CS 415: Homework 1

Due on 09/18/24 at 11:59pm

Professor Tang 11-12:15pm

Ryan Magdaleno
rmagd2@uic.edu

Question 1

1. What is the goal of computer vision?
   To give computers the ability to process, analyze, and interpret information from images/videos. Computers in general follow set instructions so giving a computer something as subjective as "what are you looking at" is simple to us humans but much more complex when it comes to computers, the goal is to give computers that ability to extract information from an input image/video.

2. Please list three computer vision tasks (for example, face detection) and their respective applications.

   (a) **Pose Estimation:** Computers determine the spatial position/ orientation of objects/body parts from some input image/video. This is used in athletics, VTubing, face filters, motion capture.

   (b) **Scene Reconstruction:** Computer identifies key geometric / topological aspects of a image/set of images and reconstruct a 3D space that aims to be a replica of said input image(s). This is used in medicine, take a scan of someone's veins, we can construct a 3D mapping of this now. Used in crime scene reconstruction.

   (c) **Segmentation:** A Computer takes an imput image and segments the image into distinct object classifications. Take an image of a dog and cat, the computer will identify portions of the image that are of a dog and portions that are of a cat, everything else will be classified as other. This is used in self driving car programs, a auto car program needs to know what portions of the image are the road, cars, pedestrians, etc.

3. What is a (digital) RGB image?
   A 3D matrix of elements each with three channels representing red, green, and blue. There may also be a alpha channel.

Question 2

1. Please briefly describe the process of linear filtering. Because an image is a 3D matrix of intensity values, we can apply operations on it to modify said image. We use something known as a kernel which will be the matrix at that will be applied to a "region". We can apply this kernel on all regions of a image by doing something like convolution or cross-correlation. What we end up with is an image that has been filtered on every pixel, using the same kernel.

2. What are the commonality and difference between (cross) correlation and convolution?
   Commonalities:
   (a) Both are applied to each region in the input image (if padded).

   (b) Same amount of iterations for each operation.

   (c) Both apply filters/kernels.

   Differences:
   (a) Correlation uses the kernel as is, convolution flips it vertically and horizontally.

   (b) Output images are different.

Question 3

Below are a 3x3 grayscale input image (left) and a 3x3 kernel (right). Please manually perform correlation and convolution. Zero padding should be used to make the size of the output image the same as that of the input image.

$$\begin{bmatrix}1 & 0 & 2\\ 2 & 2 & 1\\ 2 & 1 & 0\end{bmatrix} \qquad \begin{bmatrix}2 & 1 & 1\\ 1 & 2 & 0\\ 0 & 0 & 1\end{bmatrix}$$

Solution

Correlation:

$$\begin{bmatrix}0 & 0 & 0 & 0 & 0\\ 0 & 1 & 0 & 2 & 0\\ 0 & 2 & 2 & 1 & 0\\ 0 & 2 & 1 & 0 & 0\\ 0 & 0 & 0 & 0 & 0\end{bmatrix} * \begin{bmatrix}2 & 1 & 1\\ 1 & 2 & 0\\ 0 & 0 & 1\end{bmatrix} = \begin{bmatrix}4 & 2 & 4\\ 6 & 10 & 6\\ 8 & 11 & 6\end{bmatrix}$$

$$0*2+0*1+0*1+0*1+1*2+0*0+0*0+2*0+2*1=4$$
$$0*2+0*1+0*1+1*1+0*2+2*0+2*0+2*0+1*1=2$$
$$0*2+0*1+0*1+0*1+2*2+0*0+2*0+1*0+0*1=4$$
$$0*2+1*1+0*1+0*1+2*2+2*0+0*0+2*0+1*1=6$$
$$1*2+0*1+2*1+2*1+2*2+1*0+2*0+1*0+0*1=10$$
$$0*2+2*1+0*1+2*1+1*2+0*0+1*0+0*0+0*1=6$$
$$0*2+2*1+2*1+0*1+2*2+1*0+0*0+0*0+0*1=8$$
$$2*2+2*1+1*1+2*1+1*2+0*0+0*0+0*0+0*1=11$$
$$2*2+1*1+0*1+1*1+0*2+0*0+0*0+0*0+0*1=6$$

