ZEAL EDUCATION SOCIETY's

ZEAL COLLEGE OF ENGINEEIRNG AND RESEARCH, NARHE, PUNE

DEPARTMENT OF COMPUTER ENGINEERING

SEMESTER-I

[A.Y. : 2022 - 2023]

CYBER SECURITY AND DIGITAL FORENSICS(410244(C))

MINI PROJECT

Design and develop a tool for digital forensics of images

Institute and Department Vision and Mission

INSTITUTE VISION	To impart value added technological education through pursuit of academic excellence, research and entrepreneurial attitude.
INSTITUTE MISSION	**M1:** To achieve academic excellence through innovative teaching and learning process. **M2:** To imbibe the research culture for addressing industry and societal needs**.** **M3:** To provide conducive environment for building the entrepreneurial skills. **M4:** To produce competent and socially responsible professionals with core human values.

DEPARTMENT VISION	To emerge as a department of repute in Computer Engineering which produces competent professionals and entrepreneurs to lead technical and betterment of mankind.
DEPARTMENT MISSION	**M1:** To strengthen the theoretical and practical aspects of the learning process by teaching applications and hands on practices using modern tools and FOSS technologies. **M2:** To endeavour innovative interdisciplinary research and entrepreneurship skills to serve the needs of Industry and Society. **M3:** To enhance industry academia dialog enabling students to inculcate professional skills. **M4:** To incorporate social and ethical awareness among the students to make them conscientious professionals.

Department
Program Educational Objectives(PEOs)

PEO1: To Impart fundamentals in science, mathematics and engineering to cater the needs of society and Industries.
PEO2: Encourage graduates to involve in research, higher studies, and/or to become entrepreneurs.
PEO3: To Work effectively as individuals and as team members in a multidisciplinary environment with high ethical values for the benefit of society.

Savitribai Phule Pune University Fourth Year of Computer Engineering (2019 Course) 410244(C): Cyber Security & Digital Forensics Laboratory		
Teaching Scheme:	**Credit**	**Examination Scheme:**
PR: 04 Hours/Week	02	TW: 25 Marks PR: 50 Marks

Course Objectives:

- To enhance awareness cyber forensics.
- To understand issues in Cyber Crime and different attacks
- To understand underlying principles and many of the techniques associated with the digital forensic practices
- To know the process and methods of evidence collection
- To analyze and validate forensic data collected.
- To apply digital forensic knowledge to use computer forensic tools and investigation report writing.

Course Outcomes:

On completion of the course, student will be able to-

CO1: Analyze threats in order to protect or defend it in cyberspace from cyber-attacks.
CO2: Build appropriate security solutions against cyber-attacks.
CO3: Underline the need of digital forensic and role of digital evidences.
CO4: Explain rules and types of evidence collection
CO5: Analyze, validate and process crime scenes CO6: Identify the methods to generate legal evidence and supporting investigation reports.

List of Assignments

Group B	
Mini-Projects/ Case Study (Any two)	
01	Mini Project- Design and develop a tool for digital forensics of images
02	Mini Project- Design and develop a tool for digital forensics of audio
03	Mini Project- Design and develop a tool for digital forensics of video
04	Mini Project- Design a system for the analysis of cyber crime using various cyber forensics techniques and compare each technique with respect to integrity, confidentiality, availability

Title:

Design and develop a tool for digital forensics of images

Requirements:

