CS355 - Digital Forensics
Report for labs 4-6

Mikas Michelevičius
ID: 1835495

March 15, 2021

Contents

1 Lab 4

1.1 Exercise 1

In this exercise, the main lossy steps in JPEG compression are implemented. Lena image is being used for compression. First the 2D DCT is applied to the original image, then the coefficients below the antidiagonal are removed. Both DCT matrices are displayed, see Fig 1.

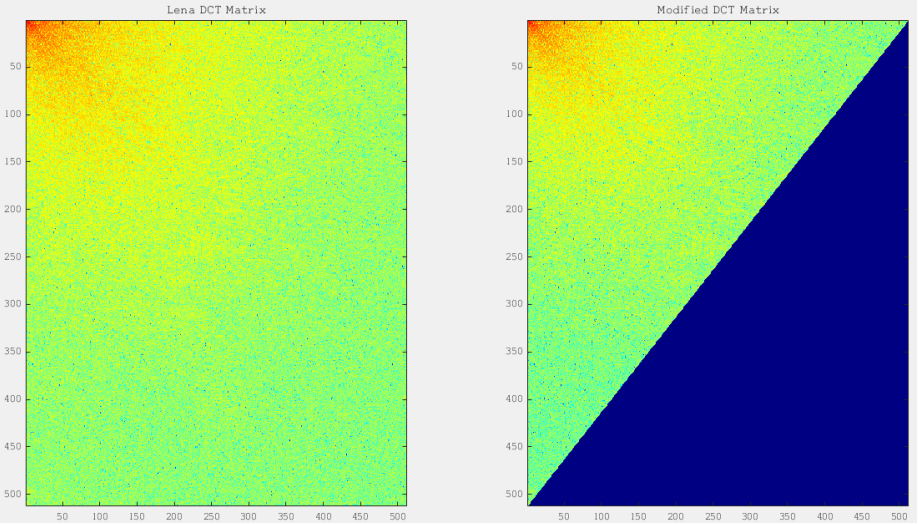


Figure 1: Lena DCT Matrix and Modified Matrix

In the first case, the DCT coefficients below the antidiagonal are removed, then the inverse DCT function is applied and the image is reconstructed, see Fig 2 (a). For the second case, the same 2D DCT function is applied, only the image is first split into 8x8 sized blocks and the function is applied on each block. Then the coefficients are removed below antidiagonal for each block. The inverse DCT function is then performed on each block and the reconstructed image is again received, see Fig 2 (a).

For the last two cases, the image is split into 8x8 blocks as well. This time the provided quantization matrices **Q50** and **Q90** are used to perform quantization. 2D DCT function is once again applied on each block, each 8x8 block is then quantized using one of the matrices. After the quantization the inverse DCT is performed to reconstruct the images. The images are then displayed, see Fig 2 (b).

As can be seen visually, the compression with removing DCT coefficients below the antidiagonal in both cases will not change the image too much, compared to the original image. However, the changes can be seen by eye after performing quantization on the original image in both cases.

After performing compression, the compression ratio is first computed by

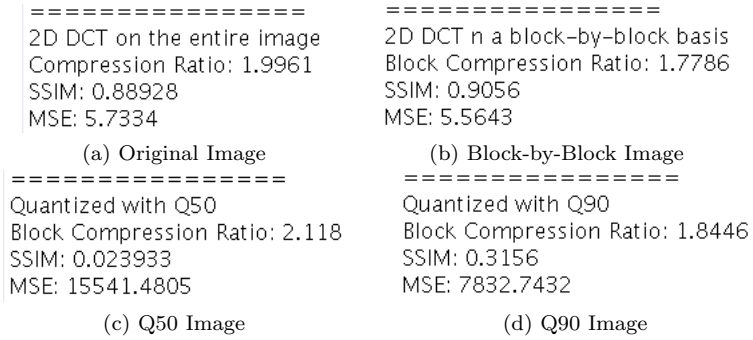
(a) Compressed Original and Compressed Block-by-Block

(b) Quantized Q50 and Quantized Q90

Figure 2: Reconstructed Images

counting the non-zero pixels in the DCT matrix and performing the division between the original image count and compressed image count. The best compression ratio was received after compression by quantization with Q50 matrix (Fig 3. (c)), however, it also provided with the worst SSIM and MSE values.

