hw2

21020462 Phùng Thành Dat

April 2024

1 Practice 1 : Image Filtering

This is padding image function

```
def padding_img(img, filter_size=3):
    """
    The surrogate function for the filter functions.
    The goal of the function: replicate padding the image such that when
        applying the kernel with the size of filter_size, the padded
        image will be the same size as the original image.
    WARNING: Do not use the exterior functions from available libraries
        such as OpenCV, scikit-image, etc. Just do from scratch using
        function from the numpy library or functions in pure Python.
    Inputs:
       img: cv2 image: original image
       filter_size: int: size of square filter
    Return:
       padded_img: cv2 image: the padding image
    """
    # Get original image dimensions
    height, width = img.shape[:2]

    # Calculate padding size
    pad_size = filter_size // 2

    # Create padded image array
    padded_img = np.zeros((height + 2 * pad_size, width + 2 * pad_size),
        dtype=img.dtype)

    # Copy original image into padded image
    padded_img[pad_size:pad_size + height, pad_size:pad_size + width] =
        img

    # Replicate padding for borders
    padded_img[:pad_size, pad_size:pad_size + width] = img[0, :]
    padded_img[pad_size + height:, pad_size:pad_size + width] =
        img[height - 1, :]
```

```
    padded_img[:, :pad_size] = padded_img[:, pad_size:2 * pad_size]
    padded_img[:, pad_size + width:] = padded_img[:, pad_size + width -
        1:pad_size + width]

    return padded_img
```

This is mean filtering function

```
def mean_filter(img, filter_size=3):
    """
    Smoothing image with mean square filter with the size of
        filter_size. Use replicate padding for the image.
    WARNING: Do not use the exterior functions from available libraries
        such as OpenCV, scikit-image, etc. Just do from scratch using
        function from the numpy library or functions in pure Python.
    Inputs:
        img: cv2 image: original image
        filter_size: int: size of square filter,
    Return:
        smoothed_img: cv2 image: the smoothed image with mean filter.
    """
    padded_img = padding_img(img, filter_size)
    height, width = img.shape
    smoothed_img = np.zeros_like(img)

    for i in range(height):
        for j in range(width):
            smoothed_img[i, j] = np.mean(padded_img[i:i + filter_size,
                j:j + filter_size])

    return smoothed_img
```

This is median filtering function

```
def median_filter(img, filter_size=3):
    """
        Smoothing image with median square filter with the size of
            filter_size. Use replicate padding for the image.
        WARNING: Do not use the exterior functions from available
            libraries such as OpenCV, scikit-image, etc. Just do from
            scratch using function from the numpy library or functions
            in pure Python.
        Inputs:
            img: cv2 image: original image
            filter_size: int: size of square filter
        Return:
            smoothed_img: cv2 image: the smoothed image with median
```

```
            filter.
    """
    padded_img = padding_img(img, filter_size)
    height, width = img.shape
    smoothed_img = np.zeros_like(img)

    for i in range(height):
        for j in range(width):
            smoothed_img[i, j] = np.median(padded_img[i:i + filter_size,
                j:j + filter_size])

    return smoothed_img
```

Function PRSN

```
def psnr(gt_img, smooth_img):
    """
        Calculate the PSNR metric
        Inputs:
            gt_img: cv2 image: groundtruth image
            smooth_img: cv2 image: smoothed image
        Outputs:
            psnr_score: PSNR score
    """
    mse = np.mean((gt_img - smooth_img) ** 2)
    if mse == 0:
        return float('inf')
    max_pixel = 255.0
    psnr_score = 20 * math.log10(max_pixel / math.sqrt(mse))
    return psnr_score
```

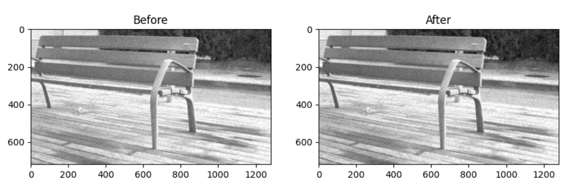

Figure 1: the result of the different of image and meanfilter image

We know Typical values for the PSNR in lossy image and video compression

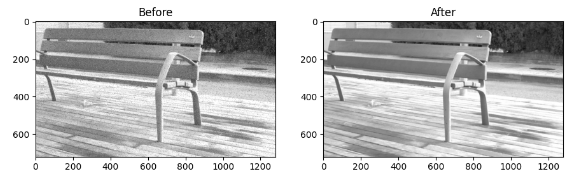

Figure 2: the result of the different of image and medianfilter image

are between 30 and 50 dB, where higher is better.

We have PSNR score of mean filter: 31.60889963499979 and PSNR score of median filter: 37.11957830085524

So we need to use media fillter.

