# CS 470/548: Assignment 02

## Programming Assignments (95%)

Your program MUST be written in **Python**. Your code file should be named **A02.py.**

The goal of this assignment is to write code to perform **convolution**.

**You may NOT use OpenCV functionality for performing filtering/convolution!** However, you may use the following OpenCV functionality:

- Loading/saving an image
- Window display
- Grayscale conversion
- convertScaleAbs
- flip
- copyMakeBorder
- getGaussianKernel (graduate only)

***If you are not sure whether a library function is permitted, ASK ME!***

A02.py should contain the following functions:

- **def read_kernel_file(filepath)**
    - Open a file for reading and grab the first line
    - The kernel files have the format:
        - (row count) (column count) (value 0,0) (value 0,1) … (value r,c)
    - For example:
        - 3 3 1 2 1 0 0 0 -1 -2 -1
    - …will ultimately become:
        - 1  2  1  
          0  0  0  
          -1 -2 -1
    - Split the line into tokens by spaces
        - Use the string split method
    - Grab the first and second tokens, convert them to ints, and store them as the row and column counts:
        - rowCnt = int(tokens[0])
    - Create a numpy array of zeros of shape (rowCnt, colCnt) to store your kernel values
    - Starting at index = 2, loop through each row and column of the kernel/filter and store the correct token (converted to a float)
    - Return your kernel

- **def apply_filter(image, kernel, alpha=1.0, beta=0.0, convert_uint8=True)**
    - Cast both the image and kernel to "float64"
    - Rotate the kernel 180 degrees so that you are performing convolution:
        - kernel = cv2.flip(kernel, -1)
    - Create a padded image:
        - Get the amount of padding as the height and width of the **kernel** integer-divided by 2
        - Use copyMakeBorder to create a padded image using borderType=cv2.BORDER_CONSTANT and value=0 (zero-padding)
    - Create a FLOATING-POINT numpy array to hold our output image
        - Note this output image should be the size of the original image, NOT the padded one!
    - For each possible center pixel (row,col) (you can use the dimensions of the original image or the output image, but NOT the padded image!!!!)
        - You can do correlation of the neighborhood using a neat numpy trick:
            - Grab the subimage from the PADDED image
              [ row : (row + kernel.shape[0]), col : (col + kernel.shape[1]) ]
            - Multiply the subimage by the kernel:
                - filtervals = subImage * kernel
            - Get the sum of these values:
                - value = np.sum(filtervals)
            - Write this to your output image
                - output[row,col] = value
    - If convert_uint8 is True:
        - Use convertScaleAbs on the output image using the provided alpha and beta
    - Return output image
- ***CS 548 ONLY:***
    - **def check_zero_cross(v1, v2, thresh)**
        - If v1 * v2 is greater than or equal to zero, return False immediately
        - If the absolute value (you can use m.fabs() from the math library) of the difference between v1 and v2 is LESS THAN thresh, return False immediately
        - Otherwise, return True
    - **def get_marr_hildreth_edges(image, scale, thresh):**
        - Use getGaussianKernel() to get the 1D Gaussian kernel:
            - gauss_1D = cv2.getGaussianKernel(scale, -1)
            - Note: the shape will be (1, scale), so this gives you a column filter
        - Use np.transpose to get the row filter:
            - gauss_1D_T = np.transpose(gauss_1D)
        - You now have the separable Gaussian filters

    - Using your apply_filter() function, blur the image with one Gaussian and then with the other, **MAKING SURE THAT convert_uint8 is False in BOTH cases**
    - Create a quick Laplacian filter:
        - lap = np.array([[0,1,0],[1,-4,1],[0,1,0]], dtype="float")
    - Use apply_filter() to get the Laplacian image of the blurred image, **AGAIN, MAKING SURE THAT convert_uint8 is False**
    - Create a destination image (filled with zeros) for your edges **using dtype="uint8"**
    - You will now look for zero-crossing points in the Laplacian image; HOWEVER, your row and column indices should NOT go to the last row and column, respectively
    - For each row and column (EXCEPT THE LAST ROW AND/OR COLUMN):
        - Get the upper-left, upper-right, lower-left, and lower-right pixel values at: [r,c], [r,c+1], [r+1,c], [r+1,c+1]
        - Using your check_zero_cross() function, if ANY of the following return True:
            - Upper-Left vs. Upper-Right (horizontal)
            - Upper-Left vs. Lower-Left (vertical)
            - Upper-Left vs. Lower-Right (diagonal)
            - Lower-Left vs. Upper-Right (diagonal)
        - …then set edges[r,c] = 255
    - Return your edge image

In addition to the functions, you will also copy and paste in the following to allow you to run your program with Gradio (be sure to "import gradio as gr" at the top):

***Everyone:***

```
def filtering_callback(input_img, filter_file, alpha_val, beta_val):
    input_img = cv2.cvtColor(input_img, cv2.COLOR_BGR2GRAY)
    kernel = read_kernel_file(filter_file.name)
    output_img = apply_filter(input_img, kernel, alpha_val, beta_val)
    return output_img


# Later, at the bottom
if __name__ == "__main__":
    main()
```

