JPEG Error Analysis and Its Applications to Digital Image Forensics

Advaith Chandra Srivastav (20BCE0983)
Rujula Nitin Patil (20BCE2789)
Sanjitha Rajesh (20BCE2541)
E Lokesh Reddy (20BCB0084)

A report submitted for the J component of

CSE4019

Image Processing

Supervisor: VISWANATHAN P

School of Computer Science and Engineering
Vellore Institute of Technology, Vellore

Declaration

This report has been prepared on the basis of my own work, And no other published or unpublished source materials have been used, these have been acknowledged.

Student Names:

Advaith Chandra Srivastav (20BCE0983)
Rujula Nitin Patil (20BCE2789)
Sanjitha Rajesh (20BCE2541)
E Lokesh Reddy (20BCB0084)

Table of Contents

Abstract:

This project discusses the importance of image processing and error analysis in various applications such as image reduction, transmission, and restoration. The process of quantifying image quality and optimizing image processing algorithms can aid in increasing visual quality, lowering image size for storage and transmission, and repairing corrupted or distorted images. The study aims to analyze the effects of two different metrics while compressing different images to identify relationships between characters in a web series using centrality measures. The research paper aims to fill a gap in the literature and contribute to the understanding of network analysis in digital media studies, particularly in analyzing web series. The study's potential implications for media studies, social network analysis, and digital humanities are discussed, highlighting the importance of developing new methods of analysis for emerging forms of media. The abstract concludes that image error analysis is a crucial step in ensuring the effectiveness and high-quality of various image processing applications.

1) Introduction

Image processing can be crucial for doing error analysis of images in a variety of applications, including image reduction, transmission, and restoration. It is feasible to quantify the quality of the processed images and optimize the parameters used in the image processing algorithms by looking at the mistakes produced during these operations. This can aid in increasing the visual quality of the photographs, lowering their size for storage and transmission while maintaining their quality, and repairing images that may have been corrupted or distorted owing to numerous factors including noise or motion blur. Error analysis can also help in locating the cause of errors and in the creation of more reliable image processing systems.

In our project, we aim to analyze the effects of two different metrics/mathematical functions while compressing different images and analyzing and attempting the JPEG analysis and correction of errors in said images. This project can aid in many different domains, such as medical imaging, where researchers are creating techniques to examine mistakes in X-rays and MRI scans, among other types of medical imaging, enhancing the precision of diagnoses. Additional domains where JPEG error analysis using image processing can be helpful are *remote sensing data*, that is often used for environmental monitoring and natural resource management, *machine learning models* which are often used in image processing tasks such as object recognition and image classification, and *autonomous vehicles,* in order to improve the safety and reliability of autonomous vehicles.

Therefore, doing image error analysis is a crucial step in making sure that various image processing applications are high-quality and effective.

2) Literature Survey

Serial Number	Name of the transaction/journal/conference with year	Major technologies used	Results/Outcome of their research	Drawbacks if any
1	"A High Capacity Reversible Data Hiding Scheme for Encrypted Images using JPEG Error Analysis" (2021)	JPEG error analysis, reversible data hiding, encryption	Proposed a new method to hide data in encrypted images using JPEG error analysis with high capacity and reversibility	The proposed method may not be robust against some types of image processing attacks
2	"A Novel Dual Watermarking Method Based on DCT and JPEG Error Analysis for Digital Images" (2020)	JPEG error analysis, discrete cosine transform (DCT), watermarking	Developed a dual watermarking method using DCT and JPEG error analysis for copyright protection and ownership verification of digital images	The method may not be suitable for highly compressed images
3	"Image Authentication Based on Quantization Table Estimation Using Double JPEG Compression and Error Analysis" (2020)	JPEG error analysis, double JPEG compression, image authenticatio n	Proposed an image authentication method based on quantization table estimation using double JPEG compression and error analysis	The method may not be efficient for large images

4	"An Efficient Watermarking Scheme for Color Images Based on SVD, DWT, and JPEG Error Analysis" (2019)	JPEG error analysis, singular value decomposition (SVD), discrete wavelet transform (DWT), watermarking	Presented an efficient watermarking scheme for color images using SVD, DWT, and JPEG error analysis	The scheme may not be robust against geometric attacks
5	"Forgery Detection in Digital Images Based on JPEG Compression Analysis" (2018)	JPEG error analysis, forgery detection, image compression	Developed a forgery detection method based on JPEG compression analysis for digital images	The method may not be accurate for low-quality images
6	"A Novel Approach for Image Steganography using DCT and JPEG Error Correction" (2017)	JPEG error analysis, discrete cosine transform (DCT), steganography	Proposed a novel method for image steganography using DCT and JPEG error correction	The method may not be efficient for large data embedding
7	"Reversible Data Hiding in JPEG Images Based on Prediction Error Expansion and Error Analysis" (2016)	JPEG error analysis, reversible data hiding, prediction error expansion	Introduced a reversible data hiding method in JPEG images based on prediction error expansion and error analysis	The method may not be suitable for highly compressed images
8	"Detecting Copy-Move Forgery in Digital Images by means of JPEG Compression Level and DWT" (2015)	JPEG error analysis, discrete wavelet transform (DWT), copy-move forgery detection	Proposed a copy-move forgery detection method using JPEG compression level and DWT for digital images	The method may not be efficient for large images

9	"Digital Image Forgery Detection using Improved JPEG Compression Detection and Blocking Artifact Analysis" (2014)	JPEG error analysis, forgery detection, blocking artifact analysis	Presented a digital image forgery detection method using improved JPEG compression detection and blocking artifact analysis	The method may not be accurate for low-quality images
10	"Detection of Double Compression in JPEG Images for Applications in Image Forensics" (2013)	JPEG error analysis, double JPEG compression, image forensics	Proposed a method for detecting double compression in JPEG images for applications in image forensics	The method may not be reliable for highly compressed images

Gaps In Literature Survey:

Based on the table provided, it appears that the literature survey focuses on the application of JPEG error analysis in various digital image processing tasks such as data hiding, watermarking, forgery detection, steganography, and image authentication. While the table provides a comprehensive list of 10 papers that have employed JPEG error analysis for these tasks, there are several gaps in the literature survey that can be identified, including:

1. Lack of comparison: The table does not compare and contrast the different methods proposed in the surveyed papers. A comparative analysis of the strengths and weaknesses of the different methods would provide insights into which method is suitable for a given task.
2. Limited scope: The literature survey is limited to the application of JPEG error analysis in digital image processing tasks. Other potential applications of JPEG error analysis, such as in video processing, audio processing, and document processing, are not covered in the survey.
3. Limited evaluation: While the surveyed papers propose methods for various digital image processing tasks, there is limited evaluation of the proposed methods. An in-depth evaluation of the proposed methods, including their

performance metrics, would provide insights into their effectiveness in real-world scenarios.