opencv-python
Pillow
exifread
scipy
PyWavelets
numpy
progressbar2
matplotlib

Code:

```
#! /usr/bin/env python2

# Copyright (C) Anh Duy TRAN

import numpy as np
import numpy.matlib as npm
import argparse
import json
import pprint
import exifread
import cv2 as cv
import os
import pywt
import math
import progressbar
import warnings
from scipy import ndimage
from PIL import Image
from PIL.ExifTags import TAGS, GPSTAGS
from matplotlib import pyplot as plt
from os.path import basename


def main():
    argparser = argparse.ArgumentParser(description="Digital Image Forensics")

    #argparser.add_argument("-e", help='export EXIF to XML')

    argparser.add_argument("datafile", metavar='file',
                           help='name of the image file')

    argparser.add_argument("-e", "--exif", help="exposing digital forgeries by EXIF metadata",
                           action="store_true")

    argparser.add_argument("-gm", "--jpegghostm", help="exposing digital forgeries by JPEG Ghost (Multiple)",
                           action="store_true")
```

```
    argparser.add_argument("-g", "--jpegghost", help="exposing digital forgeries by JPEG Ghost",
                           action="store_true")
    argparser.add_argument(
        "-n1", "--noise1", help="exposing digital forgeries by using noise inconsistencies", action="store_true")
    argparser.add_argument(
        "-n2", "--noise2", help="exposing digital forgeries by using Median-filter noise residue inconsistencies",
action="store_true")
    argparser.add_argument(
        "-el", "--ela", help="exposing digital forgeries by using Error Level Analysis", action="store_true")
    argparser.add_argument(
        "-cf", "--cfa", help="Image tamper detection based on demosaicing artifacts", action="store_true")
    argparser.add_argument("-q", "--quality", help="resaved image quality",
                           type=int)
    argparser.add_argument("-s", "--blocksize", help="block size kernel mask",
                           type=int)
    # Parses arguments
    args = argparser.parse_args()

    if check_file(args.datafile) == False:
        print("Invalid file. Please make sure the file is exist and the type is JPEG")
        return


    if args.exif:
        exif_check(args.datafile)

    elif args.jpegghostm:
        jpeg_ghost_multiple(args.datafile)
    elif args.jpegghost:
        jpeg_ghost(args.datafile, args.quality)


    elif args.noise1:
        noise_inconsistencies(args.datafile, args.blocksize)

    elif args.noise2:
        median_noise_inconsistencies(args.datafile, args.blocksize)

    elif args.ela:
        ela(args.datafile, args.quality, args.blocksize)


    elif args.cfa:
        cfa_tamper_detection(args.datafile)
    else:
        exif_check(args.datafile)


def check_file(data_path):
    if os.path.isfile(data_path) == False:
        return False
    if data_path.lower().endswith(('.jpg', '.jpeg')) == False:
        return False
```

```
    return True

def exif_check(file_path):
    # Open image file for reading (binary mode)
    f = open(file_path, 'rb')

    # Return Exif tags
    tags = exifread.process_file(f)

    # Get the pure EXIF data of Image
    exif_code_form = extract_pure_exif(file_path)
    if exif_code_form == None:
        print("The EXIF data has been stripped. Photo maybe is taken from facebook, twitter, imgur")
        return

    # Check Modify Date
    check_software_modify(exif_code_form)
    check_modify_date(exif_code_form)
    check_original_date(exif_code_form)
    check_camera_information(tags)
    check_gps_location(exif_code_form)
    check_author_copyright(exif_code_form)

    # Print Raw Image Metadata
    print("\nRAW IMAGE METADATA")
    print("============================================================== \n")
    print("EXIF Data")
    # pprint.pprint(decode_exif_data(exif_code_form))
    for tag in tags.keys():
        if tag not in ('JPEGThumbnail', 'TIFFThumbnail', 'Filename', 'EXIF MakerNote'):
            print("%-35s:  %s" % (tag, tags[tag]))


def extract_pure_exif(file_name):
