```
from importlib.resources import path
import sys
from PIL import Image, ImageFilter, ImageDraw
import operator as op
from optparse import OptionParser
# import cv2


def Dist(p1, p2):
    x1, y1 = p1
    x2, y2 = p2
    return (((x1-x2)*(x1-x2)) + ((y1-y2)*(y1-y2)))**0.5


def intersectarea(p1, p2, size):
    x1, y1 = p1
    x2, y2 = p2
    ix1, iy1 = max(x1, x2), max(y1, y2)
    ix2, iy2 = min(x1+size, x2+size), min(y1+size, y2+size)
    iarea = abs(ix2-ix1)*abs(iy2-iy1)
    if iy2 < iy1 or ix2 < ix1:
        iarea = 0
    return iarea


def Hausdorff_distance(clust1, clust2, forward, dir):
    if forward == None:
```

```
        return max(Hausdorff_distance(clust1, clust2, True, dir), Hausdorff_distance(clust1, clust2, False,
dir))
    else:
        clstart, clend = (clust1, clust2) if forward else (clust2, clust1)
        dx, dy = dir if forward else (-dir[0], -dir[1])
        return sum([min([Dist((p1[0]+dx, p1[1]+dy), p2) for p2 in clend]) for p1 in clstart])/len(clstart)



def hassimilarcluster(ind, clusters, opt):
    item = op.itemgetter
    found = False
    tx = min(clusters[ind], key=item(0))[0]
    ty = min(clusters[ind], key=item(1))[1]
    for i, cl in enumerate(clusters):
        if i != ind:
            cx = min(cl, key=item(0))[0]
            cy = min(cl, key=item(1))[1]
            dx, dy = cx - tx, cy - ty
            specdist = Hausdorff_distance(clusters[ind], cl, None, (dx, dy))
            if specdist <= int(opt.rgsim):
                found = True
                break
    return found



def blockpoints(pix, coords, size):
    xs, ys = coords
    for x in range(xs, xs+size):
        for y in range(ys, ys+size):
```

```
            yield pix[x, y]


def colortopalette(color, palette):
    for a, b in palette:
        if color >= a and color < b:
            return b


def imagetopalette(image, palcolors):
    assert image.mode == 'L', "Only grayscale images supported !"
    pal = [(palcolors[i], palcolors[i+1]) for i in range(len(palcolors)-1)]
    image.putdata([colortopalette(c, pal) for c in list(image.getdata())])


def getparts(image, block_len, opt):
    img = image.convert('L') if image.mode != 'L' else image
    w, h = img.size
    parts = []
    # Bluring image for abandoning image details and noise.
    for n in range(int(opt.imblev)):
        img = img.filter(ImageFilter.SMOOTH_MORE)
    # Converting image to custom palette
    imagetopalette(img, [x for x in range(256) if x % int(opt.impalred) == 0])
    pix = img.load()

    for x in range(w-block_len):
        for y in range(h-block_len):
            data = list(blockpoints(pix, (x, y), block_len)) + [(x, y)]
```

```
            parts.append(data)
    parts = sorted(parts)
    return parts



def similarparts(imagparts, opt):
    dupl = []
    l = len(imagparts[0])-1

    for i in range(len(imagparts)-1):
        difs = sum(abs(x-y)
                for x, y in zip(imagparts[i][:l], imagparts[i+1][:l]))
        mean = float(sum(imagparts[i][:l])) / l
        dev = float(sum(abs(mean-val) for val in imagparts[i][:l])) / l
        if mean == 0:
            mean = .000000000001
        if dev/mean >= float(opt.blcoldev):
            if difs <= int(opt.blsim):
                if imagparts[i] not in dupl:
                    dupl.append(imagparts[i])
                if imagparts[i+1] not in dupl:
                    dupl.append(imagparts[i+1])

    return dupl



def clusterparts(parts, block_len, opt):
    parts = sorted(parts, key=op.itemgetter(-1))
    clusters = [[parts[0][-1]]]
```

```
# assign all parts to clusters
for i in range(1, len(parts)):
    x, y = parts[i][-1]

    # detect box already in cluster
    fc = []
    for k, cl in enumerate(clusters):
        for xc, yc in cl:
            ar = intersectarea((xc, yc), (x, y), block_len)
            intrat = float(ar)/(block_len*block_len)
            if intrat > float(opt.blint):
                if not fc:
                    clusters[k].append((x, y))
                fc.append(k)
                break

    # if this is new cluster
    if not fc:
        clusters.append([(x, y)])
    else:
        # re-clustering boxes if in several clusters at once
        while len(fc) > 1:
            clusters[fc[0]] += clusters[fc[-1]]
            del clusters[fc[-1]]
            del fc[-1]

item = op.itemgetter
# filter out small clusters
```

```
    clusters = [clust for clust in clusters if Dist((min(clust, key=item(0))[0], min(clust, key=item(
        1))[1]), (max(clust, key=item(0))[0], max(clust, key=item(1))[1]))/(block_len*1.4) >= float(opt.rgsize)]

    # filter out clusters, which doesn`t have identical twin cluster
    clusters = [clust for x, clust in enumerate(
        clusters) if hassimilarcluster(x, clusters, opt)]

    return clusters



def marksimilar(image, clust, size, opt):
    block_len = 15
    blocks = []
    if clust:
        draw = ImageDraw.Draw(image)
        mask = Image.new('RGB', (size, size), 'cyan')
        for cl in clust:
            for x, y in cl:
                im = image.crop((x, y, x+size, y+size))
                im = Image.blend(im, mask, 0.5)