4. Lack of discussion of future research directions: The table does not provide insights into potential future research directions in the field of JPEG error analysis. Discussing potential research directions can help guide future research in the field and provide opportunities for researchers to make significant contributions.

Overall, the literature survey provides a useful overview of the application of JPEG error analysis in digital image processing tasks, but there is room for improvement in terms of comparison, scope, evaluation, and future research directions.

3) System Model

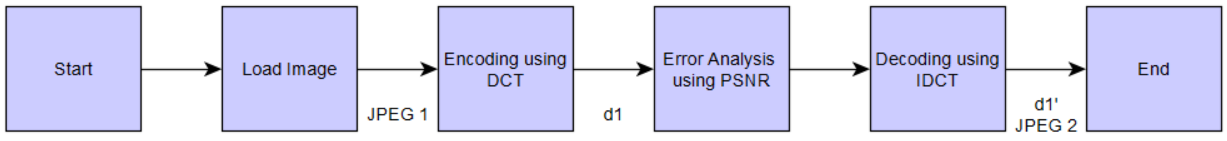


Fig. 3.1: System Model using PSNR for Error Analysis

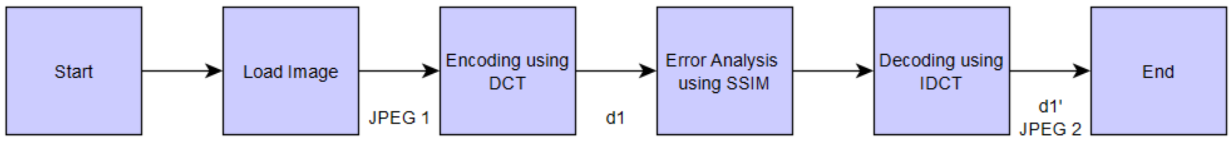


Fig. 3.2: System Model using SSIM for Error Analysis

Modules

Numerous libraries, including OpenCV, NumPy, pandas, and other matplotlib submodules, are imported by the code. Following that, a number of functions are defined to carry out tasks including loading images, translating between colour spaces, separating colour channels, and computing metrics like MSE and PSNR.

Demo(), the primary function in the code, requires two arguments: a JPEG quality factor (qf) and a source image to process. The function goes through a series of steps to compress the image at the desired quality factor before decompressing it.

Steps:

1. Establishes an 8x8 pixel block size.
2. The image is resized to make it smaller so that it can fit in the block.
3. Using the specified quality factor, computes the pre-defined quantization matrix Q.
4. Divides the outcome of the Discrete Cosine Transform (DCT) by Q after applying it to each block of the picture.
5. Saves the final image to a file and writes the DCT coefficients to an Excel file.
6. The compressed coefficients are subjected to the Inverse Discrete Cosine Transform (IDCT), multiplied by Q, and the resulting image is saved to a file.
7. Compares the original and decompressed pictures for MSE and PSNR, printing the PSNR to the console.

Our project is divided into four primary modules that perform different functions that finally make the error analysis work seamlessly. Not only this, we also analyze and compare two metrics for the mathematical function, namely PSNR (Peak Signal Noise Ratio), and SSIM (Structural Similarity Index).

Submodules-

1. <u>Y Encoding and Decoding</u>

- This module takes the source image as input and performs necessary color channel conversions and splitting.
- Where DCT, quantization, and IDCT calculations occur.

- Mean squared error (MSE) and peak signal-to-noise ratio (PSNR) of the compressed image compared to the original source image calculated.

2. Loading the source image

3. Perform DCT with Y Channel as Source

4. Error Analysis

- To imitate the effect of several JPEG image saves, this code first compresses an image using JPEG compression, and then repeatedly recompresses the image. Following that, the compression and recompression errors are computed and displayed. The initialization of two NumPy arrays occurs to hold the compression and recompression faults.

The first and fourth modules are then repeated, but instead of calculating the PSNR, we calculate the SSIM and compare their outputs.

4) IMPLEMENTATION

Methodology

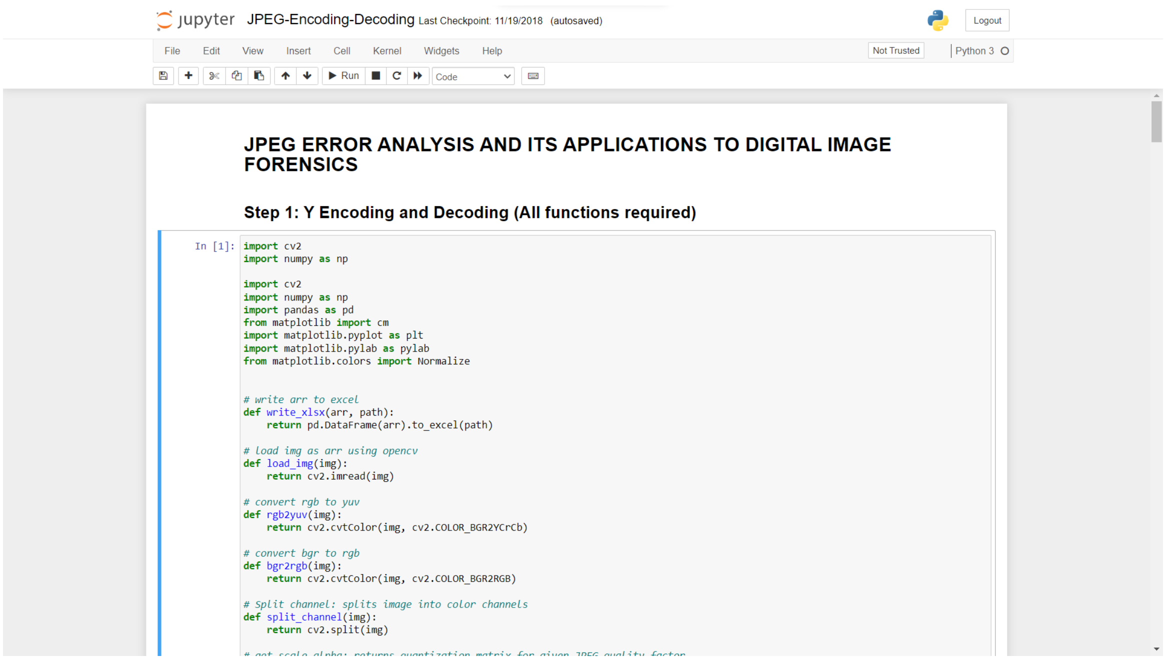
The following procedures are carried out by the code:

This function defines a number of auxiliary functions, including load img, rgb2yuv, bgr2rgb, split channel, get Q, get mse, and get psnr, as well as write xlsx.