The best SSIM and MSE values were received after compression by removing the DCT coefficients below antidiagonal on a block-by-block basis (Fig 3. (b)), also providing with the worst compression ratio.



(a) Original Image (b) Block-by-Block Image

(c) Q50 Image (d) Q90 Image

Figure 3: Compression Statistics

1.2 Exercise 2

In exercise 2 of lab4 spliced image detection was implemented. The task was to detect any, if exists, JPEG ghosts on four images. Function **jpeg_ghosts** was implemented, which takes 5 variables as arguments - **file, b, minQ, maxQ, stepQ**. Where file is the file name of the image, b is the size of **b** x **b** window used to average the differences, **minQ** and **maxQ** are the minimum and maximum qualities to be tested and **stepQ** is the increment in qualities. Function returns **diffImages** cell array used to store all the difference images computed.

When the program is running, depending on the b value and the size of the image, it will first pad the image provided if necessary. The **diffImages** array will be initialized of size (x,y,n), where x and y corresponds to the input image size, and n is the number of quality iterations. Thus after the program is done running, the 3D array will be returned containing n 2D difference images.

The difference images computation happens in the big nested for loop, where the spatially averaged difference measure is calculated. Each pixel is averaged a **b** x **b** pixel region. Each difference image is stored in the **diffImages** array.

After performing spliced image detection on all four provided images, JPEG ghost was detected only in the single image **splicedbeach.jpg** (Fig 4) where the ghost is present at the quality 70. No JPEG ghosts were detected in the remaining three images.

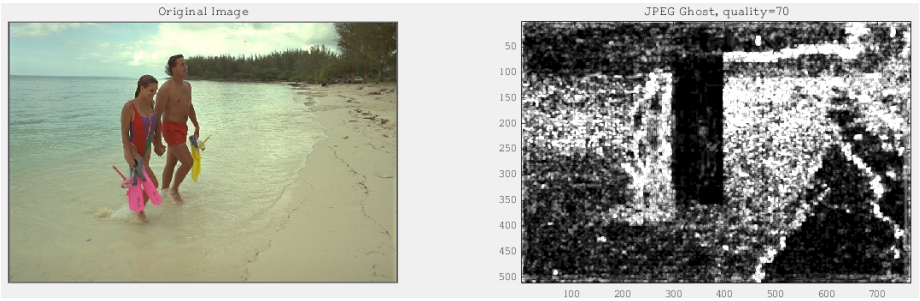


Figure 4: Spliced Image and Difference Image with Ghost

2 Lab 5

2.1 Exercise 1

The function **imCshift** is implemented in the **circMatch.m** script, where the function takes three arguments **X, k, l**, where X is an image to be shifted, and k and l are the numbers of pixels to be shifted row-wise and column-wise respectively. The shifting is performed by two nested for loops inside the function and returns a shifted image. Grayscale Lena image was used to test two different shifts, one with arguments k=64, l=64 and another one with k=128, l=128, the results are displayed, see Fig 5.

(a) k=64, l=64 (b) k=128, k=128

Figure 5: Lena Rotation Test

The function **simThresh** is also implemented in the program which takes three arguments **X, S, t**, where X is the original image, S is the shifted image and

a threshold t. The function returns a binary image **d** which is the thresholded difference image.

Having these two functions implemented, circular shift & match method can now be implemented as well. The tampered regions are being detected in the nested for loop for variables k and l in the range - $1 \leq k \leq M/2$, $1 \leq l \leq N/2$. For every loop iteration shifted image is computed by calling **imCshift** function. Then the difference threshold image is computed by calling **simThresh** function by using threshold value **t**=5. The binary image is then **eroded** and **dilated** with a **diamond** structuring image of size 11x11 which produces a new binary image. Then the **A** binary image is being updated. After performing such commands for each k and l iteration, binary image of detected tampered region is received.

