

EXPERIMENT TS: Image Processing

{ Student: Haixiang Huang

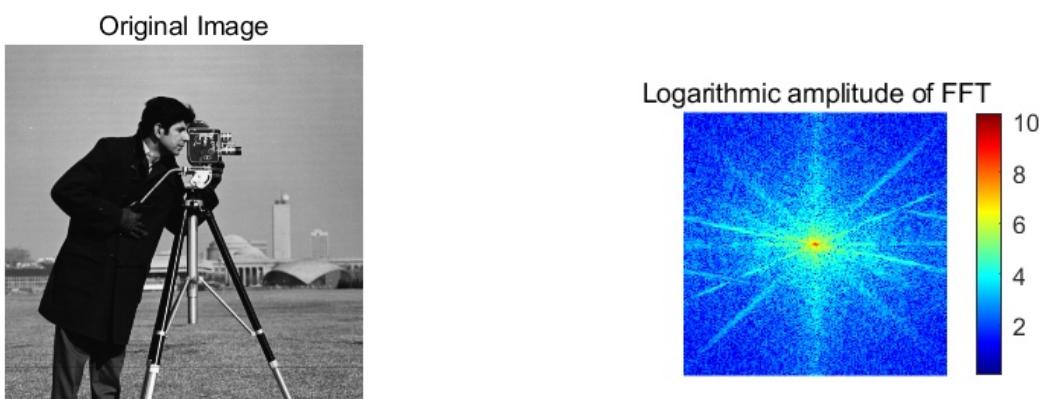
CID: 02470313

{ Demonstrator: Siying Liu

Part A: Image transformation

1. FFT

Part 1

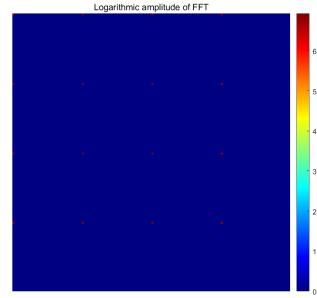
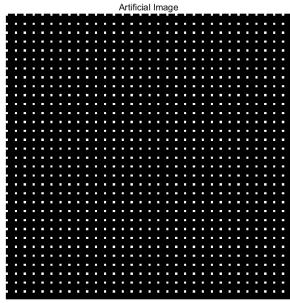


LHS is the original image, RHS is the log amplitude of FFT (shifted to the middle).

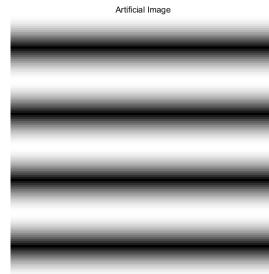
We can find that the low index usually possess higher values compared to the coefficients with high index. This is because the natural images

are very smooth.

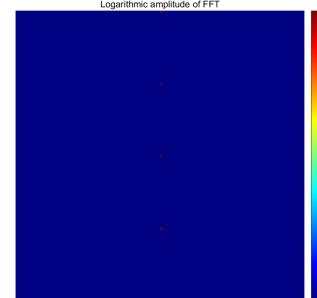
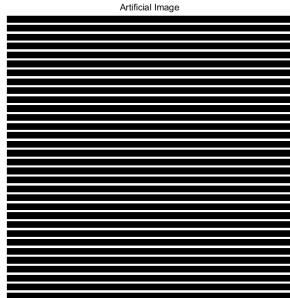
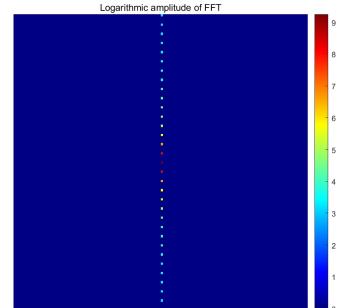
Part 2



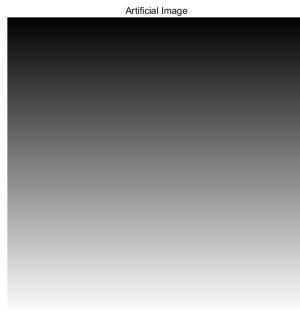
(1)



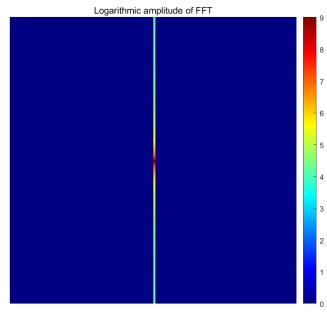
(2)



(3)



(4)



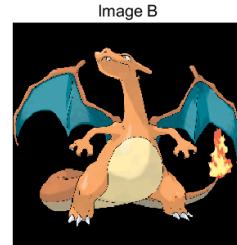
(1) the Artificial image of a grid of impulses shows the DFT is also impulse-like

(2)(3) is the periodic image, their DFT is discrete.