                blocks.append((x, y, im))
        for bl in blocks:
            x, y, im = bl
            image.paste(im, (x, y, x+size, y+size))
        if int(opt.imauto):
            for cl in clust:
                cx1 = min([cx for cx, _ in cl])
                cy1 = min([cy for _, cy in cl])
                cx2 = max([cx for cx, _ in cl]) + block_len
```

```
            cy2 = max([cy for _, cy in cl]) + block_len
            draw.rectangle([cx1, cy1, cx2, cy2], outline="magenta")
    return image


def detect(path, opt, args):
    block_len = 15
    im = Image.open(path)
    lparts = getparts(im, block_len, opt)
    dparts = similarparts(lparts, opt)
    cparts = clusterparts(dparts, block_len, opt) if int(
        opt.imauto) else [[elem[-1] for elem in dparts]]
    im = marksimilar(im, cparts, block_len, opt)
    out = path.split('.')[0] + '_analyzed.jpg'
    im.show(out)
    # im.save(out)
    identical_regions = len(cparts) if int(opt.imauto) else 0
    # print('\tCopy-move output is saved in file -', out)
    return(identical_regions)

# def detect(input, opt, args):
#     block_len = 15
#     im = Image.open(input)
#     lparts = getparts(im, block_len, opt)
#     dparts = similarparts(lparts, opt)
#     cparts = clusterparts(dparts, block_len, opt) if int(
#         opt.imauto) else [[elem[-1] for elem in dparts]]
#     im = marksimilar(im, cparts, block_len, opt)
#     out = input.split('.')[0] + '_analyzed.jpg'
```

```
#     im.save(out)
#     identical_regions = len(cparts) if int(opt.imauto) else 0
#     print('\tCopy-move output is saved in file -', out)
#     return(identical_regions)



import numpy as np
# import pandas as pd
import cv2
# import argparse
# import csv
# import sys

from scipy import fftpack as fftp
from matplotlib import pyplot as plt



def detect(image):
    firstq = 30
    secondq = 40
    thres = 0.5

    dct_rows = 0
    dct_cols = 0

    image = cv2.imread(image)
    # image = cv2.imread(image)
    shape = image.shape
```

```
if shape[0] % 8 != 0:
    dct_rows = shape[0]+8-shape[0] % 8
else:
    dct_rows = shape[0]

if shape[1] % 8 != 0:
    dct_cols = shape[1]+8-shape[1] % 8
else:
    dct_cols = shape[1]

dct_image = np.zeros((dct_rows, dct_cols, 3), np.uint8)
dct_image[0:shape[0], 0:shape[1]] = image

y = cv2.cvtColor(dct_image, cv2.COLOR_BGR2YCR_CB)[:, :, 0]

w = y.shape[1]
h = y.shape[0]
n = w*h/64

Y = y.reshape(h//8, 8, -1, 8).swapaxes(1, 2).reshape(-1, 8, 8)

qDCT = []

for i in range(0, Y.shape[0]):
    qDCT.append(cv2.dct(np.float32(Y[i])))

qDCT = np.asarray(qDCT, dtype=np.float32)
qDCT = np.rint(qDCT - np.mean(qDCT, axis=0)).astype(np.int32)
f, a1 = plt.subplots(8, 8)
```

```
a1 = a1.ravel()


k = 0
# flag = True
for idx, ax in enumerate(a1):
    k += 1
    data = qDCT[:, int(idx/8), int(idx % 8)]
    val, key = np.histogram(data, bins=np.arange(data.min(), data.max()+1))
    # val, key = np.histogram(data, bins=np.arange(data.min(), data.max()+1), normed=True)
    z = np.absolute(fftp.fft(val))
    z = np.reshape(z, (len(z), 1))
    rotz = np.roll(z, int(len(z)/2))

    slope = rotz[1:] - rotz[:-1]
    indices = [i+1 for i in range(len(slope)-1)
               if slope[i] > 0 and slope[i+1] < 0]

    peak_count = 0

    for j in indices:
        if rotz[j][0] > thres:
            peak_count += 1

    if(k==3):
        if peak_count>=20: return True
        else: return False
        # flag = False
```

```
import cv2



# Encryption function
def encrypt():

    # img1 and img2 are the
    # two input images
    img1 = cv2.imread('1.jpg')
    img2 = cv2.imread('2.jpg')

    for i in range(img2.shape[0]):
        for j in range(img2.shape[1]):
            for l in range(3):

                # v1 and v2 are 8-bit pixel values
                # of img1 and img2 respectively
                v1 = format(img1[i][j][l], '08b')
                v2 = format(img2[i][j][l], '08b')

                # Taking 4 MSBs of each image
                v3 = v1[:4] + v2[:4]

                img1[i][j][l] = int(v3, 2)

    cv2.imwrite('3.png', img1)
```

encrypt()

```
from sklearn.cluster import DBSCAN
import numpy as np
import cv2


class Detect(object):
    def __init__(self, input):
        self.image = cv2.imread(input)

    def siftDetector(self):
        sift = cv2.SIFT_create()
        # sift = cv2.xfeatures2d.SIFT_create()
        gray = cv2.cvtColor(self.image, cv2.COLOR_BGR2GRAY)
        self.key_points, self.descriptors = sift.detectAndCompute(gray, None)
        return self.key_points, self.descriptors

    def showSiftFeatures(self):
        gray_image = cv2.cvtColor(self.image, cv2.COLOR_BGR2GRAY)
        sift_image = cv2.drawKeypoints(
            self.image, self.key_points, self.image.copy())
        return sift_image

    def locateForgery(self, eps=40, min_sample=2):
        clusters = DBSCAN(eps=eps, min_samples=min_sample).fit(
            self.descriptors)
        size = np.unique(clusters.labels_).shape[0]-1
        forgery = self.image.copy()
```

```
        if (size == 0) and (np.unique(clusters.labels_)[0] == -1):
            print('No Forgery Found!!')