The curcular shift & match method was applied on the *jeep.png* image (Fig 6). The output image of detected tampered region with the highlighted pixel locations in the given image is displayed, see Fig 7.

Figure 6: Jeep Image Grayscale

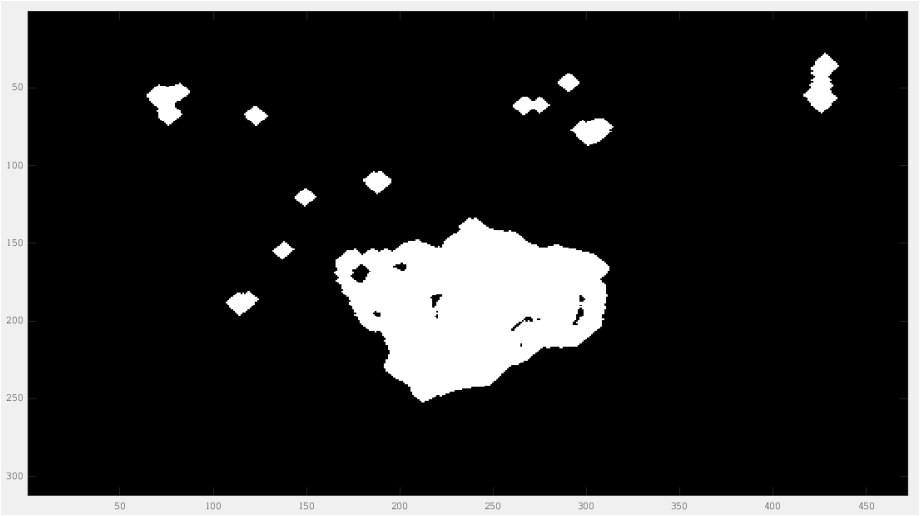


Figure 7: Tampered Region

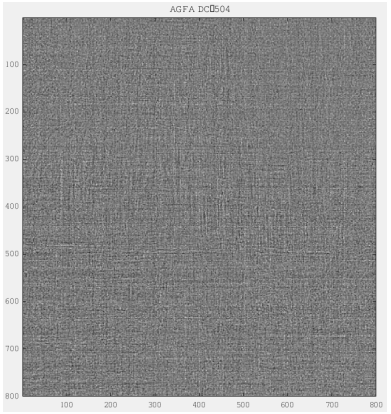
3 Lab 6

3.1 Exercise 1

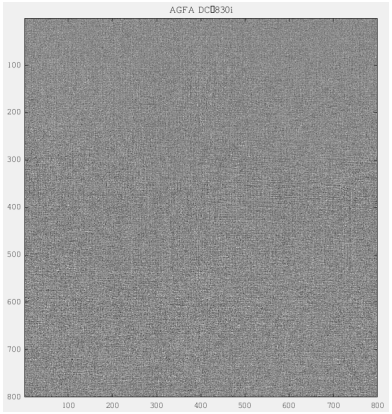
When running the **matchSPN.m** script, program will first compute the reference SPN of each camera. It will be done within six separate for loops. Each loop will open and read all 50 images for each camera in their respective directory. Each image is read using **imread** command and converted to grayscale. After denoising, using **wiener2** filter, denoised version is subtracted from the original image to calculate the SPN, only 800x800 central region of SPN is used. Then the reference SPN for each camera is calculated by averaging the noise residuals of each image. The calculated reference SPN of each camera is then displayed, see Fig 8.

In the same way as previously done with reference images, SPNs for test images are calculated in the final for loop, using only the 800x800 central region of the SPN as well. Once the SPN is computed, correlation coefficient is calculated between test image SPN and all 6 reference SPNs of each camera. The SPNs are centered by dividing mean value of SPN from SPN matrix before calculating the correlation coefficient. Once all the coefficients are calculated, the program classifies the images to the respective camera by the highest correlation coefficient and the results are displayed, see Fig 9.