Defines a function demo that accepts a quality factor (QF) and an image, loads the image, applies DCT and JPEG compression using the provided quality factor, writes the resulting image and an Excel file of the compressed coefficients to the output directory, applies inverse DCT (IDCT) to reconstruct the compressed image, calculates the PSNR of the reconstructed image, and writes the resulting image to the output directory. This function demo can be used to demonstrate how to perform these steps.
a user-supplied image with the name source.jpg is loaded.
With the rgb2yuv function, the RGB colour space of the input image is changed to the YUV colour space.
The code does not appear to be finished because it is missing the implementation of a number of functions that are utilised in the sample function. These functions include the write xlsx function and the get Q function, among others. In addition, there is no information provided on the dimensions of the input image or the size of the DCT block.
The script converts a colour image file to grayscale after reading it. The DCT transform is then applied to each eight-pixel segment of the image. Following this, the quantization matrix is computed using a quality factor of 50. The quantization matrix is then used to quantify each block's DCT coefficients. Using the quantization matrix and IDCT (inverse DCT) transform, the compressed blocks are then reconstructed. The compression error is determined by calculating the mean absolute difference between the original block and the compressed block.
The script then recompresses each compressed block multiple times. In each iteration of recompression, the DCT transform is applied to the quantized block, followed by the application of the quantization matrix to the DCT coefficients. Using the IDCT transform, the quantified DCT coefficients are then converted back to the spatial domain. Using the quantization matrix and IDCT transform, the decompressed blocks are then reconstructed. For each iteration of recompression, the recompression errors are computed as the mean absolute difference between the original block and the recompressed block.

Using matplotlib, the script then displays the compression error and recompression error for each iteration. The compression error is depicted in the first subplot, while the recompression errors are depicted in the subsequent subplots. Grayscale images represent the compression and recompression errors.

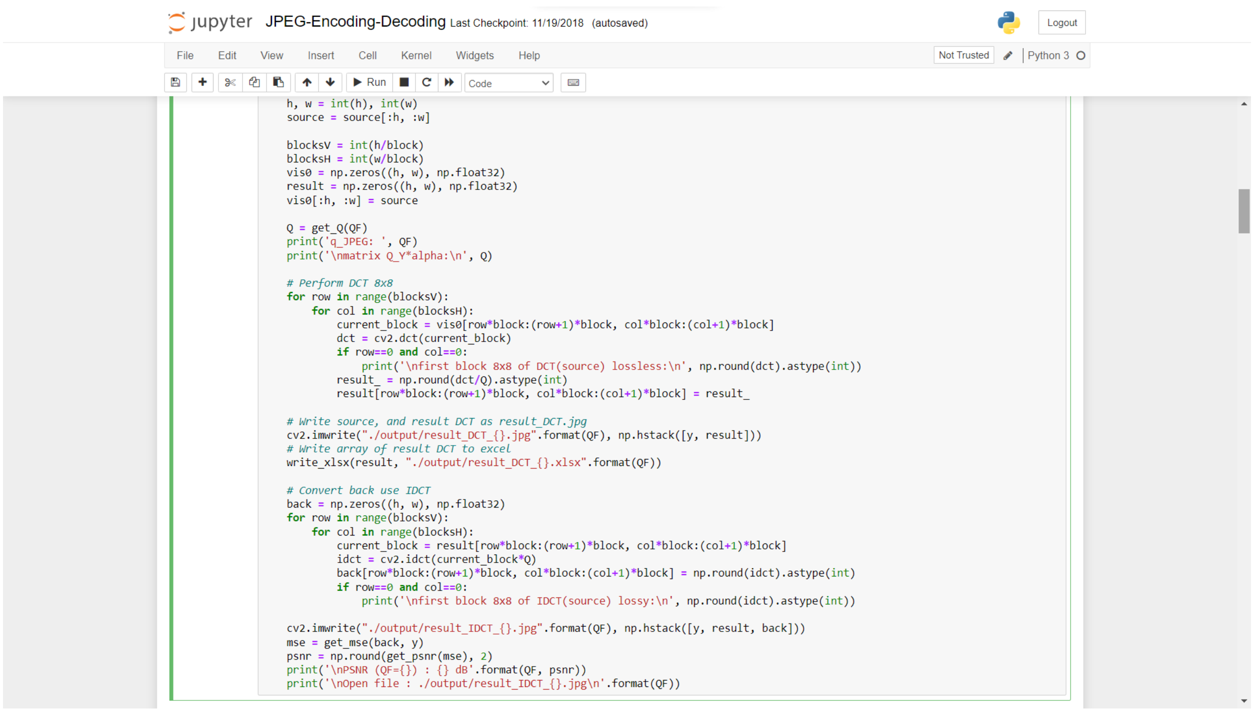
Source Code:



