```
import cv2 as cv
import matplotlib.pyplot as plt
import numpy as np
from skimage.metrics import structural_similarity as ssim

def rescaleFrame(frame,width,height):
    width=int(700)
    height=int(300)
    dimensions=(width,height)

    return cv.resize(frame, dimensions,interpolation=cv.INTER_AREA)

def grayScale(img):
    return cv.cvtColor(img,cv.COLOR_BGR2GRAY)

def extract_roi(image, coords):
    x1, y1 = coords[0]
    x2, y2 = coords[1]
    return image[y1:y2, x1:x2]

def compare_histograms(hist1, hist2, method=cv.HISTCMP_CORREL):
    return cv.compareHist(hist1, hist2, method)

# Compute histograms for each ROI
def compute_histograms(rois):
    histograms = []
    for roi in rois:
        hist = cv.calcHist([roi], [0], None, [256], [0, 256])
        hist = cv.normalize(hist, hist).flatten()
        histograms.append(hist)
    return histograms

# Compare intensities using SSIM (Structural Similarity Index)
def compare_ssim(img1, img2):
    return ssim(img1, img2)


def validate(string, denomination) -> str:
    # Determine the reference image based on the denomination
    if denomination == 500:
        reference_image_path = 'Deployment\Realfiveh.png'
    elif denomination == 200:
        reference_image_path = 'Deployment\Realtwoh.jpg'
    elif denomination == 100:
        reference_image_path = 'Deployment\Realh.jpg'
    else:
        return "Invalid denomination selected."


    # Read the reference image
    original = cv.imread(reference_image_path)
    if original is None:
```

```
    return f"Reference image for denomination {denomination} not found."

# Read the uploaded image
check = cv.imread(string)
if check is None:
    return "Uploaded image not found or could not be read."

#Resizing the Image
rescaled_image=rescaleFrame(original,width=700,height=300)
rescaled_image_check=rescaleFrame(check,width=700,height=300)
cv.imshow('Note',rescaled_image)
cv.imshow('Note2',rescaled_image_check)


#Converting into Gray Scale
grayImg=grayScale(rescaled_image)
grayImg_check=grayScale(rescaled_image_check)
cv.imshow('GraySclae',grayImg)
cv.imshow('GrayScale2',grayImg_check)
# cv.waitKey(0)

#We convert into Gray Scale because we can only see the intensity
#distribution of pixles rather than the colour itself

# Apply Gaussian Blurring
blurred_genuine = cv.GaussianBlur(grayImg, (5, 5), 0)
blurred_test = cv.GaussianBlur(grayImg_check, (5, 5), 0)

# Apply Histogram Equalization
equalized_genuine = cv.equalizeHist(blurred_genuine)
equalized_test = cv.equalizeHist(blurred_test)

# Edge Detection
genuine_img = cv.Canny(equalized_genuine, 180, 255)
test_img = cv.Canny(equalized_test, 180, 255)
cv.imshow('Edge detection',genuine_img)
cv.imshow('Edge detection2',test_img)

# ROI coordinates
security_mark_coords = [(0, 56), (28, 150)]
green_strip_coords = [(380, 0), (430, 300)]
serial_number_coords = [(445, 240), (630, 300)]
gandhiji_coords = [(152, 65), (385, 300)]

genuine_rois = [
    extract_roi(genuine_img, security_mark_coords),
    extract_roi(genuine_img, green_strip_coords),
    extract_roi(genuine_img, serial_number_coords),
    extract_roi(genuine_img, gandhiji_coords)
]

test_rois = [
```

```
        extract_roi(test_img, security_mark_coords),
        extract_roi(test_img, green_strip_coords),
        extract_roi(test_img, serial_number_coords),
        extract_roi(test_img, gandhiji_coords)
    ]

    # Ensure the ROIs are properly extracted and displayed
    for i, roi in enumerate(genuine_rois):
        cv.imshow(f'Genuine ROI {i+1}', roi)
    for i, roi in enumerate(test_rois):
        cv.imshow(f'Test ROI {i+1}', roi)