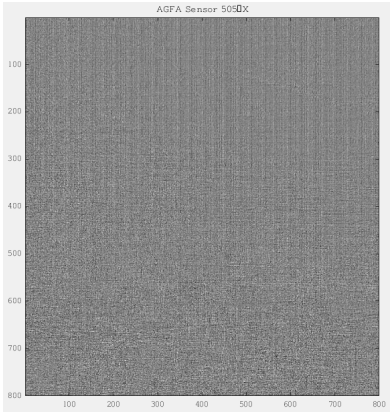
As it can be seen from the output received, two images were misclassified. The images that were misidentified are **350728.JPG** and **625619.JPG**. With that being said, 28 out of 30 images were classified correctly giving the matching accuracy of **93%**.



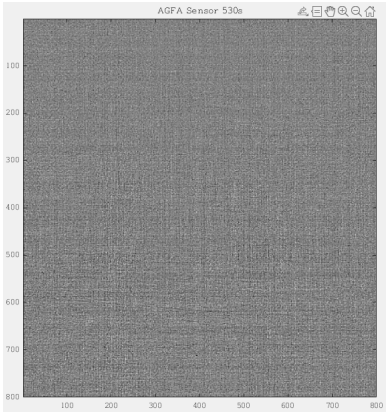
(a) AGFA DC-504



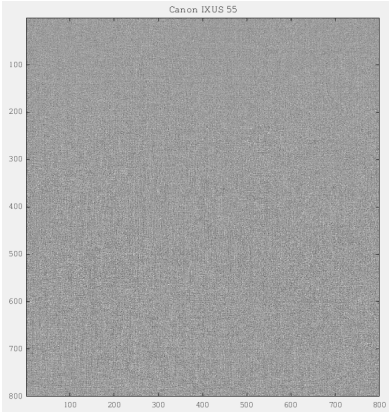
(b) AGFA DC-830i



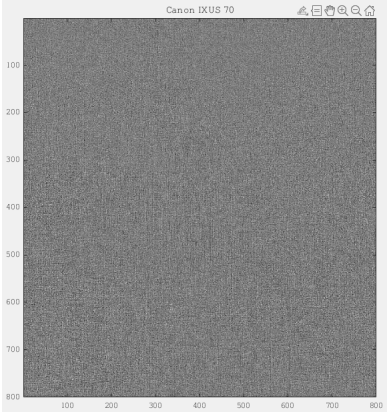
(c) AGFA Sensor 505-X



(d) AGFA Sensor 530s



(e) Canon IXUS 55



(f) Canon IXUS 70

Figure 8: Reference SPN Figures

```
Test image       Class
---------------------
081126.JPG   AGFA DC-504
350728.JPG   AGFA DC-504
775713.JPG   AGFA DC-504
780228.JPG   AGFA DC-504
929386.JPG   AGFA DC-504
---------------------
306350.JPG   AGFA DC-830i
435859.JPG   AGFA DC-830i
446784.JPG   AGFA DC-830i
486792.JPG   AGFA DC-830i
508509.JPG   AGFA DC-830i
---------------------
378610.JPG   AGFA Sensor 505-X
510772.JPG   AGFA Sensor 505-X
644319.JPG   AGFA Sensor 505-X
794832.JPG   AGFA Sensor 505-X
817628.JPG   AGFA Sensor 505-X
---------------------
532826.JPG   AGFA Sensor 530s
811581.JPG   AGFA Sensor 530s
875943.JPG   AGFA Sensor 530s
939002.JPG   AGFA Sensor 530s
---------------------
194765.JPG   Canon IXUS 70
225922.JPG   Canon IXUS 70
230489.JPG   Canon IXUS 70
470924.JPG   Canon IXUS 70
844309.JPG   Canon IXUS 70
---------------------
207743.JPG   Canon IXUS 55
301247.JPG   Canon IXUS 55
550157.JPG   Canon IXUS 55
587045.JPG   Canon IXUS 55
622476.JPG   Canon IXUS 55
625619.JPG   Canon IXUS 55
```

Figure 9: Classification output