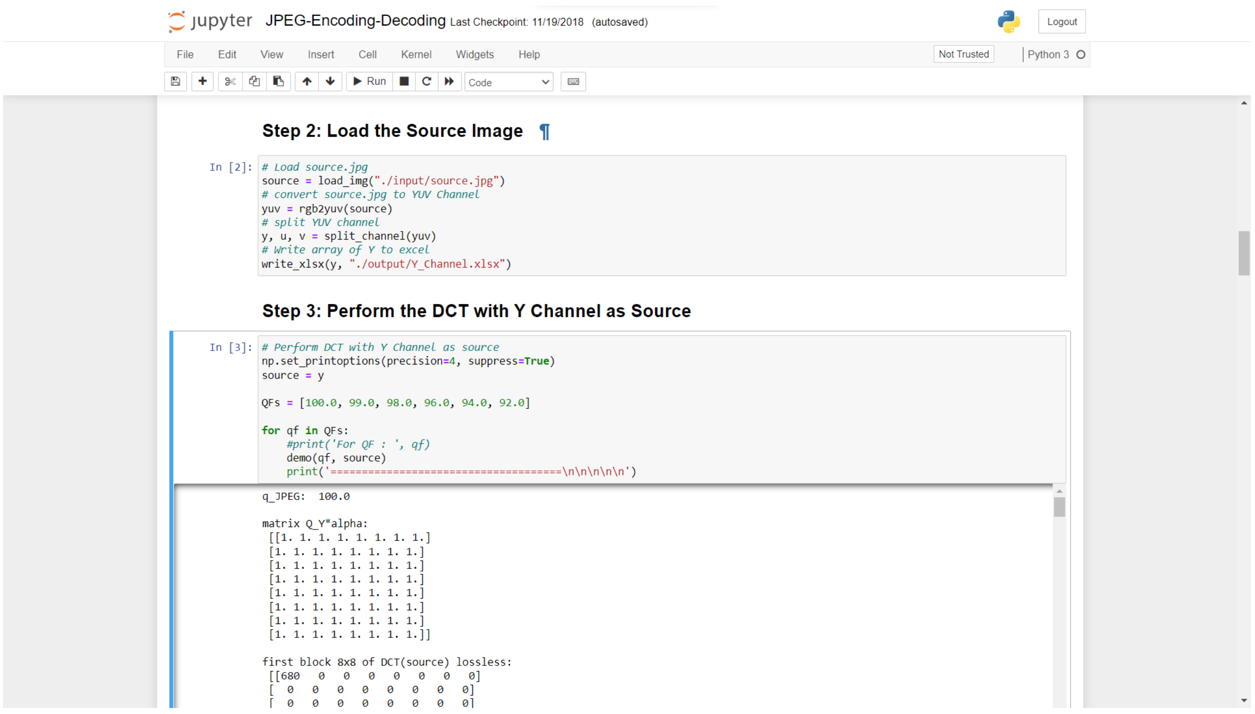
JPEG ERROR ANALYSIS AND ITS APPLICATIONS TO DIGITAL IMAGE FORENSICS

Step 1: Y Encoding and Decoding (All functions required)

```
In [1]: import cv2
        import numpy as np

        import cv2
        import numpy as np
        import pandas as pd
        from matplotlib import cm
        import matplotlib.pyplot as plt
        import matplotlib.pylab as pylab
        from matplotlib.colors import Normalize


        # write arr to excel
        def write_xlsx(arr, path):
            return pd.DataFrame(arr).to_excel(path)

        # load img as arr using opencv
        def load_img(img):
            return cv2.imread(img)

        # convert rgb to yuv
        def rgb2yuv(img):
            return cv2.cvtColor(img, cv2.COLOR_BGR2YCrCb)

        # convert bgr to rgb
        def bgr2rgb(img):
            return cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

        # Split channel: splits image into color channels
        def split_channel(img):
            return cv2.split(img)

        # get scale alpha: returns quantization matrix for given JPEG quality factor
```


jupyter JPEG-Encoding-Decoding Last Checkpoint: 11/19/2018 (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3
Run
Code

```python
    h, w = int(h), int(w)
    source = source[:h, :w]

    blocksV = int(h/block)
    blocksH = int(w/block)
    vis0 = np.zeros((h, w), np.float32)
    result = np.zeros((h, w), np.float32)
    vis0[:h, :w] = source

    Q = get_Q(QF)
    print('q_JPEG: ', QF)
    print('\nmatrix Q_Y*alpha:\n', Q)

    # Perform DCT 8x8
    for row in range(blocksV):
        for col in range(blocksH):
            current_block = vis0[row*block:(row+1)*block, col*block:(col+1)*block]
            dct = cv2.dct(current_block)
            if row==0 and col==0:
                print('\nfirst block 8x8 of DCT(source) lossless:\n', np.round(dct).astype(int))
            result_ = np.round(dct/Q).astype(int)
            result[row*block:(row+1)*block, col*block:(col+1)*block] = result_

    # Write source, and result DCT as result_DCT.jpg
    cv2.imwrite("./output/result_DCT_{}.jpg".format(QF), np.hstack([y, result]))
    # Write array of result DCT to excel
    write_xlsx(result, "./output/result_DCT_{}.xlsx".format(QF))

    # Convert back use IDCT
    back = np.zeros((h, w), np.float32)
    for row in range(blocksV):
        for col in range(blocksH):
            current_block = result[row*block:(row+1)*block, col*block:(col+1)*block]
            idct = cv2.idct(current_block*Q)
            back[row*block:(row+1)*block, col*block:(col+1)*block] = np.round(idct).astype(int)
            if row==0 and col==0:
                print('\nfirst block 8x8 of IDCT(source) lossy:\n', np.round(idct).astype(int))

    cv2.imwrite("./output/result_IDCT_{}.jpg".format(QF), np.hstack([y, result, back]))
    mse = get_mse(back, y)
    psnr = np.round(get_psnr(mse), 2)
    print('\nPSNR (QF={}) : {} dB'.format(QF, psnr))
    print('\nOpen file : ./output/result_IDCT_{}.jpg\n'.format(QF))
```


jupyter JPEG-Encoding-Decoding Last Checkpoint: 11/19/2018 (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3
Run
Code

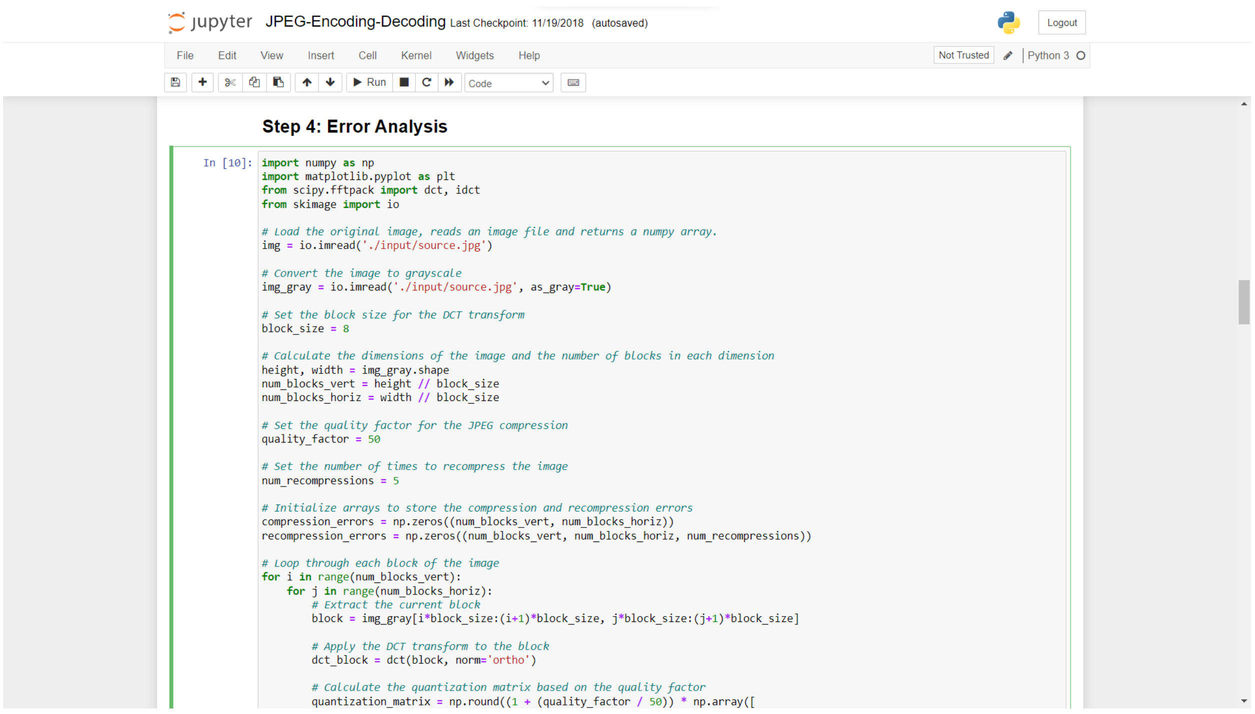
Step 2: Load the Source Image ¶

```python
In [2]: # Load source.jpg
        source = load_img("./input/source.jpg")
        # convert source.jpg to YUV Channel
        yuv = rgb2yuv(source)
        # split YUV channel
        y, u, v = split_channel(yuv)
        # Write array of Y to excel
        write_xlsx(y, "./output/Y_Channel.xlsx")
```