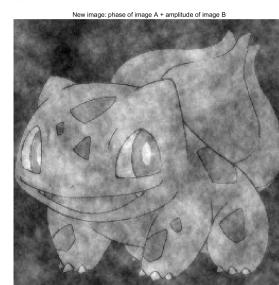
(4) is the non periodic image, its DFT is continuous

Part 3

original



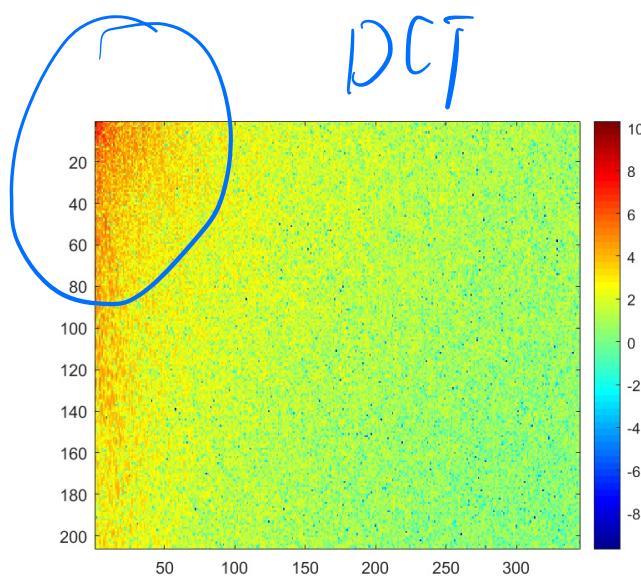
Phase A + amplitude B



The phase of image indicates the distribution of frequency of the image. The high frequency components of the images are preserved by the phase. Therefore, we can recognize the image content based on phase.

2. DCT

Original

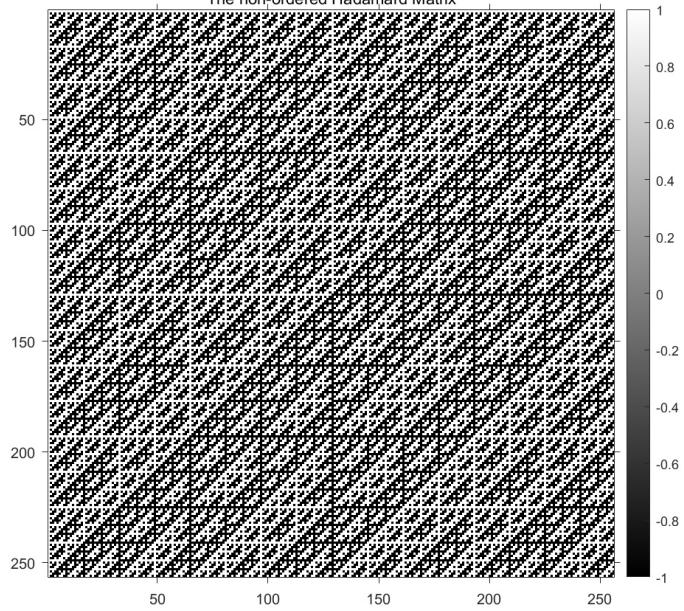


DCT low frequency components are located at the top-left corner while FFT are located at four corners.