    img = Image.open(file_name)
    info = img._getexif()
    return info


def decode_exif_data(info):
    exif_data = {}
    if info:
        for tag, value in info.items():
            decoded = TAGS.get(tag, tag)
            exif_data[decoded] = value

    return exif_data


def get_if_exist(data, key):
    if key in data:
        return data[key]
    return None
```

```
def export_json(data):
    with open('data.txt', 'w') as outfile:

# Check Software Edit
def check_software_modify(info):
    software = get_if_exist(info, 0x0131)
    if software != None:
        print("Image edited with: %s" % software)
        return True
    return False
def check_modify_date(info):
    modify_date = get_if_exist(info, 0x0132)
    if modify_date != None:
        print("Photo has been modified since it was created. Modified: %s" %
              modify_date)
        return True
    return False
def check_original_date(info):
    original_date = get_if_exist(info, 0x9003)
    create_date = get_if_exist(info, 0x9004)
    if original_date != None:
        print("The shutter actuation time: %s" % original_date)
    if create_date != None:
        print("Image created at: %s" % create_date)
def check_camera_information_2(info):
    make = get_if_exist(info, 0x010f)
    model = get_if_exist(info, 0x0110)
    exposure = get_if_exist(info, 0x829a)
    aperture = get_if_exist(info, 0x829d)
    focal_length = get_if_exist(info, 0x920a)
    iso_speed = get_if_exist(info, 0x8827)
    flash = get_if_exist(info, 0x9209)

    print("\nCamera Infomation")
    print("Make: \t \t %s" % make)
    print("Model: \t \t %s" % model)
    #print("Exposure: \t \t %s " % exposure)
    #print("Aperture: \t \t %s" % aperture)
    #print("Focal Length: \t \t %s" % focal_length)
    print("ISO Speed: \t %s" % iso_speed)
    print("Flash: \t \t %s" % flash)
def check_camera_information(info):
    make = get_if_exist(info, 'Image Make')
    model = get_if_exist(info, 'Image Model')
    exposure = get_if_exist(info, 'EXIF ExposureTime')
    aperture = get_if_exist(info, 'EXIF ApertureValue')
    focal_length = get_if_exist(info, 'EXIF FocalLength')
    iso_speed = get_if_exist(info, 'EXIF ISOSpeedRatings')
    flash = get_if_exist(info, 'EXIF Flash')
    print("\nCamera Infomation")
    print("------------------------------------------------------------ ")
```

```
    print("Make: \t \t %s" % make)
    print("Model: \t \t %s" % model)
    print("Exposure: \t %s " % exposure)
    print("Aperture: \t %s" % aperture)
    print("Focal Length: \t %s mm" % focal_length)
    print("ISO Speed: \t %s" % iso_speed)
    print("Flash: \t \t %s" % flash)
def check_gps_location(info):
    gps_info = get_if_exist(info, 0x8825)

    print("\nLocation (GPS)")
    print("------------------------------------------------------------ ")
    if gps_info == None:
        print("GPS coordinates not found")
        return False
    # print gps_info
    lat = None
    lng = None
    gps_latitude = get_if_exist(gps_info, 0x0002)
    gps_latitude_ref = get_if_exist(gps_info, 0x0001)
    gps_longitude = get_if_exist(gps_info, 0x0004)
    gps_longitude_ref = get_if_exist(gps_info, 0x0003)
    if gps_latitude and gps_latitude_ref and gps_longitude and gps_longitude_ref:
        lat = convert_to_degress(gps_latitude)
        if gps_latitude_ref != "N":
            lat = 0 - lat
        lng = convert_to_degress(gps_longitude)
        if gps_longitude_ref != "E":
            lng = 0 - lng

    print("Latitude \t %s North" % lat)
    print("Longtitude \t %s East" % lng)
return True
def convert_to_degress(value):
    """Helper function to convert the GPS coordinates
    stored in the EXIF to degress in float format"""
    d = float(value[0])
    m = float(value[1])
    s = float(value[2])
    return d + (m / 60.0) + (s / 3600.0)
def check_author_copyright(info):
    author = get_if_exist(info, 0x9c9d)
    copyright_tag = get_if_exist(info, 0x8298)
    profile_copyright = get_if_exist(info, 0xc6fe)
    print("\nAuthor and Copyright")
    print("------------------------------------------------------------ ")