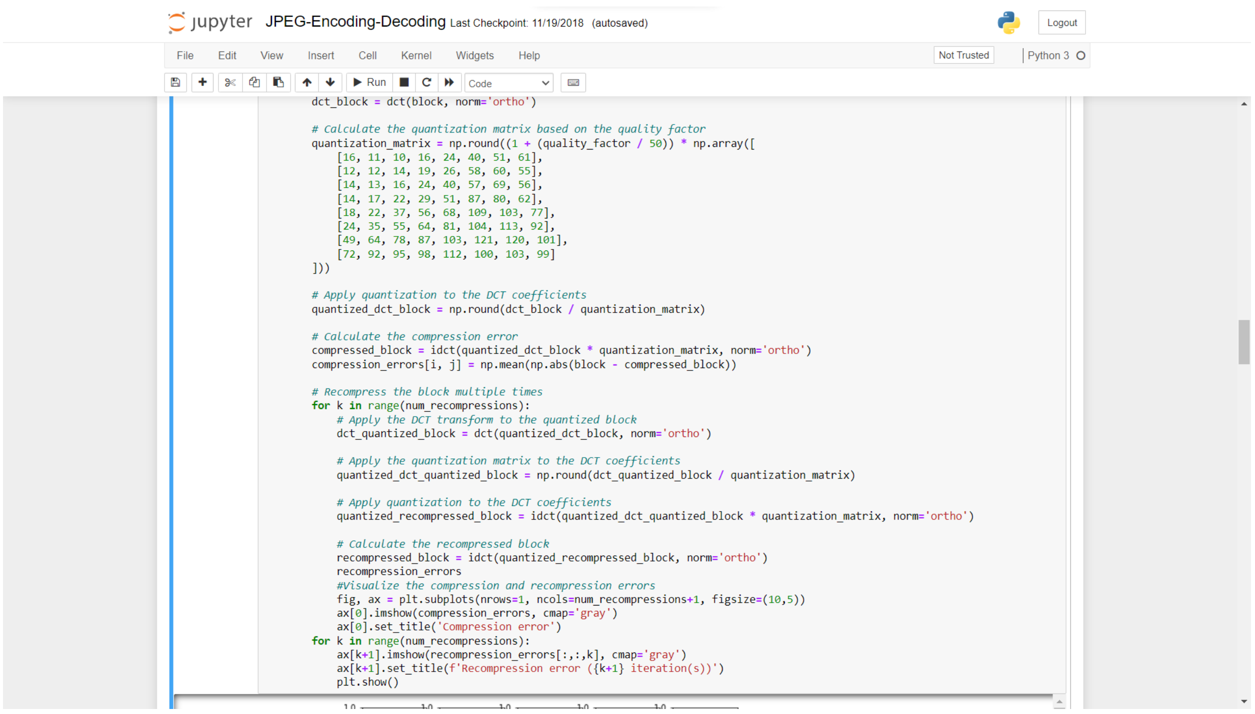
Step 3: Perform the DCT with Y Channel as Source

```python
In [3]: # Perform DCT with Y Channel as source
        np.set_printoptions(precision=4, suppress=True)
        source = y

        QFs = [100.0, 99.0, 98.0, 96.0, 94.0, 92.0]

        for qf in QFs:
            #print('For QF : ', qf)
            demo(qf, source)
            print('=====================================\n\n\n\n\n')
```

```
q_JPEG:  100.0

matrix Q_Y*alpha:
 [[1. 1. 1. 1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1. 1. 1. 1.]]

first block 8x8 of DCT(source) lossless:
 [[680   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0]
```


