1. Question

Adding **Salt & Pepper Noise**

Processing noisy image with Mean and Median Filter（**Filter size**: 3x3、5x5）

Show six images as below: Original image, adding noise, 3\*3(5\*5) mean and median filtering

(Try different filter size, if possible)

1. Code

* Whole Code

import cv2

import numpy as np

from random import randint

# Read, make grayscale ============================================

jpg = "bear.jpg"

img = cv2.imread(jpg)

r,g,b = img[:,:,0], img[:,:,1], img[:,:,2]

grayscale = r\*0.299 + g\*0.587 + b\*0.114

# Seasoning the image =============================================

seasoned = np.copy(grayscale)

y,x = seasoned.shape

for yy in range(y):

  for xx in range(x):

    toSeasonOrNotToSeason = randint(0,255)

    if(toSeasonOrNotToSeason == 0 or toSeasonOrNotToSeason == 255):

      seasoned[yy][xx] = toSeasonOrNotToSeason

cv2.imshow("Original Grayscale", grayscale/255)

cv2.imshow("Seasoned Image", seasoned/255)

cv2.waitKey(0)

cv2.destroyAllWindows()

# preparing functions ====================================================

def mirrorPadding(size, array):

  new = np.copy(array)

  new = np.pad(new,((size,size),(size,size)),'reflect')

  return new

def meanFilter(size, array):

  y,x = array.shape

  mirrored = mirrorPadding(size,array)

  filtered = np.copy(array)

  for yy in range(size,size+y):

    for xx in range(size,size+x):

      mask = mirrored[yy-int(size/2):yy+int(size/2)+1,

                      xx-int(size/2):xx+int(size/2)+1]

      sum = 0

      for i in mask:

        for j in i:

          sum+=j

      filtered[yy-size][xx-size] = sum / (size\*size)

  cv2.imshow(f"before Mean {size}\*{size}", array/255)

  cv2.imshow(f"after Mean {size}\*{size}", filtered/255)

  cv2.waitKey(0)

  cv2.destroyAllWindows()

  return filtered

def medianFilter(size, array):

  y,x = array.shape

  mirrored = mirrorPadding(size,array)

  filtered = np.copy(array)

  for yy in range(size,size+y):

    for xx in range(size,size+x):

      mask = mirrored[yy-int(size/2):yy+int(size/2)+1,

                      xx-int(size/2):xx+int(size/2)+1]

      maskarr = []

      for i in mask:

        for j in i:

          maskarr.append(j)

      maskarr.sort()

      filtered[yy-size][xx-size] = maskarr[int((size\*size)/2)]

  cv2.imshow(f"before Median {size}\*{size}", array/255)

  cv2.imshow(f"after Median {size}\*{size}", filtered/255)

  cv2.waitKey(0)

  cv2.destroyAllWindows()

  return filtered

# executing the code =====================================================

meanFilter(3,seasoned)

meanFilter(5,seasoned)

medianFilter(3,seasoned)

medianFilter(5,seasoned)

* Explanation
  + Libraries Used

import cv2

import numpy as np

from random import randint

These are the libraries used for this code. Random was used to add Salt and Pepper noise, numpy was used to edit the image arrays (such as copying and padding etc). Cv2 was used to read and show the images which were in the form of numpy arrays.

* + Reading The Image

jpg = "bear.jpg"

img = cv2.imread(jpg)

The image was renamed as bear.jpg by me before I ran the code so I put that in a variable to be read later in the second line. The image that has been read is then stored to the variable “img” as a numpy array.

* + Gray scaling The Image

r,g,b = img[:,:,0], img[:,:,1], img[:,:,2]

grayscale = r\*0.299 + g\*0.587 + b\*0.114

Creating a grayscale image from img. Although the original image is black and white but the formatting is still rgb so it consists of 3 2D arrays according to each colour channel. The image is then put through a formula to create a new grayscale image in the variable grayscale.