    print("Author \t \t %s " % author)
    print("Copyright \t %s " % copyright_tag)
    print("Profile: \t %s" % profile_copyright)
def jpeg_ghost_multiple(file_path):

    print("Analyzing...")
    bar = progressbar.ProgressBar(maxval=20,
```

```
                    widgets=[progressbar.Bar('=', '[', ']'), ' ', progressbar.Percentage()])
bar.start()

img = cv.imread(file_path)
img_rgb = img[:, :, ::-1]

# Quality of the reasaved images
quality = 60

# Size of the block
smoothing_b = 17
offset = int((smoothing_b-1)/2)

# Size of the image
height, width, channels = img.shape

# Plot the original image
plt.subplot(5, 4, 1), plt.imshow(img_rgb), plt.title('Original')
plt.xticks([]), plt.yticks([])

# Get the name of the image
base = basename(file_path)
file_name = os.path.splitext(base)[0]
save_file_name = file_name+"_temp.jpg"
bar.update(1)

# Try 19 different qualities
for pos_q in range(19):

    # Resaved the image with the new quality
    encode_param = [int(cv.IMWRITE_JPEG_QUALITY), quality]
    cv.imwrite(save_file_name, img, encode_param)

    # Load resaved image
    img_low = cv.imread(save_file_name)
    img_low_rgb = img_low[:, :, ::-1]

    # Compute the square different between original image and the resaved image
    tmp = (img_rgb-img_low_rgb)**2

    # Take the average by kernel size b
    kernel = np.ones((smoothing_b, smoothing_b),
                np.float32)/(smoothing_b**2)
    tmp = cv.filter2D(tmp, -1, kernel)

    # Take the average of 3 channels
    tmp = np.average(tmp, axis=-1)

    # Shift the pixel from the center of the block to the left-top
    tmp = tmp[offset:(int(height-offset)), offset:(int(width-offset))]

    # Compute the nomalized component
    nomalized = tmp.min()/(tmp.max() - tmp.min())
```

```
        # Nomalization
        dst = tmp - nomalized

        # print(dst)
        # Plot the diffrent images
        plt.subplot(5, 4, pos_q+2), plt.imshow(dst,
                                    cmap='gray'), plt.title(quality)
        plt.xticks([]), plt.yticks([])
        quality = quality + 2
        bar.update(pos_q+2)

    bar.finish()
    print("Done")
    plt.suptitle('Exposing digital forgeries by JPEG Ghost')
    plt.show()
    os.remove(save_file_name)


def jpeg_ghost(file_path, quality):

    print("Analyzing...")
    bar = progressbar.ProgressBar(maxval=20,
                          widgets=[progressbar.Bar('=', '[', ']'), ' ', progressbar.Percentage()])
    bar.start()

    img = cv.imread(file_path)
    img_rgb = img[:, :, ::-1]

    # Quality of the reasaved images
    if quality == None:
        quality = 60

    # Size of the block
    smoothing_b = 17
    offset = (smoothing_b-1)/2

    # Size of the image
    height, width, channels = img.shape

    # Plot the original image
    plt.subplot(1, 2, 1), plt.imshow(img_rgb), plt.title('Image')
    plt.xticks([]), plt.yticks([])

    # Get the name of the image
    base = basename(file_path)
    file_name = os.path.splitext(base)[0]
    save_file_name = file_name+"_temp.jpg"
    bar.update(1)

    # Resaved the image with the new quality
    encode_param = [int(cv.IMWRITE_JPEG_QUALITY), quality]
    cv.imwrite(save_file_name, img, encode_param)
```

```
    # Load resaved image
    img_low = cv.imread(save_file_name)
    img_low_rgb = img_low[:, :, ::-1]
    bar.update(5)
    # Compute the square different between original image and the resaved image
    tmp = (img_rgb-img_low_rgb)**2

    # Take the average by kernel size b
    kernel = np.ones((smoothing_b, smoothing_b), np.float32)/(smoothing_b**2)
    tmp = cv.filter2D(tmp, -1, kernel)
    bar.update(10)
    # Take the average of 3 channels
    tmp = np.average(tmp, axis=-1)

    # Shift the pixel from the center of the block to the left-top
    tmp = tmp[int(offset):int(height-offset), int(offset):int(width-offset)]

    # Compute the nomalized component
    nomalized = tmp.min()/(tmp.max() - tmp.min())
    bar.update(15)
    # Nomalization
    dst = tmp - nomalized

    # print(dst)
    # Plot the diffrent images