Convolution:

$$\begin{bmatrix}0 & 0 & 0 & 0 & 0\\ 0 & 1 & 0 & 2 & 0\\ 0 & 2 & 2 & 1 & 0\\ 0 & 2 & 1 & 0 & 0\\ 0 & 0 & 0 & 0 & 0\end{bmatrix} * \begin{bmatrix}1 & 0 & 0\\ 0 & 2 & 1\\ 1 & 1 & 2\end{bmatrix} = \begin{bmatrix}8 & 8 & 7\\ 10 & 9 & 3\\ 5 & 4 & 2\end{bmatrix}$$

$$0*1+0*0+0*0+0*0+1*2+0*1+0*1+2*1+2*2=8$$
$$0*1+0*0+0*0+1*0+0*2+2*1+2*1+2*1+1*2=8$$
$$0*1+0*0+0*0+0*0+2*2+0*1+2*1+1*1+0*2=7$$
$$0*1+1*0+0*0+0*0+2*2+2*1+0*1+2*1+1*2=10$$
$$1*1+0*0+2*0+2*0+2*2+1*1+2*1+1*1+0*2=9$$
$$0*1+2*0+0*0+2*0+1*2+0*1+1*1+0*1+0*2=3$$
$$0*1+2*0+2*0+0*0+2*2+1*1+0*1+0*1+0*2=5$$
$$2*1+2*0+1*0+2*0+1*2+0*1+0*1+0*1+0*2=4$$
$$2*1+1*0+0*0+1*0+0*2+0*1+0*1+0*1+0*2=2$$

Question 4

P1: Implement the convolution operator. Directly calling a convolution or filtering function from any library is prohibited. You can use the linear filtering code in our code tutorial as a template (available in Blackboard) or build your own code from scratch. You are encouraged to implement your own Gaussian function. Please use padding to keep the image size unchanged.

1. Use convolution to apply mean, Gaussian (std=1), and sharping filters to lena.png.
2. Try different kernel sizes: 3x3, 5x5, and 7x7.

P2: Implement the median filter (same requirement as P1). To keep the image size unchanged, you may simply ignore the pixels outside the input image when calculating the median value of a patch.