            return None
        if size == 0:
            size = 1
        cluster_list = [[] for i in range(size)]
        for idx in range(len(self.key_points)):
            if clusters.labels_[idx] != -1:
                cluster_list[clusters.labels_[idx]].append(
                    (int(self.key_points[idx].pt[0]), int(self.key_points[idx].pt[1])))
        for points in cluster_list:
            if len(points) > 1:
                for idx1 in range(1, len(points)):
                    # Green color in BGR
                    cv2.line(forgery, points[0], points[idx1], (0, 255, 0), 5)
                    # cv2.line(forgery, points[0], points[idx1], (255, 0, 0), 5)
        return forgery

from sklearn.cluster import DBSCAN
import numpy as np
import cv2


class Detect(object):
    def __init__(self, input):
        self.image = cv2.imread(input)

    def siftDetector(self):
        sift = cv2.SIFT_create()
```

```
    # sift = cv2.xfeatures2d.SIFT_create()
    gray = cv2.cvtColor(self.image, cv2.COLOR_BGR2GRAY)
    self.key_points, self.descriptors = sift.detectAndCompute(gray, None)
    return self.key_points, self.descriptors

def showSiftFeatures(self):
    gray_image = cv2.cvtColor(self.image, cv2.COLOR_BGR2GRAY)
    sift_image = cv2.drawKeypoints(
        self.image, self.key_points, self.image.copy())
    return sift_image

def locateForgery(self, eps=40, min_sample=2):
    clusters = DBSCAN(eps=eps, min_samples=min_sample).fit(
        self.descriptors)
    size = np.unique(clusters.labels_).shape[0]-1
    forgery = self.image.copy()
    if (size == 0) and (np.unique(clusters.labels_)[0] == -1):
        print('No Forgery Found!!')
        return None
    if size == 0:
        size = 1
    cluster_list = [[] for i in range(size)]
    for idx in range(len(self.key_points)):
        if clusters.labels_[idx] != -1:
            cluster_list[clusters.labels_[idx]].append(
                (int(self.key_points[idx].pt[0]), int(self.key_points[idx].pt[1])))
    for points in cluster_list:
        if len(points) > 1:
            for idx1 in range(1, len(points)):
```

```
                # Green color in BGR
                cv2.line(forgery, points[0], points[idx1], (0, 255, 0), 5)
                # cv2.line(forgery, points[0], points[idx1], (255, 0, 0), 5)
        return forgery
from importlib.resources import path
from tkinter import *
from tkinter import filedialog, ttk, messagebox
from PIL import ImageTk, Image, ExifTags, ImageChops
from optparse import OptionParser
from datetime import datetime
from matplotlib import image
from prettytable import PrettyTable
import numpy as np
import random
import sys
import cv2
import re
import os
from PIL import Image
from PIL import ImageTk
from pyparsing import Opt
from ForgeryDetection import Detect
import double_jpeg_compression
import noise_variance
import copy_move_cfa


# Global variables
IMG_WIDTH = 400
```

```
IMG_HEIGHT = 400
uploaded_image = None

# copy-move parameters
cmd = OptionParser("usage: %prog image_file [options]")
cmd.add_option('', '--imauto',
        help='Automatically search identical regions. (default: %default)', default=1)
cmd.add_option('', '--imblev',
        help='Blur level for degrading image details. (default: %default)', default=8)
cmd.add_option('', '--impalred',
        help='Image palette reduction factor. (default: %default)', default=15)
cmd.add_option(
   '', '--rgsim', help='Region similarity threshold. (default: %default)', default=5)
cmd.add_option(
   '', '--rgsize', help='Region size threshold. (default: %default)', default=1.5)
cmd.add_option(
   '', '--blsim', help='Block similarity threshold. (default: %default)', default=200)
cmd.add_option('', '--blcoldev',
        help='Block color deviation threshold. (default: %default)', default=0.2)
cmd.add_option(
   '', '--blint', help='Block intersection threshold. (default: %default)', default=0.2)
opt, args = cmd.parse_args()
# if not args:
#    cmd.print_help()
#    sys.exit()


def getImage(path, width, height):
   """
```

```
    Function to return an image as a PhotoImage object
    :param path: A string representing the path of the image file
    :param width: The width of the image to resize to
    :param height: The height of the image to resize to
    :return: The image represented as a PhotoImage object
    """
    img = Image.open(path)
    img = img.resize((width, height), Image.LANCZOS)

    return ImageTk.PhotoImage(img)



def browseFile():
    """
    Function to open a browser for users to select an image
    :return: None
    """
    # Only accept jpg and png files
    filename = filedialog.askopenfilename(title="Select an image", filetypes=[("image", ".jpeg"),("image",
".png"),("image", ".jpg")])

    # No file selected (User closes the browsing window)
    if filename == "":
        return

    global uploaded_image

    uploaded_image = filename
```

```
    progressBar['value'] = 0   # Reset the progress bar
    fileLabel.configure(text=filename)     # Set the path name in the fileLabel

    # Display the input image in imagePanel
    img = Image.open(filename)
    img.thumbnail((400, 400))  # Resize the image to fit within 400x400
    img = ImageTk.PhotoImage(img)
    img = getImage(filename, IMG_WIDTH, IMG_HEIGHT)
    imagePanel.configure(image=img)
    imagePanel.image = img

    # Display blank image in resultPanel
    blank_img = Image.new('RGB', (400, 400), color='white')
    blank_img = ImageTk.PhotoImage(blank_img)
    blank_img = getImage("images/output.png", IMG_WIDTH, IMG_HEIGHT)
    resultPanel.configure(image=blank_img)
    resultPanel.image = blank_img

    # Reset the resultLabel
    resultLabel.configure(text="READY TO SCAN", foreground="green")


def copy_move_forgery():
    # Retrieve the path of the image file
    path = uploaded_image
    eps = 60
    min_samples = 2

    # User has not selected an input image
```

```
if path is None:
    # Show error message
    messagebox.showerror('Error', "Please select image")
    return

detect = Detect(path)
key_points, descriptors = detect.siftDetector()
forgery = detect.locateForgery(eps, min_samples)

# Set the progress bar to 100%
progressBar['value'] = 100