    plt.subplot(1, 2, 2), plt.imshow(dst), plt.title(
        "Analysis. Quality = " + str(quality))
    plt.xticks([]), plt.yticks([])
    bar.update(20)

    bar.finish()
    print("Done")
    plt.suptitle('Exposing digital forgeries by JPEG Ghost')
    plt.show()
    os.remove(save_file_name)

def noise_inconsistencies(file_path, block_size):

    print("Analyzing...")
    bar = progressbar.ProgressBar(maxval=20,
                                  widgets=[progressbar.Bar('=', '[', ']'), ' ', progressbar.Percentage()])
    bar.start()
    if block_size == None:
        block_size = 8

    img = cv.imread(file_path)
    img_rgb = img[:, :, ::-1]

    imgYCC = cv.cvtColor(img, cv.COLOR_BGR2YCrCb)
    y, _, _ = cv.split(imgYCC)

    coeffs = pywt.dwt2(y, 'db8')
```

```
    bar.update(5)

    cA, (cH, cV, cD) = coeffs
    cD = cD[0:(len(cD)//block_size)*block_size,
           0:(len(cD[0])//block_size)*block_size]
    block = np.zeros(
        (len(cD)//block_size, len(cD[0])//block_size, block_size**2))
    bar.update(10)

    for i in range(0, len(cD), block_size):
        for j in range(0, len(cD[0]), block_size):
            blockElement = cD[i:i+block_size, j:j+block_size]
            temp = np.reshape(blockElement, (1, 1, block_size**2))
            block[int((i-1)/(block_size+1)),
                  int((j-1)/(block_size+1)), :] = temp

    bar.update(15)
    abs_map = np.absolute(block)
    med_map = np.median(abs_map, axis=2)
    noise_map = np.divide(med_map, 0.6745)
    bar.update(20)

    bar.finish()
    print("Done")

    plt.subplot(1, 2, 1), plt.imshow(img_rgb), plt.title('Image')
    plt.xticks([]), plt.yticks([])
    plt.subplot(1, 2, 2), plt.imshow(noise_map), plt.title('Analysis')
    plt.xticks([]), plt.yticks([])
    plt.suptitle('Exposing digital forgeries by using Noise Inconsistencies')
    plt.show()

def median_noise_inconsistencies(file_path, n_size):
    print("Analyzing...")
    bar = progressbar.ProgressBar(maxval=20,
                                  widgets=[progressbar.Bar('=', '[', ']'), ' ', progressbar.Percentage()])
    bar.start()

    img = cv.imread(file_path)
    img_rgb = img[:, :, ::-1]

    flatten = True
    multiplier = 10
    if n_size == None:
        n_size = 3
    bar.update(5)

    img_filtered = img

    img_filtered = cv.medianBlur(img, n_size)

    noise_map = np.multiply(np.absolute(img - img_filtered), multiplier)
    bar.update(15)
```

```
    if flatten == True:
        #noise_map = np.average(noise_map,axis=-1)
        noise_map = cv.cvtColor(noise_map, cv.COLOR_BGR2GRAY)
    bar.update(20)
    bar.finish()
    print("Done")

    plt.subplot(1, 2, 1), plt.imshow(img_rgb), plt.title('Image')
    plt.xticks([]), plt.yticks([])
    plt.subplot(1, 2, 2), plt.imshow(noise_map), plt.title('Analysis')
    plt.xticks([]), plt.yticks([])
    plt.suptitle(
        'Exposing digital forgeries by using Median-filter noise residue inconsistencies')
    plt.show(

def ela(file_path, quality, block_size):
    print("Analyzing...")
    bar = progressbar.ProgressBar(maxval=20,
                                  widgets=[progressbar.Bar('=', '[', ']'), ' ', progressbar.Percentage()])
    if block_size == None:
        block_size = 8
    img = cv.imread(file_path)
    img_rgb = img[:, :, ::-1]
    bar.update(5)

    # Get the name of the image
    base = basename(file_path)
    file_name = os.path.splitext(base)[0]
    save_file_name = file_name+"_temp.jpg"

    if quality == None:
        quality = 90
    multiplier = 15
    flatten = True

    # Resaved the image with the new quality
    encode_param = [int(cv.IMWRITE_JPEG_QUALITY), quality]
    cv.imwrite(save_file_name, img, encode_param)
    bar.update(10)

    # Load resaved image
    img_low = cv.imread(save_file_name)
    img_low = img_low[:, :, ::-1]

    ela_map = np.zeros((img_rgb.shape[0], img_rgb.shape[1], 3))

    ela_map = np.absolute(1.0*img_rgb - 1.0*img_low)*multiplier

    #ela_map = ela_map[:,:,::-1]
    bar.update(15)
    if flatten == True:
        ela_map = np.average(ela_map, axis=-1)
```

