Homework 1 Written Questions

Template Instructions

This document is a template with specific answer regions and a fixed number of pages. Given large class sizes and limited TA time, the template helps the course staff to grade efficiently and still focus on the content of your submissions. Please help us in this task:

- Make this document anonymous.
- Questions are in the orange boxes. Provide answers in the green boxes.
- Use the footer to check for correct page alignment.
- **Do NOT remove the answer box.**
- **Do NOT change the size of the answer box.**
- **Extra pages are not permitted unless otherwise specified.**
- **Template edits or page misalignment will lead to a 10 point deduction.**

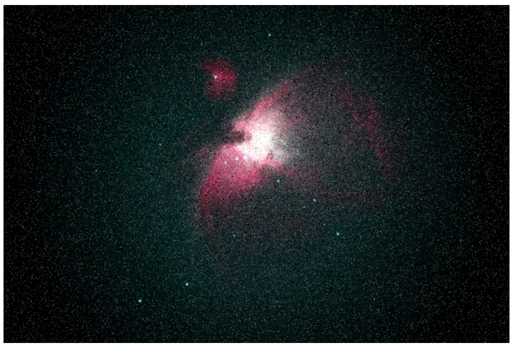
Gradescope Submission

- Compile this document to a PDF and submit it to Gradescope.
- Pages will be automatically assigned to the right questions on Gradescope.

This Homework

- 5 questions **[15 + 11 + 12 + 7 + 7 = 49]**.
- Include code, images, and equations where appropriate.

Q1: [15 points] We have been given special permission to use the telescope on the roof of Barus and Holley. Unfortunately, our fantastic image of the Orion nebula has noise caused by the imaging sensor:



One way to deal with this noise is with image convolution. Convolution is a type of image filtering that is a fundamental image processing tool.

(a) *Explicitly describe* the input, transformation, and output components of 2D discrete convolution. Please be precise; define variables as need.

(i) **[2 points]** Input **[2–4 sentences]**

(ii) **[2 points]** Transformation (how is the image transformed?) **[2–4 sentences]**

(iii) **[2 points]** Output **[2-4 sentences]**

(b) **[3 points]**

Briefly describe three different kernels we may use with convolution along with an example application for each. **[5–6 sentences]**

(c) **(Bonus) [1 point]**

What kind of filter might we use to process our image of the Orion Nebula, and why? **[2 - 3 sentences]**

(d) **[5 points]**

Why is image convolution important in computer vision? Which applications can it benefit and where can it be misused? **[5-8 sentences]**

Q2: **[11 points]** Now that we've successfully used convolution to de-noise our image of the Orion nebula, you decide to explore the filtering technique more closely.

Specifically, you know two filtering operations exist: correlation and convolution. Both techniques extract (or delete) information from images.

(a) **[3 points]**

In what ways can extraction of information from images be helpful or beneficial? What about deletion of information? Discuss this both in the context of your image of the Orion nebula and other real-life examples. **[5–6 sentences]**

(b) **[2 points]**

Comment on the difference between convolution and correlation, including their properties. **[5–6 sentences]**

(c) **[1 + 1 points]**

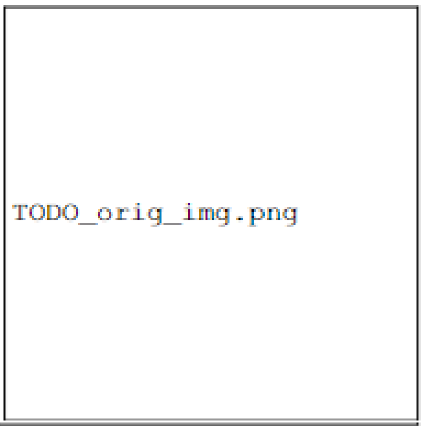
You attempt to use both correlation and convolution on the mean filter over the orion nebula. Do you expect different output images? Generally, when do correlation and convolution produce identical results? **[4–5 sentences]**