1. Apply both mean and median filters to art.png.
2. Try different kernel sizes: 3x3, 5x5, 7x7, and 9x9

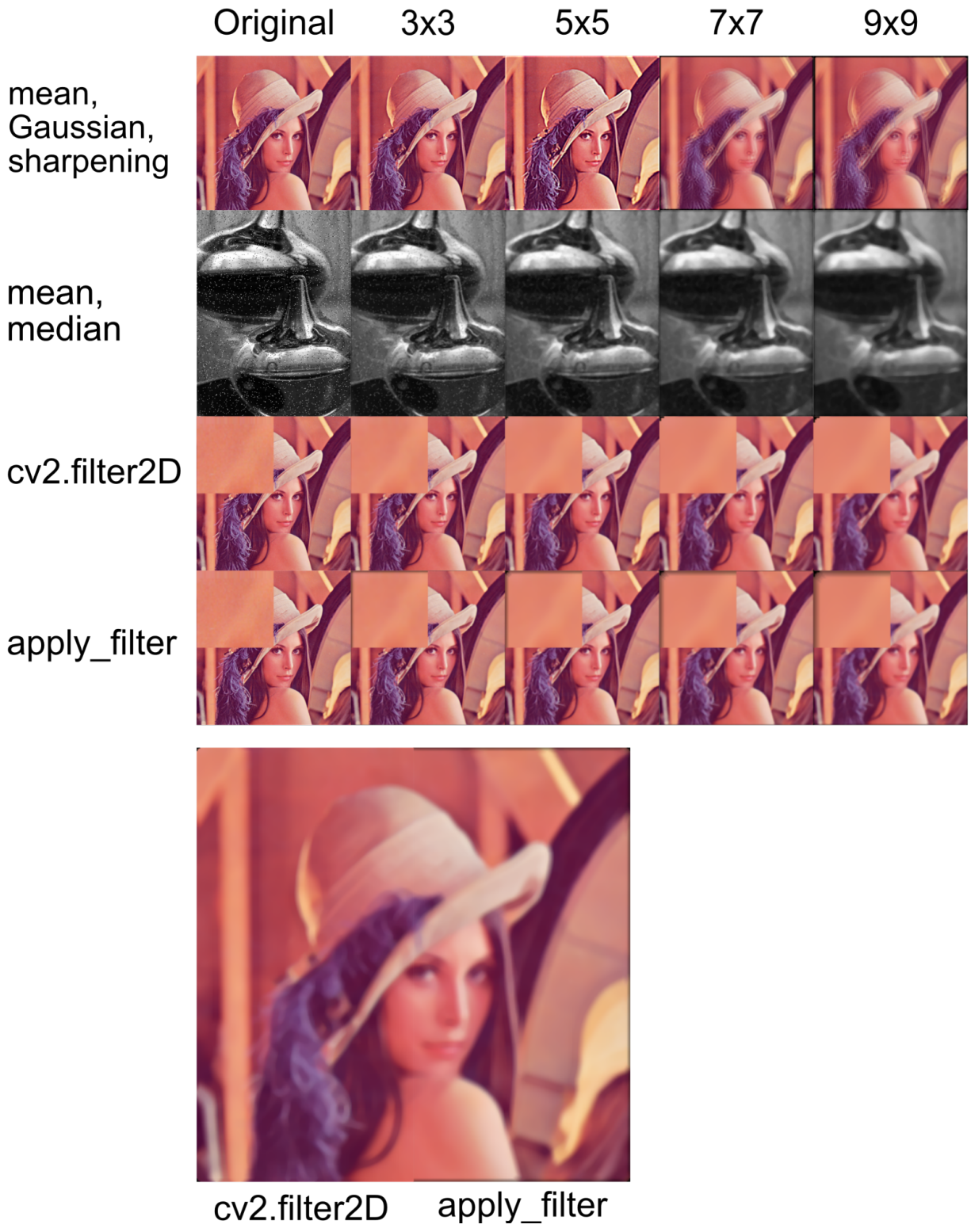
P3: Self-study the filter2D function in OpenCV. Use it to perform Gaussian filtering on lena.png with different kernel sizes (3x3, 5x5, and 7x7). Are the results the same as those obtained by your implementation in P1?

Solution

For **P1** and **P2**, i've included the full code below in this PDF for you to check, you can also find it within the bundled main.py file. I will include an image showcasing this program's output image on the next page.

P3: The images were nearly identical, the only difference was that the edges of my program's output images had black lines which is from the zero padding I did. OpenCV's filter2D function did not have this, most likely it was implemented using something like reflect padding. The reason mine has black lines on the edges is because my convolution function, specifically apply_filter uses those zeroes from the padded kernel in its calculations.

Problem 4 continued on next page...