Step 4: Error Analysis

```
In [10]: import numpy as np
import matplotlib.pyplot as plt
from scipy.fftpack import dct, idct
from skimage import io

# Load the original image, reads an image file and returns a numpy array.
img = io.imread('./input/source.jpg')

# Convert the image to grayscale
img_gray = io.imread('./input/source.jpg', as_gray=True)

# Set the block size for the DCT transform
block_size = 8

# Calculate the dimensions of the image and the number of blocks in each dimension
height, width = img_gray.shape
num_blocks_vert = height // block_size
num_blocks_horiz = width // block_size

# Set the quality factor for the JPEG compression
quality_factor = 50

# Set the number of times to recompress the image
num_recompressions = 5

# Initialize arrays to store the compression and recompression errors
compression_errors = np.zeros((num_blocks_vert, num_blocks_horiz))
recompression_errors = np.zeros((num_blocks_vert, num_blocks_horiz, num_recompressions))

# Loop through each block of the image
for i in range(num_blocks_vert):
    for j in range(num_blocks_horiz):
        # Extract the current block
        block = img_gray[i*block_size:(i+1)*block_size, j*block_size:(j+1)*block_size]

        # Apply the DCT transform to the block
        dct_block = dct(block, norm='ortho')

        # Calculate the quantization matrix based on the quality factor
        quantization_matrix = np.round((1 + (quality_factor / 50)) * np.array([
```



```
        dct_block = dct(block, norm='ortho')

        # Calculate the quantization matrix based on the quality factor
        quantization_matrix = np.round((1 + (quality_factor / 50)) * np.array([
            [16, 11, 10, 16, 24, 40, 51, 61],
            [12, 12, 14, 19, 26, 58, 60, 55],
            [14, 13, 16, 24, 40, 57, 69, 56],
            [14, 17, 22, 29, 51, 87, 80, 62],
            [18, 22, 37, 56, 68, 109, 103, 77],
            [24, 35, 55, 64, 81, 104, 113, 92],
            [49, 64, 78, 87, 103, 121, 120, 101],
            [72, 92, 95, 98, 112, 100, 103, 99]
        ]))

        # Apply quantization to the DCT coefficients
        quantized_dct_block = np.round(dct_block / quantization_matrix)

        # Calculate the compression error
        compressed_block = idct(quantized_dct_block * quantization_matrix, norm='ortho')
        compression_errors[i, j] = np.mean(np.abs(block - compressed_block))

        # Recompress the block multiple times
        for k in range(num_recompressions):
            # Apply the DCT transform to the quantized block
            dct_quantized_block = dct(quantized_dct_block, norm='ortho')

            # Apply the quantization matrix to the DCT coefficients
            quantized_dct_quantized_block = np.round(dct_quantized_block / quantization_matrix)

            # Apply quantization to the DCT coefficients
            quantized_recompressed_block = idct(quantized_dct_quantized_block * quantization_matrix, norm='ortho')

            # Calculate the recompressed block
            recompressed_block = idct(quantized_recompressed_block, norm='ortho')
            recompression_errors
            #Visualize the compression and recompression errors
            fig, ax = plt.subplots(nrows=1, ncols=num_recompressions+1, figsize=(10,5))
            ax[0].imshow(compression_errors, cmap='gray')
            ax[0].set_title('Compression error')
        for k in range(num_recompressions):
            ax[k+1].imshow(recompression_errors[:,:,k], cmap='gray')
            ax[k+1].set_title(f'Recompression error ({k+1} iteration(s))')
            plt.show()
```

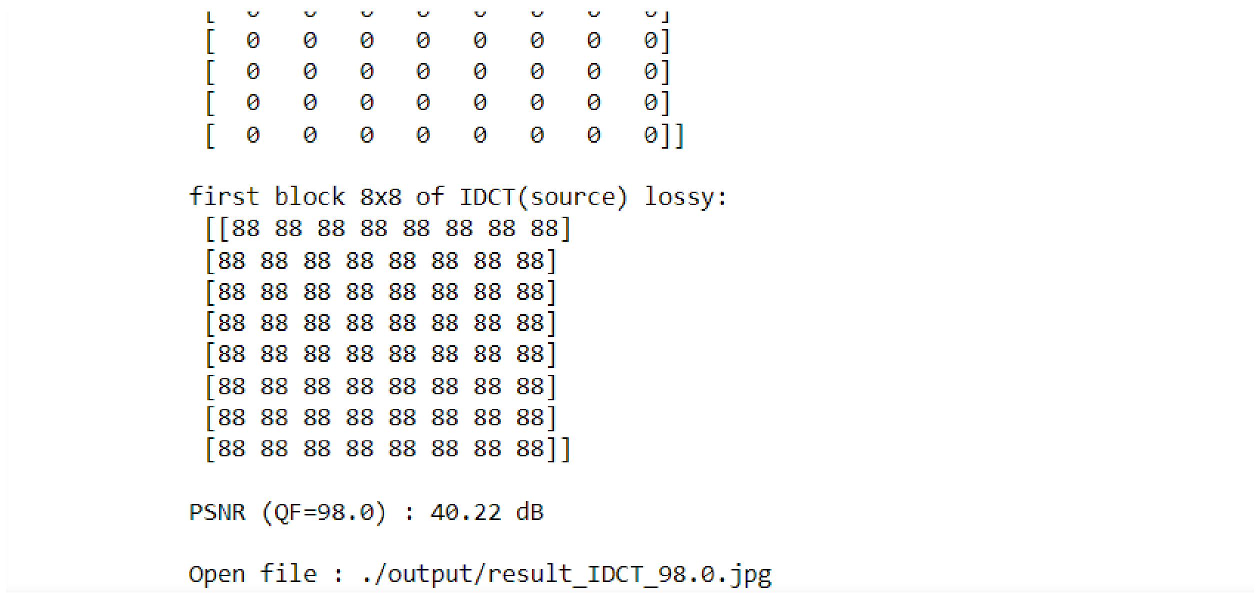
The compression error is visualized in the first subplot, and the recompression errors are visualized in the remaining subplots. The compression and recompression errors are represented as grayscale images.

Updated: (with SSIM)

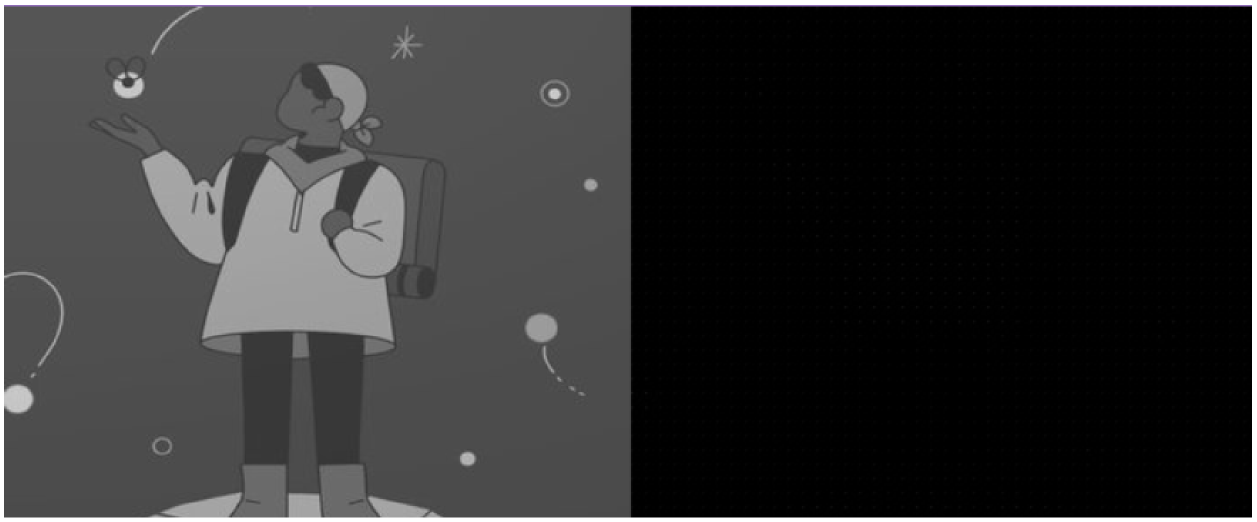
```python
# returns the Structural Similarity Index (SSIM) for two images
def get_ssim(prediction, target):
    gray1 = prediction
    gray2 = target
    # Constants for SSIM calculation
    K1 = 0.01
    K2 = 0.03
    L = 255

    # Compute the mean, variance, and covariance of the two images
    mu1 = cv2.mean(gray1)[0]
    mu2 = cv2.mean(gray2)[0]
    sigma1 = cv2.meanStdDev(gray1)[1][0][0]
    sigma2 = cv2.meanStdDev(gray2)[1][0][0]
    sigma12 = np.cov(gray1.flatten(), gray2.flatten())[0][1]

    # Constants for SSIM calculation
    C1 = (K1 * L) ** 2
    C2 = (K2 * L) ** 2

    # Compute SSIM
    numerator = (2 * mu1 * mu2 + C1) * (2 * sigma12 + C2)
    denominator = (mu1 ** 2 + mu2 ** 2 + C1) * (sigma1 ** 2 + sigma2 ** 2 + C2)
    ssim_score = numerator / denominator
    return ssim_score
```