if forgery is None:
    # Retrieve the thumbs up image and display in resultPanel
    img = getImage("images/no_copy_move.png", IMG_WIDTH, IMG_HEIGHT)
    resultPanel.configure(image=img)
    resultPanel.image = img

    # Display results in resultLabel
    resultLabel.configure(text="ORIGINAL IMAGE", foreground="green")
else:
    # Retrieve the output image and display in resultPanel
    img = getImage("images/copy_move.png", IMG_WIDTH, IMG_HEIGHT)
    resultPanel.configure(image=img)
    resultPanel.image = img

    # Display results in resultLabel
    resultLabel.configure(text="Image Forged", foreground="red")
    # cv2.imshow('Original image', detect.image)
```

```
        cv2.imshow('Forgery', forgery)
        wait_time = 1000
        while(cv2.getWindowProperty('Forgery', 0) >= 0) or (cv2.getWindowProperty('Original image', 0) >=
0):
            keyCode = cv2.waitKey(wait_time)
            if (keyCode) == ord('q') or keyCode == ord('Q'):
                cv2.destroyAllWindows()
                break
            elif keyCode == ord('s') or keyCode == ord('S'):
                name = re.findall(r'(.+?)(\.[^.]*$|$)', path)
                date = datetime.today().strftime('%Y_%m_%d_%H_%M_%S')
                new_file_name = name[0][0]+'_'+str(eps)+'_'+str(min_samples)
                new_file_name = new_file_name+'_'+date+name[0][1]

                vaue = cv2.imwrite(new_file_name, forgery)
                print('Image Saved as....', new_file_name)



def metadata_analysis():
    # Retrieve the path of the image file
    path = uploaded_image
    # User has not selected an input image
    if path is None:
        # Show error message
        messagebox.showerror('Error', "Please select image")
        return

    img = Image.open(path)
    img_exif = img.getexif()
```

```
# Set the progress bar to 100%
progressBar['value'] = 100

if img_exif is None:
    # print('Sorry, image has no exif data.')
    # Retrieve the output image and display in resultPanel
    img = getImage("images/no_metadata.png", IMG_WIDTH, IMG_HEIGHT)
    resultPanel.configure(image=img)
    resultPanel.image = img

    # Display results in resultLabel
    resultLabel.configure(text="NO Data Found", foreground="red")
else:
    # Retrieve the thumbs up image and display in resultPanel
    img = getImage("images/metadata.png", IMG_WIDTH, IMG_HEIGHT)
    resultPanel.configure(image=img)
    resultPanel.image = img

    # Display results in resultLabel
    resultLabel.configure(text="Metadata Details", foreground="green")

    # print('image has exif data.')
    with open('Metadata_analysis.txt', 'w') as f:
        for key, val in img_exif.items():
            if key in ExifTags.TAGS:
                # print(f'{ExifTags.TAGS[key]} : {val}')
                    f.write(f'{ExifTags.TAGS[key]} : {val}\n')
    os.startfile('Metadata_analysis.txt')
```

```
def noise_variance_inconsistency():
    # Retrieve the path of the image file
    path = uploaded_image
    # User has not selected an input image
    if path is None:
        # Show error message
        messagebox.showerror('Error', "Please select image")
        return

    noise_forgery = noise_variance.detect(path)

    # Set the progress bar to 100%
    progressBar['value'] = 100

    if(noise_forgery):
        # print('\nNoise variance inconsistency detected')
        # Retrieve the output image and display in resultPanel
        img = getImage("images/varience.png", IMG_WIDTH, IMG_HEIGHT)
        resultPanel.configure(image=img)
        resultPanel.image = img

        # Display results in resultLabel
        resultLabel.configure(text="Noise variance", foreground="red")

    else:
        # Retrieve the thumbs up image and display in resultPanel
```

```
        img = getImage("images/no_varience.png", IMG_WIDTH, IMG_HEIGHT)
        resultPanel.configure(image=img)
        resultPanel.image = img

        # Display results in resultLabel
        resultLabel.configure(text="No Noise variance", foreground="green")

def cfa_artifact():
    # Retrieve the path of the image file
    path = uploaded_image
    # User has not selected an input image
    if path is None:
        # Show error message
        messagebox.showerror('Error', "Please select image")
        return

    identical_regions_cfa = copy_move_cfa.detect(path, opt, args)
    # identical_regions_cfa = copy_move_cfa.detect(path, opt, args)



    # Set the progress bar to 100%
    progressBar['value'] = 100

    # print('\n' + str(identical_regions_cfa), 'CFA artifacts detected')

    if(identical_regions_cfa):
        # Retrieve the output image and display in resultPanel
        img = getImage("images/cfa.png", IMG_WIDTH, IMG_HEIGHT)
        resultPanel.configure(image=img)
```

```
            resultPanel.image = img

            # Display results in resultLabel
            resultLabel.configure(text=f"{str(identical_regions_cfa)}, CFA artifacts detected", foreground="red")

        else:
            # print('\nSingle compressed')
            # Retrieve the thumbs up image and display in resultPanel
            img = getImage("images/no_cfa.png", IMG_WIDTH, IMG_HEIGHT)
            resultPanel.configure(image=img)
            resultPanel.image = img

            # Display results in resultLabel
            resultLabel.configure(text="NO-CFA artifacts detected", foreground="green")


    def ela_analysis():
        # Retrieve the path of the image file
        path = uploaded_image
        TEMP = 'temp.jpg'
        SCALE = 10

        # User has not selected an input image
        if path is None:
            # Show error message
            messagebox.showerror('Error', "Please select image")
            return

        original = Image.open(path)
```

```
    original.save(TEMP, quality=90)
    temporary = Image.open(TEMP)

    diff = ImageChops.difference(original, temporary)
    d = diff.load()
    WIDTH, HEIGHT = diff.size
    for x in range(WIDTH):
        for y in range(HEIGHT):
            d[x, y] = tuple(k * SCALE for k in d[x, y])

    # Set the progress bar to 100%
    progressBar['value'] = 100
    diff.show()



def jpeg_Compression():

    # Retrieve the path of the image file