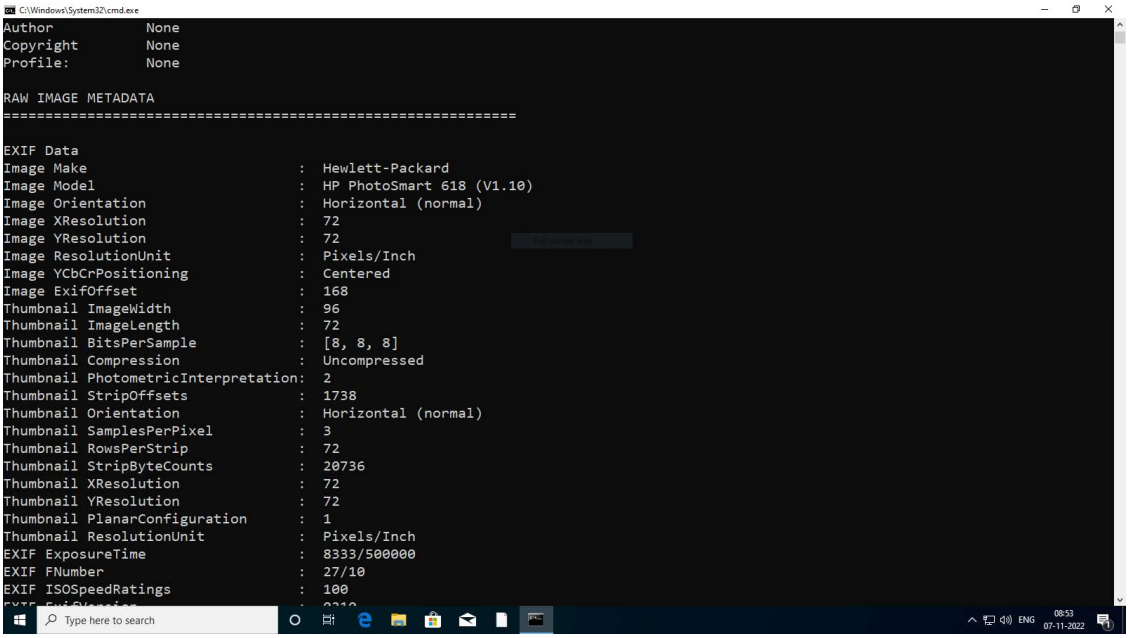
```
    bar.update(20)
    bar.finish()
    print("Done")

    plt.subplot(1, 2, 1), plt.imshow(img_rgb), plt.title('Image')
    plt.xticks([]), plt.yticks([])
    plt.subplot(1, 2, 2), plt.imshow(ela_map), plt.title('Analysis')
    plt.xticks([]), plt.yticks([])
    plt.suptitle('Exposing digital forgeries by using Error Level Analysis')
    plt.show()
    os.remove(save_file_name)
def cfa_tamper_detection(file_path):
    print("Analyzing...")
    bar = progressbar.ProgressBar(maxval=20,
                                  widgets=[progressbar.Bar('=', '[', ']'), ' ', progressbar.Percentage()])
    warnings.filterwarnings("ignore")

    img = cv.imread(file_path)
    img = img[:, :, ::-1]

    std_thresh = 5
    depth = 3

    img = img[0:int(round(math.floor(img.shape[0]/(2**depth))*(2**depth))),
            0:int(round(math.floor(img.shape[1]/(2**depth))*(2**depth))), :]
    bar.update(5)

    small_cfa_list = np.asarray([[[2, 1], [3, 2]], [[2, 3], [1, 2]], [
                                 [3, 2], [2, 1]], [[1, 2], [2, 3]]])

    # print(small_cfa_list)
    # print(small_cfa_list.shape)

    cfa_list = small_cfa_list

    # block size
    w1 = 16

    if img.shape[0] < w1 | img.shape[1] < w1:
        f1_map = np.zeros((img.shape))
        cfa_detected = [0, 0, 0, 0]
        return

    mean_error = np.ones((cfa_list.shape[0], 1))
    # print(mean_error.shape)
    bar.update(10)
    diffs = []
    f1_maps = []
    for i in range(cfa_list.shape[0]):

        bin_filter = np.zeros((img.shape[0], img.shape[1], 3))
        proc_im = np.zeros((img.shape[0], img.shape[1], 6))
        cfa = cfa_list[i]
```