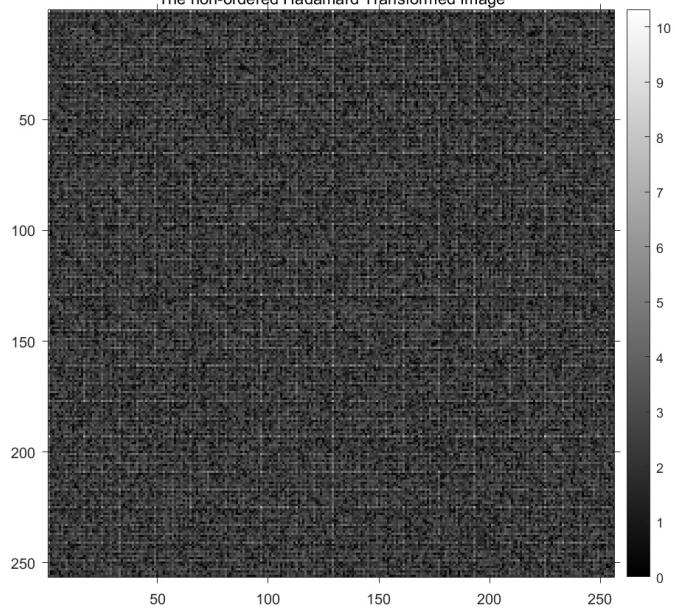
3. Hadamard

- Pros:
1. Computational efficiency
 2. Fast Algorithm like Fast-FFT
 3. Orthogonality of basic vector
 4. Noise Resilience

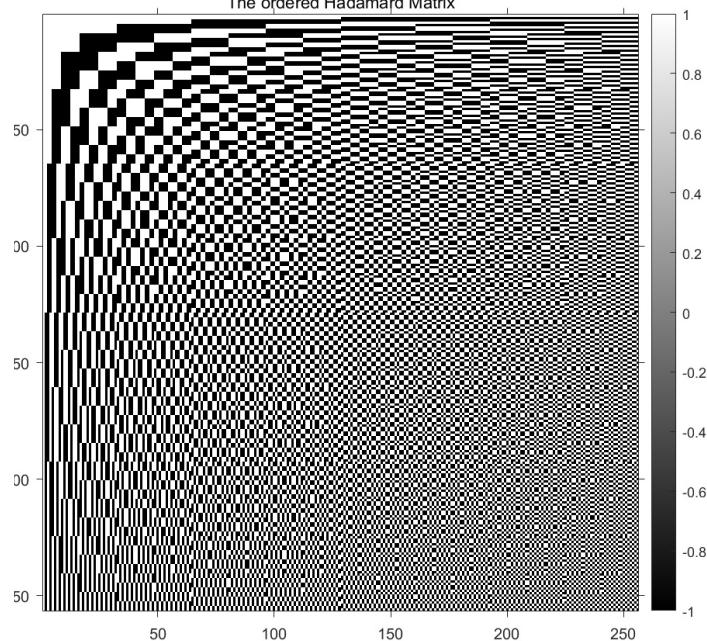
The non-ordered Hadamard Matrix



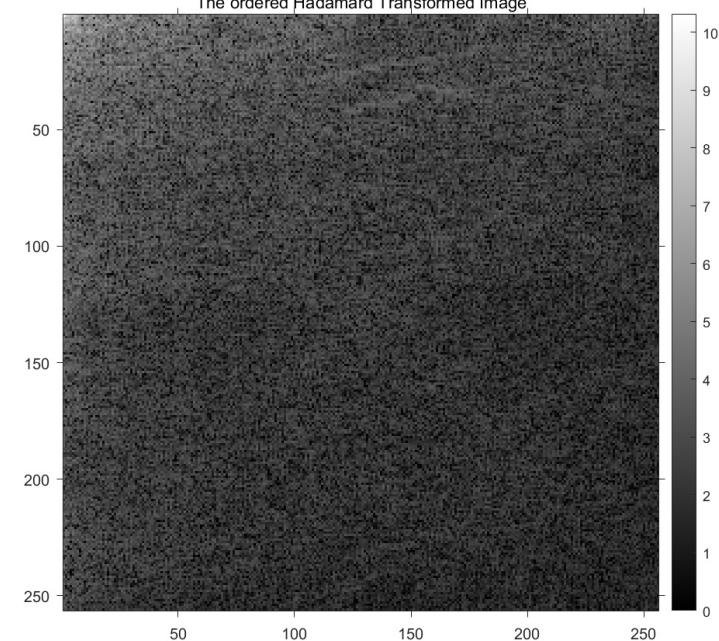
The non-ordered Hadamard Transformed Image