    path = uploaded_image
    # User has not selected an input image
    if path is None:
        # Show error message
        messagebox.showerror('Error', "Please select image")
        return

    double_compressed = double_jpeg_compression.detect(path)

    # Set the progress bar to 100%
```

```
    progressBar['value'] = 100

    if(double_compressed):
        # print('\nDouble compression detected')
        # Retrieve the output image and display in resultPanel
        img = getImage("images/double_compression.png", IMG_WIDTH, IMG_HEIGHT)
        resultPanel.configure(image=img)
        resultPanel.image = img

        # Display results in resultLabel
        resultLabel.configure(text="Double compression", foreground="red")

    else:
        # print('\nSingle compressed')
        # Retrieve the thumbs up image and display in resultPanel
        img = getImage("images/single_compression.png", IMG_WIDTH, IMG_HEIGHT)
        resultPanel.configure(image=img)
        resultPanel.image = img

        # Display results in resultLabel
        resultLabel.configure(text="Single compression", foreground="green")
 #``````````````````````````````````````````````````````Written Code
def image_decode():
    # Retrieve the path of the image file
    path = uploaded_image
    # User has not selected an input image
    if path is None:
        # Show error message
        messagebox.showerror('Error', "Please select image")
```

```
    return

# Encrypted image
img = cv2.imread(path)
width = img.shape[0]
height = img.shape[1]

# img1 and img2 are two blank images
img1 = np.zeros((width, height, 3), np.uint8)
img2 = np.zeros((width, height, 3), np.uint8)

for i in range(width):
    for j in range(height):
        for l in range(3):
            v1 = format(img[i][j][l], '08b')
            v2 = v1[:4] + chr(random.randint(0, 1)+48) * 4
            v3 = v1[4:] + chr(random.randint(0, 1)+48) * 4

            # Appending data to img1 and img2
            img1[i][j][l]= int(v2, 2)
            img2[i][j][l]= int(v3, 2)

# Set the progress bar to 100%
progressBar['value'] = 100

# These are two images produced from
# the encrypted image
# cv2.imwrite('pic2_re.png', img1)
cv2.imwrite('output.png', img2)
```

```
    # Image.show(img2)
    # creating a object
    im = Image.open('output.png')
    im.show()

def string_analysis():
    # Retrieve the path of the image file
    path = uploaded_image
    # User has not selected an input image
    if path is None:
        # Show error message
        messagebox.showerror('Error', "Please select image")
        return

    x=PrettyTable()
    x.field_names = ["Bytes", "8-bit", "string"]
    # x.border = False
    with open(path, "rb") as f:
            n = 0
            b = f.read(16)

            while b:
                s1 = " ".join([f"{i:02x}" for i in b])  # hex string
                # insert extra space between groups of 8 hex values
                s1 = s1[0:23] + " " + s1[23:]

                # ascii string; chained comparison
                s2 = "".join([chr(i) if 32 <= i <= 127 else "." for i in b])
```

```
            # print(f"{n * 16:08x}  {s1:<48}  |{s2}|")
            x.add_row([f"{n * 16:08x}",f"{s1:<48}",f"{s2}"])

            n += 1
            b = f.read(16)

        # Set the progress bar to 100%
        progressBar['value'] = 100

        with open('hex_viewer.txt', 'w') as w:
            w.write(str(x))
            # w.write(f"{os.path.getsize(path):08x}")
        os.startfile('hex_viewer.txt')
        # print(f"{os.path.getsize(filename):08x}")




# Initialize the app window
root = Tk()
root.title("IMAGE FORGERY DETECTION SYSTEM ")
root.iconbitmap('images/favicon.ico')

# Set a fixed width for the window
window_width = 1400
root.state('zoomed')

#background color
```

```
root.configure(bg='white')
# Ensure the program closes when window is closed
root.protocol("WM_DELETE_WINDOW", root.quit)

# Maximize the size of the window
root.state("zoomed")

# Add the GUI into the Tkinter window
# GUI(parent=root)

root.grid_rowconfigure(0, weight=1)
root.grid_rowconfigure(1, weight=5)
root.grid_rowconfigure(2, weight=1)
root.grid_columnconfigure(0, weight=1)
root.grid_columnconfigure(1, weight=36)
root.grid_columnconfigure(2, weight=3)
root.grid_columnconfigure(3, weight=1)
root.grid_columnconfigure(4, weight=1)

# Label for the results of scan
resultLabel = Label(text="INPUT IMAGE TO START", bg='white',font=("Leelawadee UI", 30))
# resultLabel.pack(pady=100)
resultLabel.grid(row=20, column=0, columnspan=3)
# resultLabel.bg('black')

# resultLabel.grid(row=0, column=1, columnspan=2)

# Get the blank image
blank_img = Image.new('RGB', (400, 400), color='white')
```

```
blank_img = ImageTk.PhotoImage(blank_img)
input_img = getImage("images/input.png", IMG_WIDTH, IMG_HEIGHT)
middle_img = getImage("images/middle.png", IMG_WIDTH, IMG_HEIGHT)
output_img = getImage("images/output.png", IMG_WIDTH, IMG_HEIGHT)

# Displays the input image
imagePanel = Label(image=input_img)
imagePanel.image = input_img
imagePanel.grid(row=3, column=0, padx=5)

# Label to display the middle image
middle = Label(image=middle_img)
middle.image = middle_img
middle.grid(row=3, column=1, padx=5)

# Label to display the output image
resultPanel = Label(image=output_img)
resultPanel.image = output_img
resultPanel.grid(row=3, column=2, padx=5)

# Label to display the path of the input image
fileLabel = Label(text="No file selected", fg="black", font=("Leelawadee UI", 15))
fileLabel.grid(row=4, column=1)
# fileLabel.grid(row=2, column=0, columnspan=2)


# Progress bar
progressBar = ttk.Progressbar(length=500)
progressBar.grid(row=5, column=1)
```

```
# progressBar.grid(row=3, column=0, columnspan=2)



# Configure the style of the buttons
s = ttk.Style()
s.configure('my.TButton', font=('Leelawadee UI', 10))

# Button to upload images
uploadButton = ttk.Button(
   text="Upload Image", style="my.TButton", command=browseFile)
uploadButton.grid(row=6, column=1, sticky="nsew", pady=10)
# uploadButton.grid(row=4, column=0, columnspan=2, sticky="nsew", pady=5)

# Button to run the Compression detection algorithm
compression = ttk.Button(text="Compression-Detection",