2 Practice 2 : Fourier Transform

2.1 1D Fourier Transform

This is DFT slow function

```
def DFT_slow(data):
  """
  Implement the discrete Fourier Transform for a 1D signal
  params:
    data: Nx1: (N, ): 1D numpy array
  returns:
    DFT: Nx1: 1D numpy array
  """
  N = len(data)
  n = np.arange(N)
  k = n.reshape((N, 1))
  M = np.exp(-2j * np.pi * k * n / N)
  return np.dot(M, data)
```

2.2 2D Fourier Transform

This is DFT 2D function

```python
def DFT_2D(gray_img):
    """
    Implement the 2D Discrete Fourier Transform
    Note that: dtype of the output should be complex_
    params:
        gray_img: (H, W): 2D numpy array

    returns:
        row_fft: (H, W): 2D numpy array that contains the row-wise FFT
             of the input image
        row_col_fft: (H, W): 2D numpy array that contains the
             column-wise FFT of the input image
    """
    row_fft = np.fft.fft(gray_img, axis=1)
    row_col_fft = np.fft.fft(row_fft, axis=0)
    return row_fft, row_col_fft
```

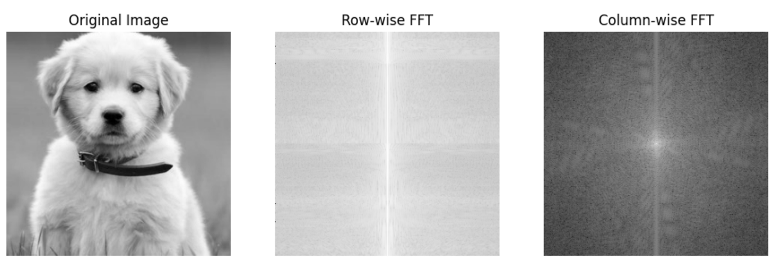

Figure 3: the result of 2D Fourier Transform exercise

2.3 Frequency Removal Procedure

This is the filter frequency function in the notebook

```python
def filter_frequency(orig_img, mask):
  """
  You need to remove frequency based on the given mask.
  Params:
    orig_img: numpy image
```

```
    mask: same shape with orig_img indicating which frequency hold or
        remove
  Output:
    f_img: frequency image after applying mask
    img: image after applying mask
  """
    # Step 1: Transform using fft2
  frequency_coefs = np.fft.fft2(orig_img)

    # Step 2: Shift frequency coefs to center using fftshift
  shifted_coefs = np.fft.fftshift(frequency_coefs)

    # Step 3: Filter in frequency domain using the given mask
  filtered_coefs = shifted_coefs * mask

    # Step 4: Shift frequency coefs back using ifftshift
  inverse_shifted_coefs = np.fft.ifftshift(filtered_coefs)

    # Step 5: Invert transform using ifft2
  filtered_img = np.fft.ifft2(inverse_shifted_coefs)

    # Take the absolute value to get the real part (ignoring imaginary
        part)
  filtered_img = np.abs(filtered_img)

  return filtered_coefs , filtered_img
```

The expected output should look similar to what is described in note titles.

2.4 Creating a Hybrid Image

This is create hybrid image function

```
def create_hybrid_img(img1, img2, r):
  """
  Create hydrid image
  Params:
    img1: numpy image 1
    img2: numpy image 2
    r: radius that defines the filled circle of frequency of image 1.
        Refer to the homework title to know more.
  """
  # Step 1: Transform images using fft2
  img1_frequency_coefs = np.fft.fft2(img1)
  img2_frequency_coefs = np.fft.fft2(img2)

  # Step 2: Shift frequency coefs to center using fftshift
```

```python
img1_shifted_coefs = np.fft.fftshift(img1_frequency_coefs)
img2_shifted_coefs = np.fft.fftshift(img2_frequency_coefs)

# Step 3: Create a mask based on the given radius (r)
mask = np.zeros(img1.shape)
rows, cols = mask.shape
center_row, center_col = rows // 2, cols // 2
for i in range(rows):
    for j in range(cols):
        distance = np.sqrt((i - center_row) ** 2 + (j - center_col) **
            2)
        if distance <= r:
            mask[i, j] = 1

# Step 4: Combine frequency of 2 images using the mask
hybrid_shifted_coefs = img1_shifted_coefs * mask + img2_shifted_coefs
    * (1 - mask)

# Step 5: Shift frequency coefs back using ifftshift
hybrid_inverse_shifted_coefs = np.fft.ifftshift(hybrid_shifted_coefs)

# Step 6: Invert transform using ifft2
hybrid_img = np.fft.ifft2(hybrid_inverse_shifted_coefs)

# Take the absolute value to get the real part (ignoring imaginary
    part)
hybrid_img = np.abs(hybrid_img)

return hybrid_img
```

The expected output should look similar to what is described in note titles.