    genuine_histograms = compute_histograms(genuine_rois)
    test_histograms = compute_histograms(test_rois)

    # Compare each histogram from the test image with the corresponding histogram from the genuine image
    hist_comparison_results = []
    for genuine_hist, test_hist in zip(genuine_histograms, test_histograms):
        hist_comparison_result = compare_histograms(genuine_hist, test_hist)
        hist_comparison_results.append(hist_comparison_result)

    # Compare each ROI from the test image with the corresponding ROI from the genuine image
    ssim_comparison_results = []
    for genuine_roi, test_roi in zip(genuine_rois, test_rois):
        ssim_comparison_result = compare_ssim(genuine_roi, test_roi)
        ssim_comparison_results.append(ssim_comparison_result)

    # Combine histogram and SSIM results
    combined_results = []
    for hist_result, ssim_result in zip(hist_comparison_results, ssim_comparison_results):
        combined_results.append((hist_result + ssim_result) / 2)

    # Print comparison results
    print("Combined Comparison Results (Histogram + SSIM):")
    for i, result in enumerate(combined_results):
        print(f"ROI {i+1}: {result:.4f}")

    # Threshold for genuine vs. counterfeit decision (to be adjusted as needed)
    threshold = 0.54  # Adjusting threshold based on empirical testing and sensitivity needed
    is_genuine = all(result > threshold for result in combined_results)

    return (f" {'The note is genuine' if is_genuine else 'It is a fake note'}")




import pytesseract
import PIL.Image
```

```
import cv2 as cv
import numpy as np

def extract_roi(image):
    x1, y1 = 536, 195
    x2, y2 = 625, 255
    return image[y1:y2, x1:x2]

#This is a function defined to rescale the images

def rescaleFrame(frame):
    width = int(700)
    height = int(300)
    dimensions = (width, height)
    return cv.resize(frame, dimensions, interpolation=cv.INTER_AREA)

def preprocess_image(image):
    # Convert to grayscale
    gray = cv.cvtColor(image, cv.COLOR_BGR2GRAY)

    # Apply a less aggressive denoising algorithm (Bilateral Filtering)
    blurred = cv.bilateralFilter(gray, 5, 50, 50)

    # Apply thresholding to binarize the image
    _, binary = cv.threshold(blurred, 0, 255, cv.THRESH_BINARY + cv.THRESH_OTSU)

    return binary

def extract(path):
    # Load the image using OpenCV
    image = cv.imread(path)
    rescaled_image = rescaleFrame(image)
    roi = extract_roi(rescaled_image)

    # Preprocess the ROI
    preprocessed_roi = preprocess_image(roi)

    # cv.imshow('Processed ROI', preprocessed_roi)
    # cv.waitKey(0)

    # Convert the ROI to a PIL Image for pytesseract to process
    pil_image = PIL.Image.fromarray(preprocessed_roi)

    # Use pytesseract to extract text from the ROI
    myconfig = '--psm 13 --oem 3'  # Single line of text
    ans = pytesseract.image_to_string(pil_image, config=myconfig)

    print(f"Extracted Text: {ans}")
    print(type(ans))

    return ans
```

# import

ORIGINALITY REPORT

| 10% | 10% | 0% | % |
|---|---|---|---|
| SIMILARITY INDEX | INTERNET SOURCES | PUBLICATIONS | STUDENT PAPERS |

PRIMARY SOURCES

| | | |
|---|---|---|
| 1 | pythonmana.com<br>Internet Source | 2% |
| 2 | repozitorij.fsb.unizg.hr<br>Internet Source | 2% |
| 3 | medium.com<br>Internet Source | 2% |
| 4 | 4f84b2.xyz<br>Internet Source | 2% |
| 5 | docs.opencv.org<br>Internet Source | 1% |
| 6 | openi.pcl.ac.cn<br>Internet Source | 1% |

| | | | |
|---|---|---|---|
| Exclude quotes | On | Exclude matches | < 10 words |
| Exclude bibliography | On | | |