                         style="my.TButton", command=jpeg_Compression)
compression.grid(row=7, column=0, columnspan=1, pady=20)
# startButton.grid(row=5, column=0, columnspan=2, sticky="nsew", pady=5)

# Button to run the Metadata-Analysis detection algorithm
metadata = ttk.Button(text="Metadata-Analysis",
                      style="my.TButton", command=metadata_analysis)
metadata.grid(row=7, column=0, columnspan=2, pady=20)

# Button to run the CFA-artifact detection algorithm
# artifact = ttk.Button(text="CFA-artifact detection", style="my.TButton", command=cfa_artifact)
# artifact.grid(row=5, column=1, columnspan=1, pady=20)

# Button to run the noise variance inconsistency detection algorithm
```

```
noise = ttk.Button(text="noise-inconsistency",
                   style="my.TButton", command=noise_variance_inconsistency)
noise.grid(row=7, column=1, columnspan=2, pady=20)

# Button to run the Copy-Move  detection algorithm
copy_move = ttk.Button(text="Copy-Move", style="my.TButton", command=copy_move_forgery)
copy_move.grid(row=7, column=2, columnspan=1, pady=20)

# Button to run the Error-Level Analysis algorithm
# ela = ttk.Button(text="Error-Level Analysis", style="my.TButton", command=ela_analysis)
# ela.grid(row=8, column=0, columnspan=2, pady=5)

# Button to run the Image pixel Analysis algorithm
image_stegnography = ttk.Button(text="Image-Extraction", style="my.TButton",
command=image_decode)
image_stegnography.grid(row=8, column=2, pady=5)

# Button to run the String Extraction Analysis algorithm
String_analysis = ttk.Button(text="String Extraction", style="my.TButton", command=string_analysis)
String_analysis.grid(row=8, column=0, columnspan=1, pady=20)

# Button to exit the program
style = ttk.Style()
style.configure('W.TButton', font = ('calibri', 10, 'bold'),foreground = 'red')

quitButton = ttk.Button(text="Exit program", style = 'W.TButton', command=root.quit)
quitButton.grid(row=7, column=1, pady=20)
# quitButton.grid(row=6, column=0, columnspan=2, sticky="e", pady=5)
```

```
root.geometry('')
# Open the GUI
root.mainloop()
# import binascii
# filename = '1.jpg'
# with open(filename, 'rb') as f:
#     content = f.read()
# print(f'{binascii.hexlify(content)}\n')

import sys
import argparse
import os.path
# This will import all the widgets
# and modules which are available in
# tkinter and ttk module
from tkinter import *
# from tkinter.ttk import *

from prettytable import PrettyTable

filename = '2.jpg'

# creates a Tk() object
master = Tk()

t=Text(master)#Inside t ext widget we would put our table

x=PrettyTable()
```

```
x.field_names = ["Bytes", "8-bit", "string"]



with open(filename, "rb") as f:
    n = 0
    b = f.read(16)

    while b:
        s1 = " ".join([f"{i:02x}" for i in b])  # hex string
        # insert extra space between groups of 8 hex values
        s1 = s1[0:23] + " " + s1[23:]

        # ascii string; chained comparison
        s2 = "".join([chr(i) if 32 <= i <= 127 else "." for i in b])

        # print(f"{n * 16:08x}  {s1:<48}  |{s2}|")
        # hex = Label(master,text= f"{n * 16:08x}  {s1:<48}  |{s2}|")
        # hex.pack()
        x.add_row([f"{n * 16:08x}",f"{s1:<48}",f"{s2}"])

        n += 1
        b = f.read(16)

    t.insert(INSERT,x)#Inserting table in text widget
    t.config(state=DISABLED)
    t.pack()
    # print(f"{os.path.getsize(filename):08x}")
    # function to open a new window on a button click.
    label = Label(master,text="This is the main window " f"{os.path.getsize(filename):08x}")
```

```
        label.pack(pady=10)

# mainloop, runs infinitely
mainloop()









# import sys
# import argparse
# import os.path



# parser = argparse.ArgumentParser()
# parser.add_argument(
#     "FILE", help="the name of the file that you wish to dump", type=str)
# args = parser.parse_args()

# try:
#     with open(args.FILE, "rb") as f:
#         n = 0
#         b = f.read(16)

#         while b:
```

```
#           s1 = " ".join([f"{i:02x}" for i in b])  # hex string
#           # insert extra space between groups of 8 hex values
#           s1 = s1[0:23] + " " + s1[23:]

#           # ascii string; chained comparison
#           s2 = "".join([chr(i) if 32 <= i <= 127 else "." for i in b])

#           print(f"{n * 16:08x}  {s1:<48}  |{s2}|")

#           n += 1
#           b = f.read(16)
#     print(f"{os.path.getsize(args.FILE):08x}")

# except Exception as e:
#     print(__file__, ": ", type(e).__name__, " - ", e, sep="", file=sys.stderr)
import sys
import cv2
from ForgeryDetection import Detect
import re
from datetime import datetime
import os.path as path
# from exif import Image

from PIL import Image, ExifTags

import double_jpeg_compression
import copy_move_cfa
import noise_variance
```

```
from optparse import OptionParser


# copy-move parameters
cmd = OptionParser("usage: %prog image_file [options]")
cmd.add_option('', '--imauto',
        help='Automatically search identical regions. (default: %default)', default=1)
cmd.add_option('', '--imblev',
        help='Blur level for degrading image details. (default: %default)', default=8)
cmd.add_option('', '--impalred',
        help='Image palette reduction factor. (default: %default)', default=15)
cmd.add_option(
   '', '--rgsim', help='Region similarity threshold. (default: %default)', default=5)
cmd.add_option(
   '', '--rgsize', help='Region size threshold. (default: %default)', default=1.5)
cmd.add_option(
   '', '--blsim', help='Block similarity threshold. (default: %default)', default=200)
cmd.add_option('', '--blcoldev',
        help='Block color deviation threshold. (default: %default)', default=0.2)
cmd.add_option(
   '', '--blint', help='Block intersection threshold. (default: %default)', default=0.2)
opt, args = cmd.parse_args()
if not args:
   cmd.print_help()
   sys.exit()



def PrintBoundary():
   for i in range(50):
      print('*', end='')
```

```
    print()



file_name = sys.argv[1]
input = './/input//' + file_name
if not path.exists(input):
    sys.exit(
        "Image not found: {}. Please place the image in the images subdirectory.".format(file_name))




# double jpeg compression detection Start
PrintBoundary()
print('\nRunning double jpeg compression detection...')