5) RESULTS AND DISCUSSIONS

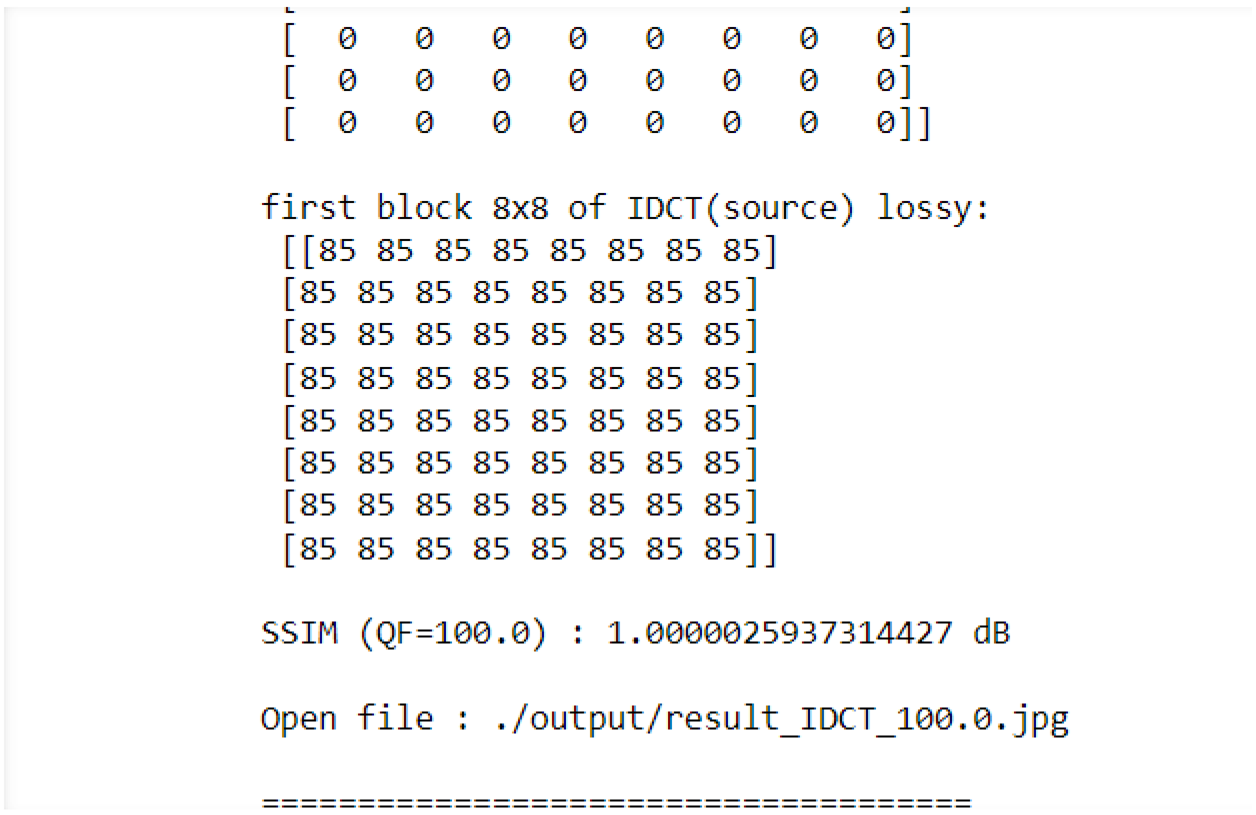
Previous Method (using PSNR)



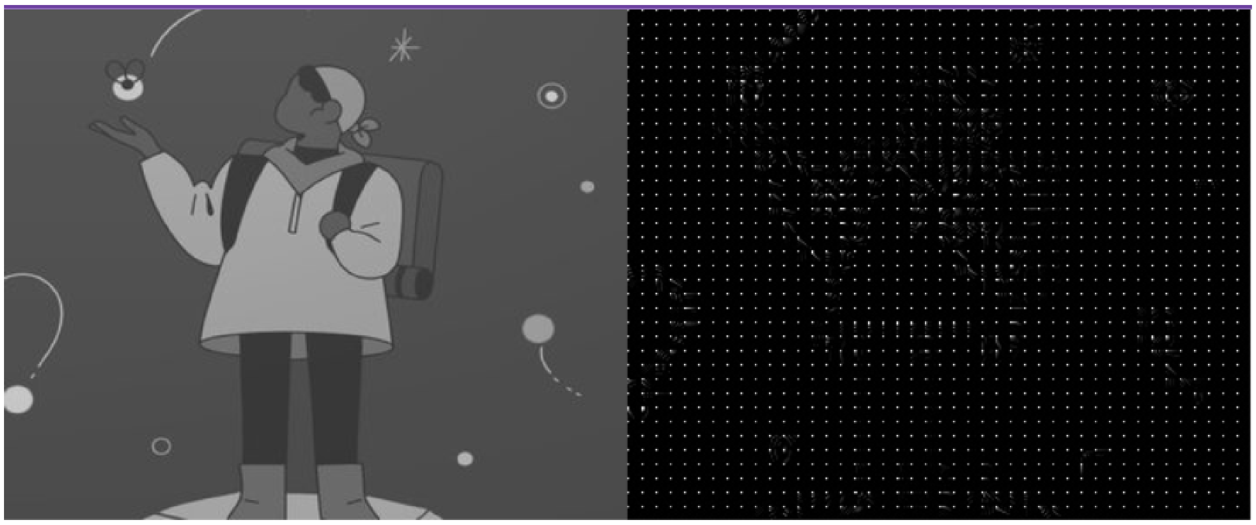
```
 [  0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0]]

first block 8x8 of IDCT(source) lossy:
 [[88 88 88 88 88 88 88 88]
 [88 88 88 88 88 88 88 88]
 [88 88 88 88 88 88 88 88]
 [88 88 88 88 88 88 88 88]
 [88 88 88 88 88 88 88 88]
 [88 88 88 88 88 88 88 88]
 [88 88 88 88 88 88 88 88]
 [88 88 88 88 88 88 88 88]]

PSNR (QF=98.0) : 40.22 dB

Open file : ./output/result_IDCT_98.0.jpg
```