Original
3x3
5x5
7x7
9x9
mean, Gaussian, sharpening
mean, median
cv2.filter2D
apply_filter
cv2.filter2D
apply_filter

```
#  main.py
''' ------------------------------------------------------------------------------------------
>> Assignment details and provided code are created and owned by Wang Tei.
>> University of Illinois at Chicago - CS 415, Fall 2024
>> ------------------------------------------------------------------------------------------
>> File   :: main.py
>> Course :: CS 415 (42844), FA24
>> Author :: Ryan Magdaleno (rmagd2)
>> System :: Windows 10 w/ Python 3.11.3
- -                                 - -
>> File overview ::
>> This program implements convolution and is able to apply multiple filters onto
>> images. This program makes heavy use of cv2 and numpy to achieve this goal.
>> There are multiple filters to use, including a custom kernel you can modify.
>> You can mix and match filters, along with using cv2's filter2D function instead
>> of this program's implementation (cv2 is way faster). The mean filter can't be
>> used by cv2's filter2D function.
- -                                 - -
>> Usage:
>> ret = convolution("lena.png", 7, ["cv2", "mean", "median"])
>> cv2 must be in the 0th index if you want to use that, you can use other filters
>> in combination after index 0.
>> Meaning: Apply 7x7 mean and median filters onto "lena.png" using cv2.filter2D.
>>
>> ret = convolution("art.png", 3, ["mean", "median"])
>> Meaning: Apply 3x3 mean and median filters onto "art.png" using apply_filter.
------------------------------------------------------------------------------------------ '''
# Module Imports ::
from os import path as opath
from typing import List
from sys import argv
import numpy as np
import cv2 as cv

# Mean kernel is all ones divided by k_size^2 :: - -                                    - -
def mean_kernel(k_size: int) -> np.ndarray:
    return np.ones((k_size, k_size), dtype=np.float32) / (k_size * k_size)
```

```python
# Use np.fromfunction to apply a 2D gaussian function on each (x,y) pos.
# G(x, y)  = 1/( 2(pi))(sigma)^2 ) * exp( -( ( x-u_x )^2 + ( y-u_y )^2 ) / 2(sigma)^2 )
# (sigma)  = std, in this case always 1.0
# u_x, u_y = center of kernel: (k_size - 1) / 2
# I(Ryan), personally used this link to help make this implementation:
# http://www.devanddep.com/tutorial/numpy/how-to-generate-2-d-gaussian-array-using-numpy.html
def gaussian_kernel(k_size: int, std: float=1.0) -> np.ndarray:
    ret = np.fromfunction(
        lambda x, y:
            (1 / (2 * np.pi * std ** 2))
            * np.exp(
                -((x - (k_size - 1) / 2) ** 2 + (y - (k_size - 1) / 2) ** 2)
                / (2 * std ** 2)
            ),
        (k_size, k_size)
    )
    return ret / np.sum(ret) # Normalized
```

```python
# The sharpen kernel emphasizes differences in adjacent pixel values  :: - -           - -
def sharpening_kernel(k_size: int) -> np.ndarray:
    if k_size == 3:
        ret = np.array([
            [ 0, -1,  0],
            [-1,  5, -1],
            [ 0, -1,  0]
        ], dtype=np.float32)
    elif k_size == 5:
        ret = np.array([
            [ 0,  0, -1,  0,  0],
            [ 0, -1, -1, -1,  0],
            [-1, -1, 13, -1, -1],
            [ 0, -1, -1, -1,  0],
            [ 0,  0, -1,  0,  0]
        ], dtype=np.float32)
    elif k_size == 7:
        ret = np.array([
            [ 0,  0,  0, -1,  0,  0,  0],
            [ 0,  0, -1, -1, -1,  0,  0],
            [ 0, -1, -1, -1, -1, -1,  0],
            [-1, -1, -1, 21, -1, -1, -1],
            [ 0, -1, -1, -1, -1, -1,  0],
            [ 0,  0, -1, -1, -1,  0,  0],
            [ 0,  0,  0, -1,  0,  0,  0]
        ], dtype=np.float32)
    elif k_size == 9:
        ret = np.array([
            [ 0,  0,  0,  0, -1,  0,  0,  0,  0],
            [ 0,  0,  0, -1, -1, -1,  0,  0,  0],
            [ 0,  0, -1, -1, -1, -1, -1,  0,  0],
            [ 0, -1, -1, -1, -1, -1, -1, -1,  0],
            [-1, -1, -1, -1,  35, -1, -1, -1, -1],
            [ 0, -1, -1, -1, -1, -1, -1, -1,  0],
            [ 0,  0, -1, -1, -1, -1, -1,  0,  0],
            [ 0,  0,  0, -1, -1, -1,  0,  0,  0],
            [ 0,  0,  0,  0, -1,  0,  0,  0,  0]
        ], dtype=np.float32)
    return ret / np.sum(ret) # Normalized
```

```python
# Custom kernel, make sure to match k_size passed in convolution function :: - -       - -
def custom_kernel(k_size: int) -> np.ndarray:
    ret = np.array([
        [0, 0, 0],
        [0, 0, 0],
        [0, 0, 0]
    ])
    if ret.shape[0] != k_size or ret.shape[1] != k_size:
        exit(" > Custom kernel shape is not k_size * k_size")
    return ret

# Padding utility function :: - -                                                    - -
def pad_image(img: np.ndarray, pad_amount: int) -> np.ndarray:
    return np.pad(
        img,
        (
            (pad_amount, pad_amount),
            (pad_amount, pad_amount),
            (0, 0)
        )
    )

# Median implementation using np.sort and np.ndarray.flatten :: - -                  - -
def median(region: np.ndarray) -> float:
    sorted_region = np.sort(region.flatten())
    n = len(sorted_region)
    if n & 0x1:
        return sorted_region[n // 2]
    else:
        return (sorted_region[n // 2 - 1] + sorted_region[n // 2]) / 2

# Similar to apply_filter, but uses median function here instead :: - -              - -
def median_filter(img: np.ndarray, k_size: int) -> np.ndarray:
    img_height, img_width, img_channels = img.shape
    output = np.zeros_like(img, dtype=np.float32)
    img = pad_image(img, k_size // 2)

    for i in range(img_height):
        for j in range(img_width):
            region = img[i:i+k_size, j:j+k_size]
            for c in range(img_channels):
                output[i, j, c] = median(region[:, :, c])

    return output
```