double_compressed = double_jpeg_compression.detect(input)

if(double_compressed):
    print('\nDouble compression detected')
else:
    print('\nSingle compressed')
PrintBoundary()
# double jpeg compression detection End



# Metadata Analysis detection Start
PrintBoundary()
print('\nRunning Metadata Analysis detection')
```

```
img = Image.open(input)
img_exif = img.getexif()

if img_exif is None:
    print('Sorry, image has no exif data.')
else:
    for key, val in img_exif.items():
        if key in ExifTags.TAGS:
            print(f'{ExifTags.TAGS[key]} : {val}')
PrintBoundary()
# Metadata Analysis detection End



# # CFA artifact detection Start
# PrintBoundary()
# print('\nRunning CFA artifact detection...\n')
# identical_regions_cfa = copy_move_cfa.detect(input, opt, args)
# print('\n' + str(identical_regions_cfa), 'CFA artifacts detected')
# PrintBoundary()
# # CFA artifact detection End




# noise variance inconsistency detection Start
PrintBoundary()
print('\nRunning noise variance inconsistency detection...')
noise_forgery = noise_variance.detect(input)

if(noise_forgery):
```

```
    print('\nNoise variance inconsistency detected')
else:
    print('\nNo noise variance inconsistency detected')
PrintBoundary()
# noise variance inconsistency detection Start




# Copy-Move detection Start


eps = 60
min_samples = 2

PrintBoundary()
print('Use \'q\' for exit and\n\'s/S\' for saving the Forgery Detected.')
PrintBoundary()
flag = True

try:
    value = sys.argv[2]

except IndexError:
    flag = False
if flag:
    try:
        value = int(value)
        if(value < 0 or value > 500):
            print('Value not in range (0,500)..... using default value.')
```

```
        else:
            eps = value
    except ValueError:
        print('Value not integer..... using default value.')

flag2 = True
try:
    value = sys.argv[3]
except IndexError:
    flag2 = False

if flag2:
    try:
        value = int(value)
        if(value < 0 or value > 50):
            print('Value not in range (0,50)..... using default value.')
        else:
            min_samples = value
    except ValueError:
        print('Value not integer..... using default value.')

PrintBoundary()
print('Detecting Copy-Move Forgery with parameter value as\neps:{}\nmin_samples:{}'.format(
    eps, min_samples))
PrintBoundary()

detect = Detect(input)

key_points, descriptors = detect.siftDetector()
```

```
forgery = detect.locateForgery(eps, min_samples)
if forgery is None:
    sys.exit(0)
cv2.imshow('Original image', detect.image)
cv2.imshow('Forgery', forgery)
wait_time = 1000
while(cv2.getWindowProperty('Forgery', 0) >= 0) or (cv2.getWindowProperty('Original image', 0) >= 0):
    keyCode = cv2.waitKey(wait_time)
    if (keyCode) == ord('q') or keyCode == ord('Q'):
        cv2.destroyAllWindows()
        break
    elif keyCode == ord('s') or keyCode == ord('S'):
        name = re.findall(r'(.+?)(\.[^.]*$|$)', file_name)
        date = datetime.today().strftime('%Y_%m_%d_%H_%M_%S')
        new_file_name = name[0][0]+'_'+str(eps)+'_'+str(min_samples)
        new_file_name = new_file_name+'_'+date+name[0][1]
        PrintBoundary()

        vaue = cv2.imwrite(new_file_name, forgery)
        print('Image Saved as....', new_file_name)

cv2.destroyAllWindows()
# Copy-Move detection End




# if ((not double_compressed) and (identical_regions_cfa == 0) and (not noise_forgery)):
#     print('\nNo forgeries were detected - this image has probably not been tampered with.')
```