The ordered Hadamard Matrix

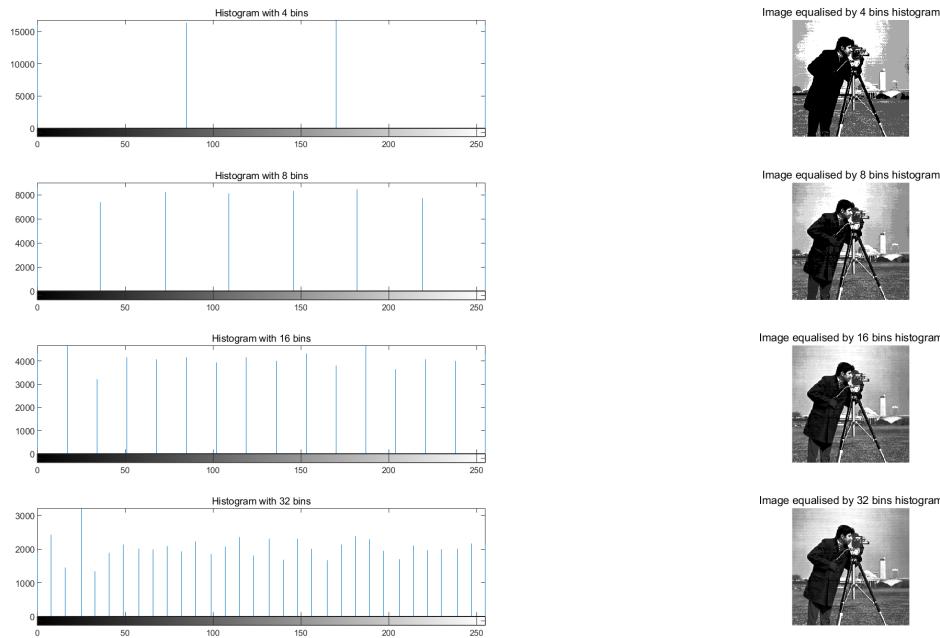


The ordered Hadamard Transformed Image

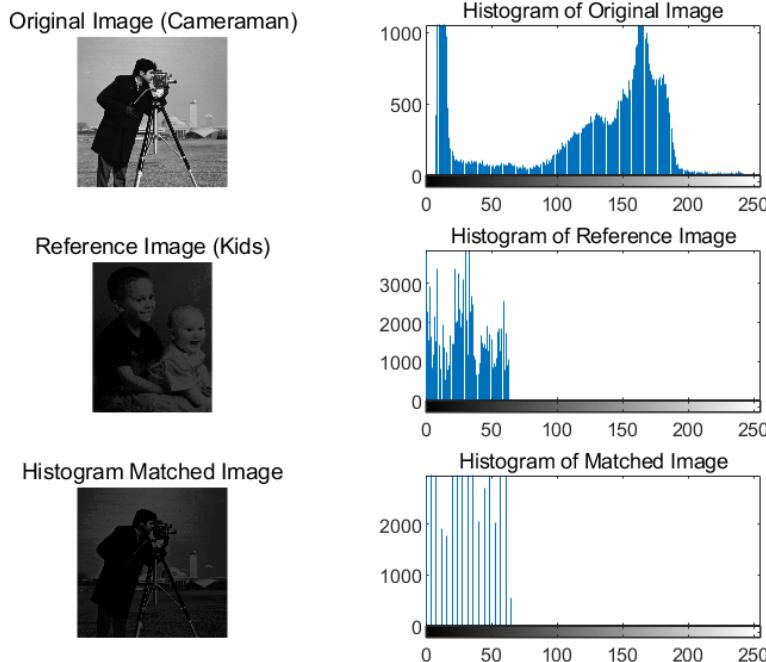


Part B Image Enhancement

1. Histogram Equalisation



Part2



we can see the result hist of matched image is similar to the reference image.