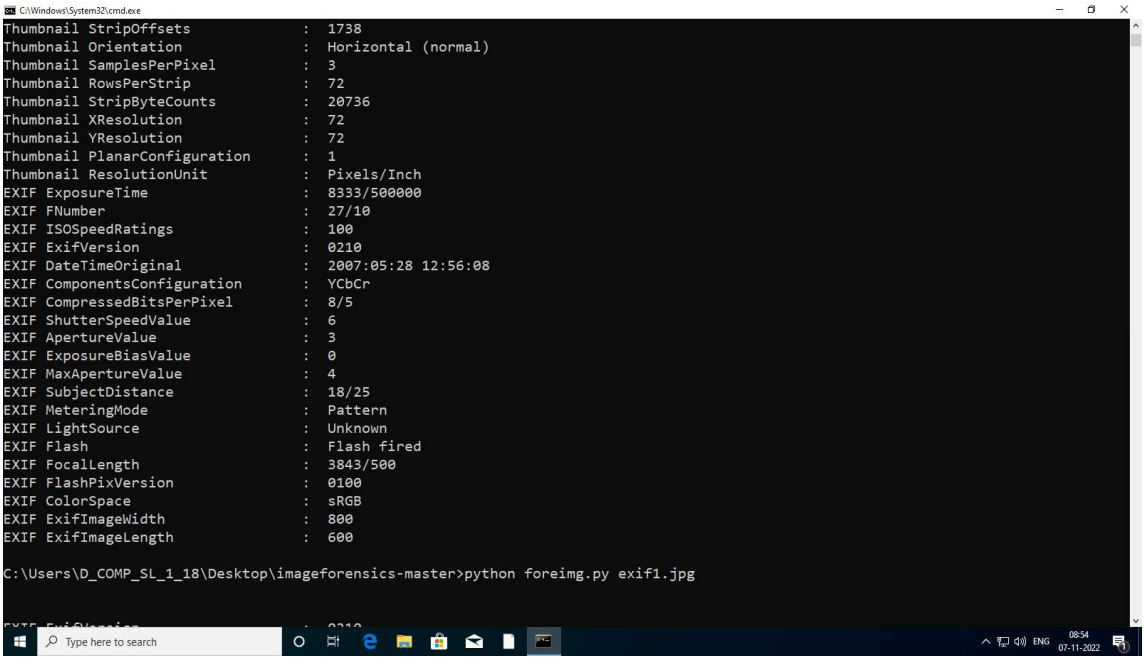
```
r = cfa == 1
g = cfa == 2
b = cfa == 3

bin_filter[:, :, 0] = npm.repmat(
   r, img.shape[0]//2, img.shape[1]//2)
bin_filter[:, :, 1] = npm.repmat(
   g, img.shape[0]//2, img.shape[1]//2)
bin_filter[:, :, 2] = npm.repmat(
   b, img.shape[0]//2, img.shape[1]//2)

cfa_im = np.multiply(1.0*img, bin_filter)

bilin_im = bilinInterolation(cfa_im, bin_filter, cfa)
# print(bilin_im[0:16,0:16,0])

proc_im[:, :, 0:3] = img
proc_im[:, :, 3:6] = 1.0*bilin_im
proc_im = 1.0*proc_im
# print(proc_im.shape)
block_result = np.zeros(
   (proc_im.shape[0]//w1, proc_im.shape[1]//w1, 6))

for h in range(0, proc_im.shape[0], w1):
   if h + w1 >= proc_im.shape[0]:
      break

   for k in range(0, proc_im.shape[1], w1):
      if k + w1 >= proc_im.shape[1]:
         break
      out = eval_block(proc_im[h:h+w1, k:k+w1, :])
      block_result[h//w1, k//w1, :] = out

stds = block_result[:, :, 3:6]
block_diffs = block_result[:, :, 0:3]
non_smooth = stds > std_thresh

bdnm = block_diffs[non_smooth]
mean_error[i] = np.average(np.reshape(bdnm, (1, bdnm.shape[0])))

temp = np.sum(block_diffs, axis=2)
rep_mat = np.zeros((temp.shape[0], temp.shape[1], 3))
rep_mat[:, :, 0] = temp
rep_mat[:, :, 1] = temp
rep_mat[:, :, 2] = temp

block_diffs = np.divide(block_diffs, rep_mat)

# print(block_diffs.shape)

diffs.append(np.reshape(
   block_diffs[:, :, 1], (1, block_diffs[:, :, 1].size)))
```

```
        f1_maps.append(block_diffs[:, :, 1])

    bar.update(15)

    diffs = np.asarray(diffs)
    diffs = np.reshape(diffs, (diffs.shape[0], diffs.shape[2]))

    for h in range(0, diffs.shape[0]):
        for k in range(0, diffs.shape[1]):
            if math.isnan(diffs[h, k]):
                diffs[h, k] = 0
    bar.update(18)
    f1_maps = np.asarray(f1_maps)
    val = np.argmin(mean_error)
    U = np.sum(np.absolute(diffs - 0.25), axis=0)
    U = np.reshape(U, (1, U.shape[0]))
    # print(U.shape)
    bar.update(19)
    F1 = np.median(U)

    CFADetected = cfa_list[val, :, :] == 2
 F1Map = f1_maps[val, :, :]
    bar.update(20)
    bar.finish()
    print("Done")

    plt.subplot(1, 2, 1), plt.imshow(img), plt.title('Image')
    plt.xticks([]), plt.yticks([])
    plt.subplot(1, 2, 2), plt.imshow(F1Map), plt.title('Analysis')
    plt.xticks([]), plt.yticks([])