```
# else:
#     print('\nSome forgeries were detected - this image may have been tampered with.')
import sys
import math
import numpy as np

from PIL import Image
from scipy import signal
from sklearn.cluster import KMeans

def estimate_noise(I):
    H, W = I.shape

    M = [[1, -2, 1], [-2, 4, -2], [1, -2, 1]]

    sigma = np.sum(np.sum(np.absolute(signal.convolve2d(I, M))))
    sigma = sigma * math.sqrt(0.5 * math.pi) / (6 * (W-2) * (H-2))

    return sigma

def detect(input, blockSize=32):
    im = Image.open(input)
    im = im.convert('1')

    blocks = []

    imgwidth, imgheight = im.size

    # break up image into NxN blocks, N = blockSize
```

```
    for i in range(0,imgheight,blockSize):
        for j in range(0,imgwidth,blockSize):
            box = (j, i, j+blockSize, i+blockSize)
            b = im.crop(box)
            a = np.asarray(b).astype(int)
            blocks.append(a)

    variances = []
    for block in blocks:
        variances.append([estimate_noise(block)])

    kmeans = KMeans(n_clusters=2, random_state=0).fit(variances)
    center1, center2 = kmeans.cluster_centers_

    if abs(center1 - center2) > .4: return True
    else: return False
```

THIS IS ONE PROJECT

```
import matplotlib.pyplot as plt
import numpy as np
from skimage import io

# Filenames of the original and altered images
original_image_filename = "ori.jpeg"
altered_image_filename = "alt.jpeg"
```

```
# Load original and altered images
original_image = io.imread(original_image_filename)
altered_image = io.imread(altered_image_filename)

# Compute absolute difference
diff_image = np.abs(original_image - altered_image)

# Plot original, altered, and difference images
plt.figure(figsize=(15, 5))

plt.subplot(1, 3, 1)
plt.imshow(original_image)
plt.title('Original Image')

plt.subplot(1, 3, 2)
plt.imshow(altered_image)
plt.title('Altered Image')

plt.subplot(1, 3, 3)
plt.imshow(diff_image, cmap='gray')
plt.title('Difference Image')

plt.show()



from PIL import Image

# Function to convert text to binary
def text_to_binary(text):
```

```
    # Convert each character in the text to its 8-bit binary representation
    binary_data = ''.join(format(ord(char), '08b') for char in text)
    return binary_data

# Function to hide text in an image
def hide_text_in_image(image_path, output_path, text_file_path, delimiter="1111111111111110"):
    # Open the original image
    img = Image.open(image_path)

    # Ensure the image is in RGB mode
    img = img.convert('RGB')

    # Read text from the file
    with open(text_file_path, 'r') as file:
        text_to_hide = file.read()

    # Convert the text to binary
    binary_text = text_to_binary(text_to_hide)

    # Add the delimiter to the binary text
    binary_text_with_delimiter = binary_text + delimiter

    # Get the pixels of the original image
    pixels = list(img.getdata())

    # Create a list to store the modified pixels
    new_pixels = []

    # Initialize the index for the binary text
```

```
    text_index = 0

    # Iterate through each pixel in the image
    for pixel in pixels:
        # Create a new pixel with the same RGB values as the original pixel
        new_pixel = list(pixel)

        # Iterate through each color channel (RGB)
        for i in range(3):
            # Check if there are more bits to encode
            if text_index < len(binary_text_with_delimiter):
                # Modify the least significant bit of the color channel with a bit from the binary text
                new_pixel[i] = (pixel[i] & 0b11111110) | int(binary_text_with_delimiter[text_index], 2)
                text_index += 1
            else:
                break  # Break the inner loop if text is fully encoded

        # Append the modified pixel to the list
        new_pixels.append(tuple(new_pixel))

    # Create a new image with the modified pixels
    new_img = Image.new('RGB', img.size)
    new_img.putdata(new_pixels)

    # Save the new image
    new_img.save(output_path)

# Main function
if __name__ == "__main__":
```

```
    # Input file paths
    image_path = "ori.jpeg"
    output_path = "alt.jpeg"
    text_file_path = "text_to_hide.txt"

    # Call the function to hide text in the image
    hide_text_in_image(image_path, output_path, text_file_path)



from PIL import Image

# Function to convert binary to text
def binary_to_text(binary_data):
    # Convert binary data to text by grouping 8 bits at a time and converting to ASCII
    text = ''.join([chr(int(binary_data[i:i+8], 2)) for i in range(0, len(binary_data), 8)])
    return text

# Function to extract text from an image using LSB (Least Significant Bit) encoding
def extract_text_from_image(image_path, output_file_path, delimiter="1111111111111110"):
    # Open the encoded image
    img = Image.open(image_path)
    img = img.convert('RGB')  # Ensure the image is in RGB mode

    # Get the pixel values of the image
    pixels = list(img.getdata())

    # Extract the binary text from the LSB of each color channel
    binary_text = ''.join([str(pixel[i] & 1) for pixel in pixels for i in range(3)])
```

```
    # Find the index of the delimiter in the binary text
    delimiter_index = binary_text.find(delimiter)

    # Extract binary text before the delimiter
    binary_text = binary_text[:delimiter_index]

    # Convert binary text to readable text
    extracted_text = binary_to_text(binary_text)

    # Write the extracted text to an external file
    with open(output_file_path, 'w') as file:
        file.write(extracted_text)

# Main function
if __name__ == "__main__":
    # Input file paths
    encoded_image_path = "alt.jpeg"
    output_file_path = "extracted_text.txt"

    # Call the function to extract text from the encoded image
    extract_text_from_image(encoded_image_path, output_file_path)

    # Print a message indicating the success of the extraction
    print("Extracted Text has been written to:", output_file_path)

from skimage import io, color
from skimage.metrics import structural_similarity as ssim

def compare_images(original_path, altered_path):
```

```
    # Load images
    original_image = io.imread(original_path)
    altered_image = io.imread(altered_path)

    # Convert images to grayscale
    original_gray = color.rgb2gray(original_image)
    altered_gray = color.rgb2gray(altered_image)

    # Compute structural similarity index
    similarity_index, _ = ssim(original_gray, altered_gray, full=True, data_range=altered_gray.max() - altered_gray.min())

    return similarity_index

if __name__ == "__main__":
    # Paths to the original and altered images
    original_image_path = "ori.jpeg"
    altered_image_path = "alt.jpeg"

    # Compare images
    similarity_index = compare_images(original_image_path, altered_image_path)

    # Print the similarity index
    print(f"Structural Similarity Index: {similarity_index}")

    # You can set a threshold to determine if the image is altered based on the similarity index
    threshold = 0.9  # Adjust this threshold as needed
    if similarity_index < threshold:
        print("The image may have been altered.")
```

```
    else:
        print("The image appears to be unchanged.")
```

THIS IS SECOND PROJECT