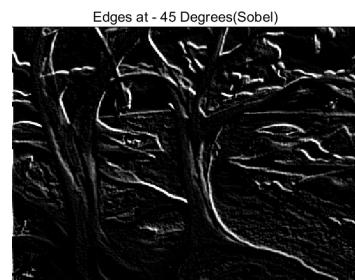
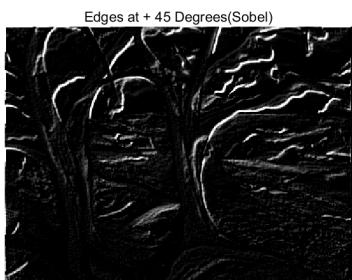
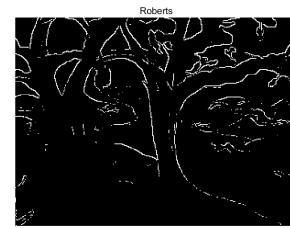
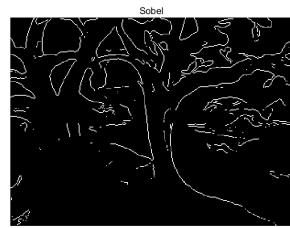
Part3 edge detection

Customed 45° sobel mask

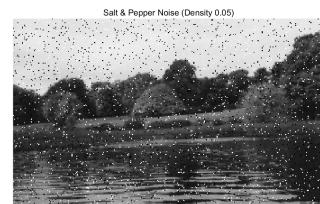
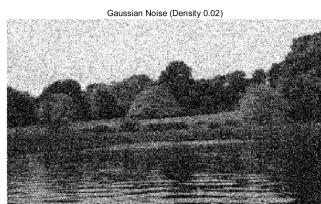
$$\begin{pmatrix} 0 & 1 & 2 \\ -1 & 0 & 1 \\ -2 & -1 & 0 \end{pmatrix}$$

Customed -45° Sobel mask

$$\begin{pmatrix} -2 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & 2 \end{pmatrix}$$



Part 4 Median Filter



We can find that the larger kernel, the result becomes more blur

Part C Image Compression



```
% Process each 8x8 block
for i = 1:blockSize:rows
    for j = 1:blockSize:cols
        % Extract the block
        block = I(i:min(i+blockSize-1, rows), j:min(j+blockSize-1, cols));

        % Apply DCT to the block
        J = dct2(block);

        % Apply the threshold
        J(abs(J) < threshold) = 0;

        % Perform IDCT
        block = idct2(J);

        % Place the processed block back into the image
        K(i:min(i+blockSize-1, rows), j:min(j+blockSize-1, cols)) = block;
    end
end
```

DCT coefficient less than threshold
will be set to 0.

Change threshold and repeat above process, we can get the image above. We find the large threshold, the image is more blurred.

Part D: Design Exercise

```
function K = fDCT_Tx(I, blockSize, compressedRatio)

[rows, cols] = size(I);
K = zeros(rows, cols);

% Calculate the number of coefficients to keep based on the compressed ratio
numCoefficients = blockSize * blockSize / compressedRatio;
numCoefficientsPerDimension = sqrt(numCoefficients);

for i = 1:blockSize:rows
    for j = 1:blockSize:cols
        % Extract the block
        block = I(i:min(i+blockSize-1, rows), j:min(j+blockSize-1, cols));

        % Apply DCT
        J = dct2(block);

        % Zero out all but the top left 'numCoefficientsPerDimension x numCoefficientsPerDimension' coefficients
        J(numCoefficientsPerDimension+1:end, :) = 0;
        J(:, numCoefficientsPerDimension+1:end) = 0;

        % Replace the block in the compressed image
        K(i:min(i+blockSize-1, rows), j:min(j+blockSize-1, cols)) = J;
    end
end
```

Based on the compressed ratio, we will zero out all but top-left.