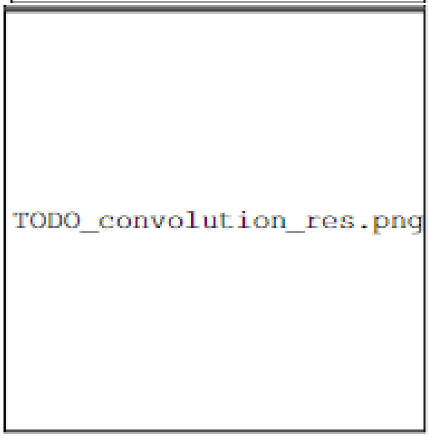
TODO: Your answer for (c) here

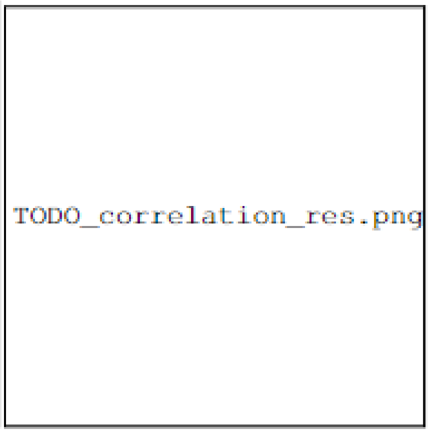
(d) **[2 + 2 points]** You decide to solidify your understanding of the distinction between correlation and convolution by taking another image.

For this, come up with a use case where the output of correlation and convolution differ.
Write some code that takes an image and produces two distinct images, one from convolution and one from correlation on some kernel of your choice. Specify your kernel, and provide the input image and two output results. Then, use your understanding of convolution and correlation to explain the outputs. **[4–5 sentences]**

Please use scipy.ndimage.convolve and scipy.ndimage.correlate to experiment!



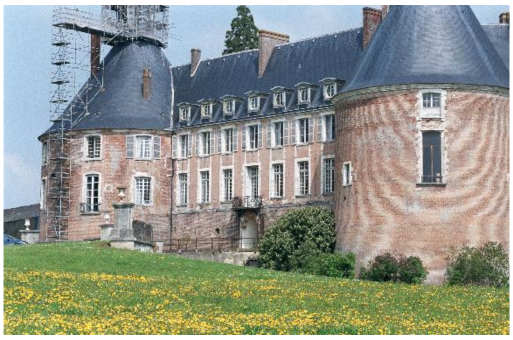




TODO: Your answer for (d) here

Q3: **[12 points]** You're a restoration architect who's in charge of determining when heritage site structures need a bit of upkeeping due to weather, erosion or other factors (natural and tourism-caused). This week, you've been assigned to visit a castle on a beautiful seafront to see if any restoration is in order.

After a heavy analysis, you decide the castle is in decent shape. As such, you need to convince your superior, so you snap a picture of the castle with your camera:



(a) **[2 points]** Oh no! Looks like there are some weird artifacts on the image, almost as though the walls are bleeding color?

This phenomena is called aliasing, but why has it happened? **[5–6 sentences]**

TODO: Your answer for (a) here

(b) **[3 points]** You decide to fix this by passing your image through an anti-aliaser online. Amongst a host of complicated things, the software reduces the 'choppiness' of the image using low-pass filter convolution (some related high-pass filters also exist):

Let's make sure we fully understand how this works though. Can you identify which of the following filters is high pass or low pass?

Note: To fill in boxes, replace '\square' with '\blacksquare' for your answer.

(i) $\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$

TODO: Select the appropriate answer.

☐ High pass
☐ Low pass
☐ Neither

(ii) $\begin{bmatrix} \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{bmatrix}$

TODO: Select the appropriate answer.

☐ High pass
☐ Low pass
☐ Neither

(iii) $\begin{bmatrix} -\frac{1}{9} & -\frac{1}{9} & -\frac{1}{9} \\ -\frac{1}{9} & \frac{8}{9} & -\frac{1}{9} \\ -\frac{1}{9} & -\frac{1}{9} & -\frac{1}{9} \end{bmatrix}$

TODO: Select the appropriate answer.

☐ High pass
☐ Low pass
☐ Neither

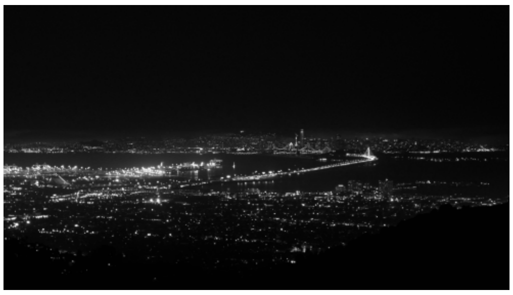
(c) **[3 points]**

You verify that the image looks as expected after you've passed it through the anti-aliaser. Do you have an obligation to tell your superior about the way you've modified the image? If yes, do you need to also tell your superior about image modifications like brightness, zoom, contrast, etc? If no, what amount of image modification is acceptable before it becomes misinformation or misleading? **[6-7 sentences]**

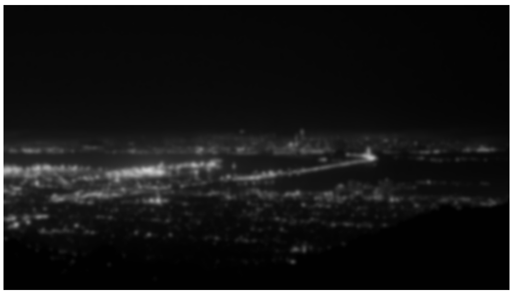
TODO: Your answer for (c) here

(d) **[2 points]** You think you've gotten a full understanding of how the filters classify, but decide to test if you can recognize which filter has been used to get a target output image.

