return
    height, width = img.shape
    detector = DetectionofCopyMoveForgery(img.copy(), height, width, 8,
3.5, 8, 100, 5)
    result = detector.detection_forgery()

    # Store result for Save As
    save_as.result_img = result

    # Convert both original and result to displayable format
    orig_rgb = cv2.cvtColor(cv2.imread(filepath), cv2.COLOR_BGR2RGB)
    res_rgb = cv2.cvtColor(result, cv2.COLOR_GRAY2RGB)
```

```
    orig_pil = Image.fromarray(orig_rgb)
    res_pil = Image.fromarray(res_rgb)

    orig_imgtk = ImageTk.PhotoImage(image=orig_pil)
    res_imgtk = ImageTk.PhotoImage(image=res_pil)

    panel_orig.config(image=orig_imgtk)
    panel_orig.image = orig_imgtk

    panel_res.config(image=res_imgtk)
    panel_res.image = res_imgtk

def save_as():
    outpath = filedialog.asksaveasfilename(defaultextension=".png",
                                           filetypes=[("PNG files",
"*.png"),
                                                      ("JPEG files",
"*.jpg")],
                                           title="Save Detected Image As")
    if outpath:
        # Save the last detected result
        cv2.imwrite(outpath, save_as.result_img)
        messagebox.showinfo("Saved", f"Image saved as {outpath}")


# ---------------- GUI Layout ----------------
root = tk.Tk()
root.title("Copy-Move Forgery Detection")

btn_frame = tk.Frame(root)
btn_frame.pack(pady=10)

btn_detect = tk.Button(btn_frame, text="Select Image and Run Detection",
command=run_detection)
btn_detect.pack(side="left", padx=5)

btn_save = tk.Button(btn_frame, text="Save As...", command=save_as)
btn_save.pack(side="left", padx=5)
```

```
# Frame to hold both images side by side
frame = tk.Frame(root)
frame.pack(padx=10, pady=10)

panel_orig = tk.Label(frame, text="Original Image")
panel_orig.pack(side="left", padx=10)

panel_res = tk.Label(frame, text="Detected Image")
panel_res.pack(side="right", padx=10)

root.mainloop()
```