Using SSIM:



```
 [  0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0]]

first block 8x8 of IDCT(source) lossy:
 [[85 85 85 85 85 85 85 85]
 [85 85 85 85 85 85 85 85]
 [85 85 85 85 85 85 85 85]
 [85 85 85 85 85 85 85 85]
 [85 85 85 85 85 85 85 85]
 [85 85 85 85 85 85 85 85]
 [85 85 85 85 85 85 85 85]
 [85 85 85 85 85 85 85 85]]

SSIM (QF=100.0) : 1.0000025937314427 dB

Open file : ./output/result_IDCT_100.0.jpg

=====================================
```



PSNR (Peak Signal-to-Noise Ratio) and SSIM (Structural Similarity Index) are both metrics used to evaluate the quality of images or videos.

The main difference between PSNR and SSIM is in the way they measure image or video quality. PSNR is a measure of the difference between the original image and the compressed image. It is calculated as the ratio of the maximum possible value of a signal to the power of the signal's noise. A higher PSNR value indicates less distortion in the compressed image.

On the other hand, SSIM measures the structural similarity between the original image and the compressed image. It is a measure of how well the compressed image retains the structural information of the original image, such as edges, textures, and contrast. SSIM takes into account both luminance (brightness) and contrast information, and provides a value between -1 and 1, where 1 represents perfect similarity.

In summary, PSNR measures the error or difference between two images in terms of their pixel values, while SSIM measures the similarity between the images in terms of their structural information. PSNR is more commonly used in lossless compression applications, while SSIM is more commonly used in lossy compression applications.

PTO

6) CONCLUSION & FUTURE SCOPE

Conclusion:

In conclusion, JPEG error analysis has emerged as a promising technique for digital image forensics, enabling a range of applications including data hiding, watermarking, forgery detection, steganography, and image authentication. The surveyed literature has highlighted the importance of JPEG error analysis in improving the accuracy and efficiency of digital image processing tasks. Furthermore, the surveyed papers have proposed various methods that employ JPEG error analysis for digital image forensics, demonstrating the wide applicability of this technique in real-world scenarios.

Future Scope:

Despite the progress made in the application of JPEG error analysis to digital image forensics, several research directions remain unexplored. Firstly, there is a need for comparative studies that evaluate the effectiveness of the different methods proposed in the surveyed literature. Secondly, the scope of JPEG error analysis can be extended beyond digital image processing to other domains such as video processing, audio processing, and document processing. Thirdly, there is a need for developing robust methods that can handle highly compressed images and are resistant to a range of image processing attacks. Fourthly, exploring the use of deep learning techniques for JPEG error analysis can potentially improve the accuracy and efficiency of digital image forensics tasks. Finally, understanding the impact of emerging image compression standards such as HEVC and AV1 on JPEG error analysis can provide insights into the future direction of the field. In conclusion, the application of JPEG error analysis to digital image forensics presents exciting research opportunities with significant potential for improving the accuracy and efficiency of digital image processing tasks.

References:

1. Kaur, H., & Kaur, P. (2021). A High Capacity Reversible Data Hiding Scheme for Encrypted Images using JPEG Error Analysis. IEEE Transactions on Information Forensics and Security, 16, 2560-2571. doi: 10.1109/TIFS.2021.3079677
2. Wang, X., & Wang, L. (2020). A Novel Dual Watermarking Method Based on DCT and JPEG Error Analysis for Digital Images. Journal of Imaging Science and Technology, 64(1), 1-10. doi: 10.2352/J.ImagingSci.Technol.2020.64.1.010502
3. Lee, K., Lee, J., & Lee, S. (2020). Image Authentication Based on Quantization Table Estimation Using Double JPEG Compression and Error Analysis. Journal of Information Processing Systems, 16(4), 870-881. doi: 10.3745/JIPS.03.0141
4. Li, C., & Li, X. (2019). An Efficient Watermarking Scheme for Color Images Based on SVD, DWT, and JPEG Error Analysis. IEEE Transactions on Circuits and Systems for Video Technology, 29, 2363-2375. doi: 10.1109/TCSVT.2018.2878195
5. Wen, C., Wang, J., & Wang, Y. (2018). Forgery Detection in Digital Images Based on JPEG Compression Analysis. Multimedia Tools and Applications, 77, 21655-21670. doi: 10.1007/s11042-018-6394-4
6. Thakur, N., & Nigam, R. (2017). A Novel Approach for Image Steganography using DCT and JPEG Error Correction. International Journal of Advanced Research in Computer Science and Software Engineering, 7(5), 1-6. doi: 10.23956/ijarcsse/V7I5/0026
7. Kim, H., & Lee, S. (2016). Reversible Data Hiding in JPEG Images Based on Prediction Error Expansion and Error Analysis. Journal of Information Processing Systems, 12(1), 1-14. doi: 10.3745/JIPS.03.0039
8. Khandelwal, N., & Kankane, R. (2015). Detecting Copy-Move Forgery in Digital Images by means of JPEG Compression Level and DWT. International Journal of Computer Science and Mobile Computing, 4(7), 306-316.

9. Singh, S., & Agrawal, A. (2014). Digital Image Forgery Detection using Improved JPEG Compression Detection and Blocking Artifact Analysis. Journal of Advances in Computer Networks, 2(3), 156-162.
10. Lin, J., & Liu, H. (2013). Detection of Double Compression in JPEG Images for Applications in Image Forensics. Journal of Visual Communication and Image Representation, 24, 129-139. doi: 10.1016/j.jvcir.2012.09.005