Given the input image below, identify the filter that has been applied.

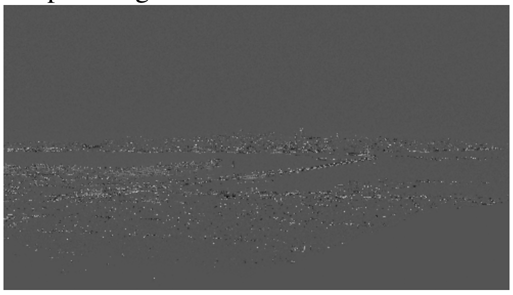


(i) Output image 1:



TODO: Select the appropriate answer.
☐ High pass
☐ Low pass

(ii) Output image 2:



TODO: Select the appropriate answer.
☐ High pass
☐ Low pass

(e) **[2 points]**

Which of the following statements are true? (Check all that apply).

TODO: Select all that apply.

- ☐ High pass filter kernels will always contain at least one negative number
- ☐ A Gaussian filter is an example of a low pass filter
- ☐ A high pass filter is the basis for most smoothing methods
- ☐ In a high pass filter, the center of the kernel must have the highest value

Q4: [7 points] It is very important to understand how convolution/correlation scale with input size. A good measure of scale is measuring how the computation times of such filtering operations varies with filter sizes.

To make an accurate assessment on this computational scaling, we will be using filters with dimensions $n \times n$ where n is an odd number and $n \in [3, 15]$ (i.e. 3×3, 5×5, 7×7, etc.), and images of sizes between 0.25 to 8 megapixels.

Note A megapixel is 1,048,576 (2^{20}) pixels (1024×1024), or sometimes also 1,000,000 pixels (especially if you manufacture cameras). You can use either of the definitions for this question. Megapixels is often shortened to MP or MPix.

(a) **[3 points]**

Complete the stencil code below to create a graph with one trace per filter size, where the x-axis represents the correlation/convolution between an image size, and y-axis represents the time to convolve/correlate that image. The stencil code imports the libraries you will need, but to understand how to use them, please look at the documentation.

- convolve/correlate - *scipy.ndimage.convolve* or *scipy.ndimage.correlate*
- rescale - *skimage.transform.rescale*
- resize - *skimage.transform.resize*
- rescale vs resize – an example here

Image: RISDance.jpg (in the .tex directory).

```
import time
import matplotlib.pyplot as plt
from skimage import io, img_as_float32
# use to rescale+resize image
from skimage.transform import rescale, resize
# use to convolve/correlate image
from scipy.ndimage import correlate

# This reads in image and converts it to a
# floating point format
# 1) TODO - replace PATH with the actual path to the
#    downloaded RISDance.jpg image linked above
image = img_as_float32(io.imread('PATH'))

# 2) TODO - change the image size so it starts
#    at 8MPix (calculated as height x width)
#    use one of the imported libraries. Note that the
#    number of channels should be kept the same
#    after this operation
original_image =
# 3) TODO - iterate through odd numbers from 3 to 15
#    (inclusive!!) these will represent your filter sizes
#    (3x3,5x5,7x7, etc.), for each filter size you will...
for kernel_size in range():

    # because for each loop you are resizing your image,
    # you want to start each loop w/ original image size
    shrinking_image = original_image

    # these lists will hold the values you plot
    image_sizes = [] #x axis
    times = [] #y axis

    # while image size is bigger than .25MPx
    while(shrinking_image.size > 250000):

      # 4) TODO - create your kernel. Your kernel can
      #    hold any values, as the kernel values
      #    shouldn't affect computation time.

      #    Avoid using np.zeros to create your kernal,
      #    as this messes up the intended output graph.

      #    The size of the kernel must be kernel_size
      #    x kernel_size
        kernel =

        # 5) TODO - reduce your image size. You can choose
        # by what increments to reduce your image.
        shrinking_image =

        # gets the current time (in seconds)
        start = time.time()

        # 6) TODO - use one of the imported libraries to do
        # your correlation/convolution on the image.
        # You can choose which operation to perform.

        # gets the current time (in seconds)
        end = time.time()

        #7) TODO - figure out what values to append, and
        #   append them here
        image_sizes.append()
        times.append()

    # each filter size will be plotted as a separate line,
    # in a multi-line 2-dimensional graph
    plt.plot(image_sizes, times, label=str(kernel.size))

# plot
plt.xlabel('image size (pixels)')
plt.ylabel('operation time (seconds)')
plt.legend(title="filter sizes (pixels)")
plt.show()
```