    plt.suptitle('Image tamper detection based on demosaicing artifacts')
    plt.show()
def bilinInterolation(cfa_im, bin_filter, cfa):

    mask_min = np.divide(np.asarray([[1, 2, 1], [2, 4, 2], [1, 2, 1]]), 4.0)
    mask_maj = np.divide(np.asarray([[0, 1, 0], [1, 4, 1], [0, 1, 0]]), 4.0)

    if (np.argwhere(np.diff(cfa, axis=0) == 0).size != 0) | (np.argwhere(np.diff(cfa.T, axis=0) == 0).size != 0):
        mask_maj = np.multiply(mask_maj, 2.0)

    mask = np.ndarray(shape=(len(mask_min), len(mask_min[0]), 3))
    mask[:, :, 0] = mask_min[:, :]
    mask[:, :, 1] = mask_min[:, :]
    mask[:, :, 2] = mask_min[:, :]

    # print(mask)
    sum_bin_filter = np.reshape(
        np.sum(np.sum(bin_filter, axis=0), axis=0), (3))

    a = max(sum_bin_filter)
    # print(a)
    maj = np.argmax(sum_bin_filter)
```

```
    # print(maj)
    mask[:, :, maj] = mask_maj
    # print(mask)

    out_im = np.zeros((cfa_im.shape))

    for i in range(3):
        mixed_im = np.zeros((cfa_im.shape[0], cfa_im.shape[1]))
        orig_layer = cfa_im[:, :, i]
        #interp_layer = ndimage.convolve(orig_layer, mask[:,:,i])
        interp_layer = ndimage.correlate(
            orig_layer, mask[:, :, i], mode='constant')

        # print(interp_layer)

        for k in range(bin_filter.shape[0]):
            for h in range(bin_filter.shape[1]):
                if bin_filter[k, h, i] == 0:
                    mixed_im[k, h] = interp_layer[k, h]
                elif bin_filter[k, h, i] == 1:
                    mixed_im[k, h] = orig_layer[k, h]
        # print(mixed_im.shape)
        out_im[:, :, i] = mixed_im
        out_im = np.round(out_im)
        # print(out_im[:,:,0])
    return out_im

def eval_block(data):
    im = data
    Out = np.zeros((1, 1, 6))
    Out[:, :, 0] = np.mean(np.power(data[:, :, 0]-data[:, :, 3], 2.0))
    Out[:, :, 1] = np.mean(np.power(data[:, :, 1]-data[:, :, 4], 2.0))
    Out[:, :, 2] = np.mean(np.power(data[:, :, 2]-data[:, :, 5], 2.0))

    Out[:, :, 3] = np.std(np.reshape(im[:, :, 0], (1, im[:, :, 1].size)))
    Out[:, :, 4] = np.std(np.reshape(im[:, :, 1], (1, im[:, :, 2].size)))
    Out[:, :, 5] = np.std(np.reshape(im[:, :, 2], (1, im[:, :, 3].size)))

    # print(Out)
    return Out

if __name__ == "__main__":
    main()
```

Output:





Date:	
Marks obtained:	
Sign of course coordinator:	
Name of course Coordinator :	