Rx-IDCT:

```
function K = fIDCT_Rx(J, blockSize)
[rows, cols] = size(J);
K = zeros(rows, cols);
for i = 1:blockSize:rows
    for j = 1:blockSize:cols
        block = J(i:min(i+blockSize-1, rows), j:min(j+blockSize-1, cols));
        % Inverse DCT
        block = idct2(block);

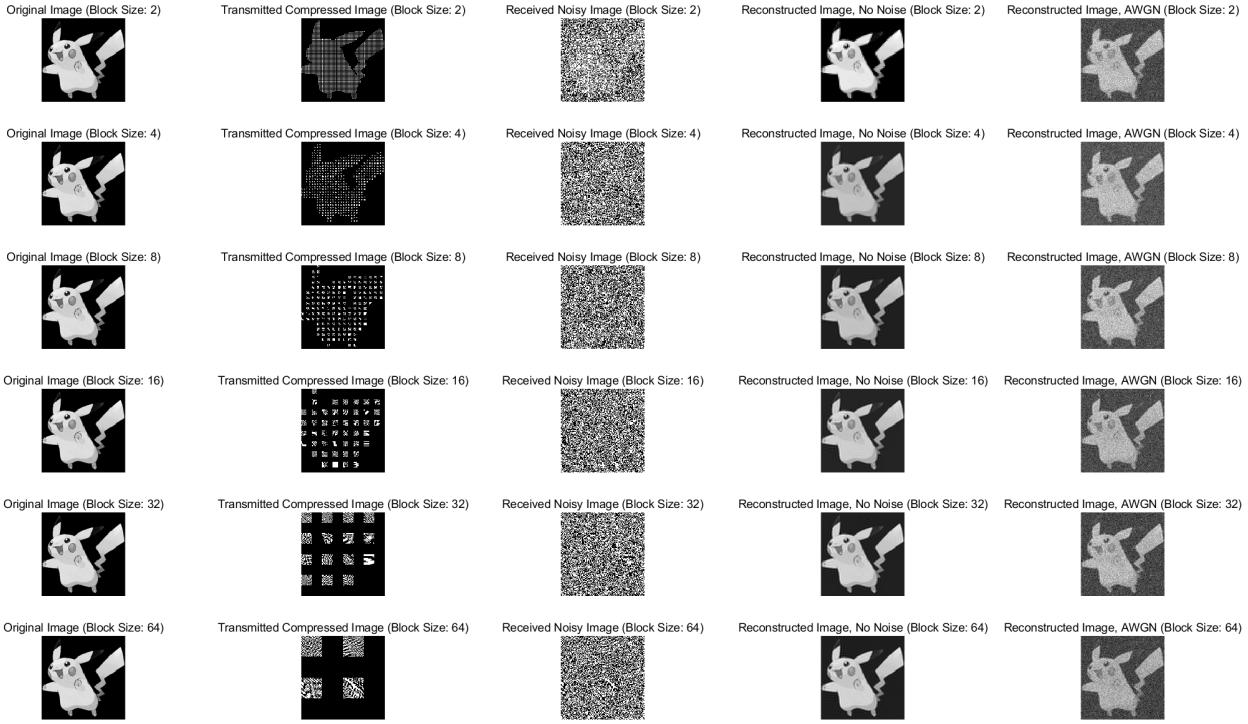
        % Replace the block
        K(i:min(i+blockSize-1, rows), j:min(j+blockSize-1, cols)) = block;
    end
end

% Normalize the image
K = mat2gray(K);
end
```

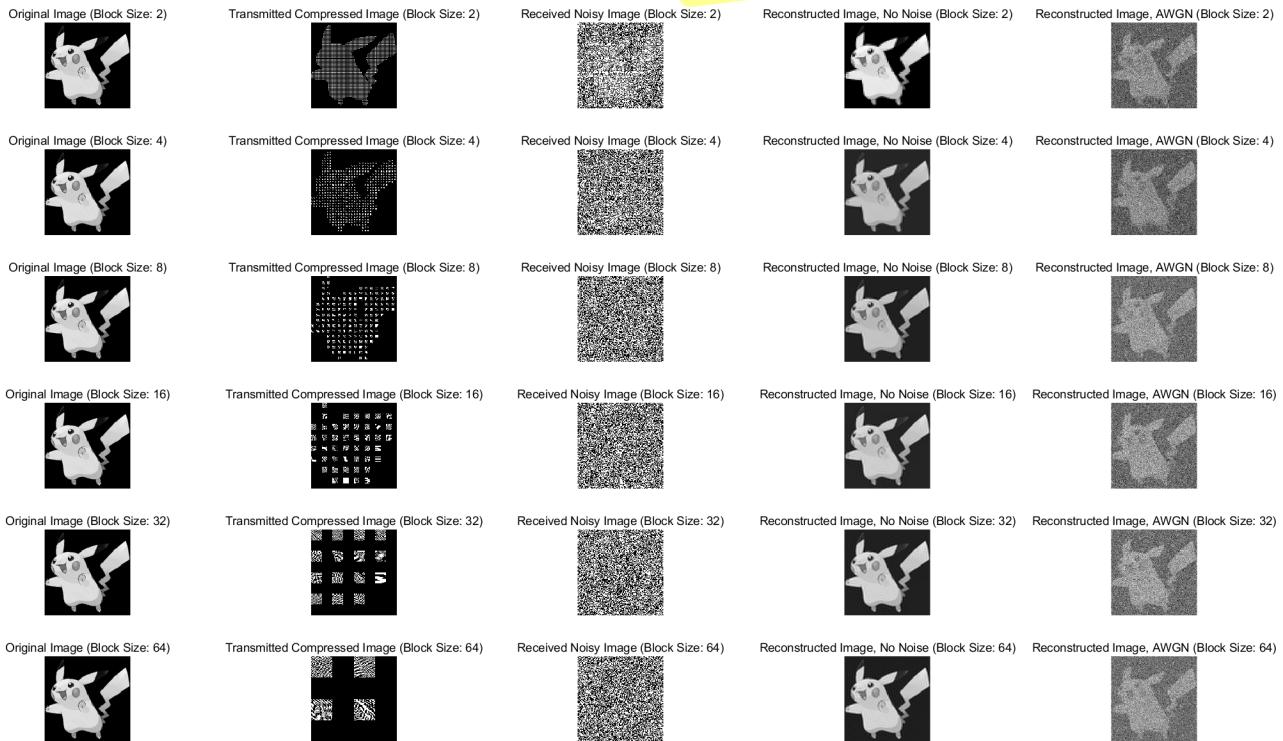
Results:

The compressed image by fDCT-TX will through the 10dB AWGN channel, and received Image will be reconstructed by fIDCT-Rx using IDCT.

Images with Different Block Sizes (SNR: 10 dB)

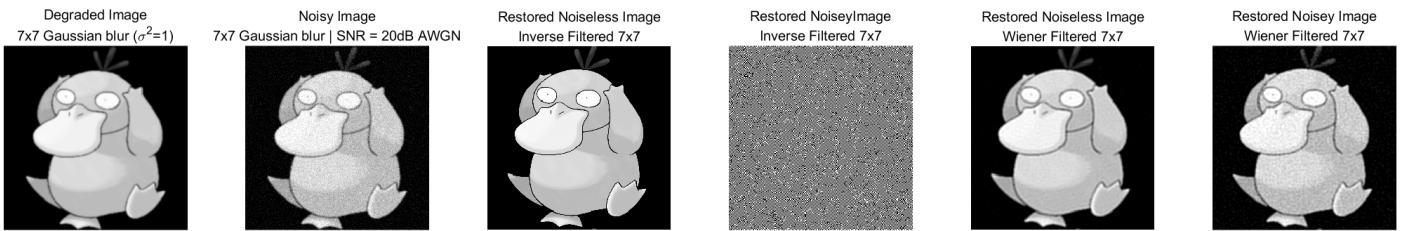
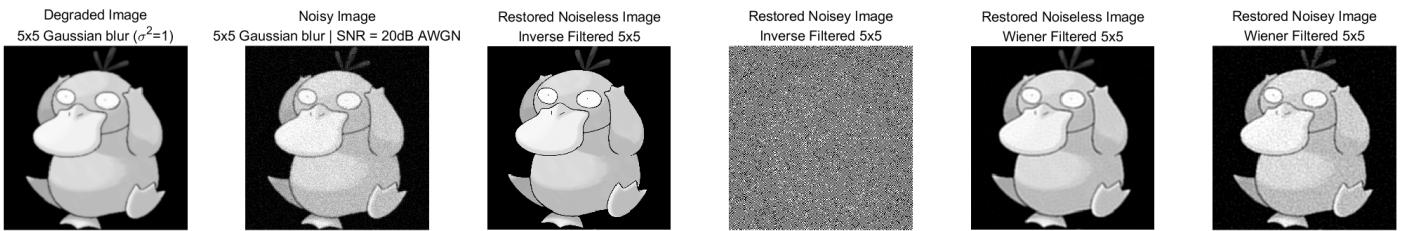


Images with Different Block Sizes (SNR: 0 dB)



Part E Image Restoration

Degraded Images with Different Filtering (SNR: 20 dB)



We can find that Inverse Filtered fails to process degraded image with AWGN. While Wiener Filtered Process it well.