Problem 4 continued on next page...

```python
# Apply given kernel onto image via convolution :: - -                    - -
def apply_filter(
    img: np.ndarray,
    kernel: np.ndarray,
    k_size: int,
    use_cv: bool) -> np.ndarray:

    if use_cv:
        return cv.filter2D(img, -1, kernel.astype(np.float32))

    img_height, img_width, img_channels = img.shape
    output = np.zeros_like(img, dtype=np.float32)
    img = pad_image(img, k_size // 2)

    for i in range(img_height):
        for j in range(img_width):
            region = img[i:i+k_size, j:j+k_size]
            for c in range(img_channels):
                output[i, j, c] = np.sum(region[:, :, c] * kernel)

    return output
```

```python
# Load kernel / image and apply a filter onto the image via convolution :: - -          - -
def convolution(input_name: str, k_size: int, filters: List[str]) -> str:
    img = cv.imread(input_name, cv.IMREAD_COLOR)
    if img is None:
        return f"Image load error (cv2): \"{input_name}\""

    if k_size < 3 or k_size > 9:
        return f"Kernel size must be 3, 5, 7, or 9: {k_size}"

    if not k_size & 0x1:
        return f"Even kernel size: {k_size}"

    use_cv = False
    if filters and filters[0] == "cv2":
        print(" > Using cv2 filter2D.")
        filters = filters[1:]
        use_cv = True

    if filters is None:
        return f"Add some filter strings"

    kernels = {
        "mean": mean_kernel,
        "custom": custom_kernel,
        "gaussian": gaussian_kernel,
        "sharpening": sharpening_kernel
    }
    for filter in filters:
        if filter == "median":
            img = median_filter(img, k_size)
        elif filter in kernels:
            img = apply_filter(img, np.flip(kernels[filter](k_size)), k_size, use_cv)
        else:
            return f"Invalid kernel filter string passed: {filter}"
        print(f" > {k_size}x{k_size} {filter} filter done.")

    if use_cv:
        filters.insert(0, "cv2")
    ext = opath.splitext(input_name)
    cv.imwrite(f"{ext[0]}_{k_size}x{k_size}_{'-'.join(filters)}{ext[1]}", img)
    return None

# Program entrypoint :: - -                                                          - -
if __name__ == "__main__":
    ret_str = convolution("lena.png", k_size=3, filters=["cv2", "mean", "gaussian"])
    if ret_str:
        print(f" > {ret_str}")
```