***CS 470 ONLY:***

```
def main():
    demo = gr.Interface(fn=filtering_callback,
                        inputs=["image",
                                "file",
                                gr.Number(value=0.125),
                                gr.Number(value=127)],
                        outputs=["image"])
    demo.launch()
```

***CS 548 ONLY:***

```
def edge_callback(input_img, scale_val, thresh_val):
    input_img = cv2.cvtColor(input_img, cv2.COLOR_BGR2GRAY)
    output_img = get_marr_hildreth_edges(input_img, scale_val, thresh_val)
    return output_img

def main():
    with gr.Blocks() as demo:
        with gr.Row():
            with gr.Column():
                image_data = gr.Image(label="Input Image")

            with gr.Tab("Filtering"):
                with gr.Row():
                    with gr.Column():
                        filter_filename = gr.File(label="Filter File")
                        alpha_number = gr.Number(label="Alpha", value=0.125)
                        beta_number = gr.Number(label="Beta", value=127)
                        filter_button = gr.Button("Perform Filtering")
                    image_output = gr.Image(label="Filtered Image")

            with gr.Tab("Marr-Hildreth"):
                with gr.Row():
                    with gr.Column():
                        scale_number = gr.Number(label="Scale", value=7,
precision=0)
```

```
                        thresh_number = gr.Number(label="Threshold", value=3,
precision=0)
                        edge_button = gr.Button("Get Marr-Hildreth Edges")
                    mh_output = gr.Image(label="Edge Image")

        filter_button.click(filtering_callback,
                            inputs=[image_data,
                                    filter_filename,
                                    alpha_number,
                                    beta_number],
                            outputs=image_output)

        edge_button.click(edge_callback,
                            inputs=[image_data,
                                    scale_number,
                                    thresh_number],
                            outputs=mh_output)

    demo.launch()
```

If you run your program, you should be able to open a web browser to http://127.0.0.1:7860 and see a nice GUI version.

## Testing Screenshot (5%)

I have provided several new files for testing:

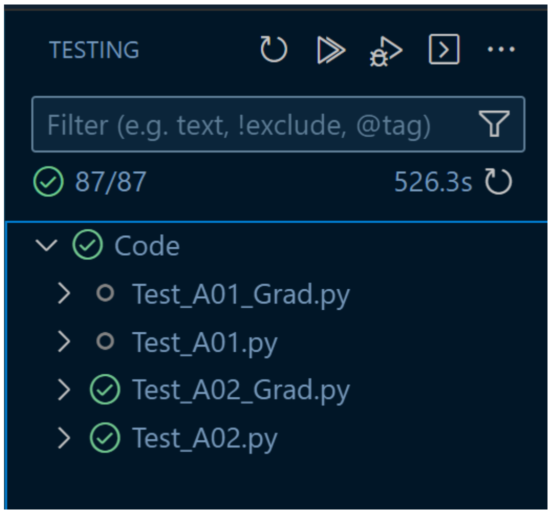
- Test_A02.py – the test program for CS 470
- Test_A02_Grad.py – the test program for CS 548
- assign02/
  - images/
    - Input images for testing
  - filters/
    - Filter files
  - ground/
    - Ground-truth images organized by filters and approach

Copy these files/folders into the MAIN project folder; your python program (A02.py) should also reside there.

You can either run the testing programs directly OR you can use the testing section of Visual Code.

**You MUST run the tests and send a screenshot of the test results!** Even if your program(s) do not pass all the tests, you MUST send this screenshot!

**Graduate students (CS 548) MUST run Test_A02_Grad!** (Please note also that you do need the Test_A02 file as well, since Test_A02_Grad relies on it.)



87/87 526.3s

Code
Test_A01_Grad.py
Test_A01.py
Test_A02_Grad.py
Test_A02.py

You may have to do "Command Palette" → "Python: Configure Tests" → unitttest → root directory → "test_*.py", and then modify .vscode/settings.json to use "Test_*.py

This screenshot should show clearly:

- The final result of the test run on the command line ("OK" for all passing, "FAILED (failures=N)" for some or all failing).
- OR
- The testing view in Visual Code (see image on right)

Because there are SIGNIFICANTLY more tests than before, I'm really looking for the checkboxes (or failed overall markers) for Test_A02 or Test_A02_Grad. I don't need to see ALL of the tests.

## Grading

Your OVERALL assignment grade is weighted as follows:

- 5% - Testing results screenshot
- 95% - Programming assignments

I reserve the right to take points off for not meeting the specifications in this assignment description.

In general, these are things that will be penalized:

- **Code that is not syntactically correct (up to 60 points off!)**
- Sloppy or poor coding style
- Bad coding design principles
- Code that crashes, does not run, or takes a VERY long time to complete
- Using code from ANY source other than the course materials
- Collaboration on code of ANY kind; this is an INDIVIDUAL PROJECT
- Sharing code with other people in this class or using code from this or any other related class
- Output that is incorrect
- Algorithms/implementations that are incorrect
- Submitting improper files
- Failing to submit ALL required files