* + Seasoning The Image

seasoned = np.copy(grayscale)

y,x = seasoned.shape

We copy the grayscaled image to a new variable to store the salt and peppered image, this variable is named seasoned. Then we store the dimensions of the image to later be iterated.

for yy in range(y):

  for xx in range(x):

    toSeasonOrNotToSeason = randint(0,255)

    if(toSeasonOrNotToSeason == 0 or

toSeasonOrNotToSeason == 255):

      seasoned[yy][xx] = toSeasonOrNotToSeason

We iterate the array to season the image pixel by pixel. This is done randomly by choosing a random number from 0-255. The chances of a pixel being salted(turned white) or peppered(turned black) in this case is 1/256 for each salt and pepper. If the random number turns out to be 255 or 0, the current pixel being iterated will be changed to that random number.

cv2.imshow("Original Grayscale", grayscale/255)

cv2.imshow("Seasoned Image", seasoned/255)

cv2.waitKey(0)

cv2.destroyAllWindows()

Showing both seasoned image and original grayscale image. Both arrays are divided by 255 because openCV’s imshow function shows the grayscale range from 0 to 1 instead of from 0 to 255. After a key is pressed, it will destroy all current windows.

* + Mirror Padding

def mirrorPadding(size, array):

  new = np.copy(array)

  new = np.pad(new,((size,size),(size,size)),'reflect')

  return new

I created a function to return a mirror padded version of the array. It takes in the parameters of the width of padding that we desire and the image array itself. The first line of the inside of the function copies the image that we want to a new array. That array is then padded by the width that is desired on it’s 4 sides (up, down, left, right) with a padding mode reflect which mirrors the image within the padding. Then the function returns the padded image.

* + Mean Filter

def meanFilter(size, array):

  y,x = array.shape

  mirrored = mirrorPadding(size,array)

  filtered = np.copy(array)

I created a function to filter for both mean and median, both work similarly and I will explain the mean first. The function takes in the parameters of the mask size and the image that wants to be filtered. We store the dimensions of the image to later iterate it. We then create a mirror padded image to base the mask and create a new array to store the result of the filtering.

  for yy in range(size,size+y):

    for xx in range(size,size+x):

      mask = mirrored[yy-int(size/2):yy+int(size/2)+1,

                      xx-int(size/2):xx+int(size/2)+1]

      sum = 0

      for i in mask:

        for j in i:

          sum+=j

      filtered[yy-size][xx-size] = sum / (size\*size)

We iterate the middle part of the mirrored image by the variables yy and xx. Then we create the mask from the mirrored image by slicing the image on it’s surrounding area. Then we calculate the mean of the values within the mask by summing them all up and dividing them. Then, we assign it to the filtered image’s currently iterated value.

  cv2.imshow(f"before Mean {size}\*{size}", array/255)

  cv2.imshow(f"after Mean {size}\*{size}", filtered/255)

  cv2.waitKey(0)

  cv2.destroyAllWindows()

  return filtered

These functions simply show the image both before and after the filtering, similar to how the seasoned image was shown.

* + Median Filter

def medianFilter(size, array):

  y,x = array.shape

  mirrored = mirrorPadding(size,array)

  filtered = np.copy(array)

  for yy in range(size,size+y):

    for xx in range(size,size+x):

      mask = mirrored[yy-int(size/2):yy+int(size/2)+1,

                      xx-int(size/2):xx+int(size/2)+1]

      maskarr = []

      for i in mask:

        for j in i:

          maskarr.append(j)

      maskarr.sort()

      filtered[yy-size][xx-size] = maskarr[int((size\*size)/2)]

  cv2.imshow(f"before Median {size}\*{size}", array/255)

  cv2.imshow(f"after Median {size}\*{size}", filtered/255)

  cv2.waitKey(0)

  cv2.destroyAllWindows()

  return filtered

As we can see, the median filter function is very similar to the mean filter function. It takes in the same parameters and executes similarly. The only different part of the code, which is when assigning the value of the iterated filtered image. Instead of taking the mean of the values of the mask, we take the median of it then assign it to the filtered array. We take the median value by putting all the elements of mask into a new 1d array then take it’s middle value to put in the output image. At the end of the code we show the images.

* + Executing The Functions

meanFilter(3,seasoned)

meanFilter(5,seasoned)

medianFilter(3,seasoned)

medianFilter(5,seasoned)

These lines simply call the previous functions with the designated parameters. Both 5\*5 mask and 3\*3 mask for mean and median filter functions.

1. Results
   * Seasoning The Image with Salt and Pepper

A picture containing text, bear, mammal, looking

Description automatically generated

* 3\*3 Mean Filter

A picture containing text, bear, mammal, looking

Description automatically generated

* 5\*5 Mean Filter

A picture containing text, bear, mammal, looking

Description automatically generated

* 3\*3 Median Filter

A picture containing text, bear, mammal, outdoor

Description automatically generated

* 5\*5 Median Filter

A picture containing bear, mammal, outdoor, standing

Description automatically generated