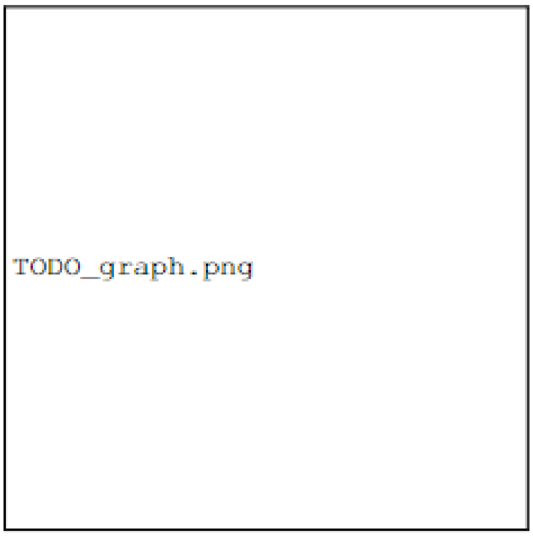
Only A4 (a) should be on this page

```
################################################
# YOU MAY USE THIS ADDITIONAL PAGE

# WARNING: IF YOU DON'T END UP USING THIS PAGE
# KEEP THESE COMMENTS TO MAINTAIN PAGE ALIGNMENT
################################################
```

(b) **[2 points]**

Present your graph with a brief description of what your graph demonstrates. **[2 - 3 sentences]**



TODO: Your answer for (b) here

(c) **[2 points]**

Do the results match your expectation given the number of multiply and add operations in convolution? **[5-6 sentences]**

TODO: Your answer for (c) here

Q5: **[7 points]** The `numpy` library is extremely important when working with input data (vectors or matrices) and their linear algebra manipulations. In Computer Vision, your data will often come in the form of matrices of images (pixel representations).

To familiarize yourself with the library, read through the following scenarios and complete the exercises. Write *one* `numpy` function to complete each of the following tasks.

Note that numpy is usually imported as

```
import numpy as np
```

at the top of the code file. You can then call numpy functions with

```
np.function_name(<arguments>)
```

You are encouraged to test out your answers by creating your own python program, importing numpy and calling and printing the results of your solutions! Some numpy functions you might find useful are np.squeeze, np.expand_dims, np.clip, np.pad, and np.zeros.

You may also use operators like `[]` and `:`, but remember to use only *one* function/operator shorthand.

(a) **[1 point]** You're attempting to create a black image base, `img`. All values in this matrix are 0.

Create `img` where `np.shape(img) == (320,640)`. **[1 sentence]**

```
# TODO: Your expression here
```

(b) **[1 point]** You've been working on a Computer Vision pipeline that de-noises your image. Unfortunately, it seems to mess up the dimensions, and outputs an image-array, `img_out`, where `np.shape(img_out) == (1, 1, 320, 640)`.

Convert `img_out` to a new 2D image-array, `img_fixed`, where `np.shape(img_fixed) == (320, 640)`. In other words, remove all the 1-sized dimensions. **[1 sentence]**

```
# TODO: Your expression here
```

(c) **[1 point]** Usually, image-arrays are represented with three dimensions. The first dimension represents the number of channels, and can be used to identify if an image is RGB or grayscale.

Say you have a grayscale image-array `img` where `np.shape(img) == (320, 640)`. Convert this to a new image-array, `img_expanded`, which appropriately captures the number of channels, where `np.shape(img_expanded) == (1, 320, 640)`. In other words, add a dimension to `img`. **[1 sentence]**

```
# TODO: Your expression here
```

(d) **[1 point]** When you learn about Convolutional Neural Networks later in the course, you'll find yourself needing to normalize input images such that each channel has intensities ranging from -1 to 1 instead of 0 to 255.

Assume you have a linearly normalized grayscale 2D image-array, `img` (the channel dimension of 1 has already been removed), and want to remove outlier intensities and artefacts (such as glare). Clip `img` so all its values lie within the range [-0.5, 0.5]. **[1 sentence]**

```
# TODO: Your expression here
```

(e) **[1 point]** Let's say you're trying to code a red-green-filter function. This will take in an RGB image-array, `img`, (this means the image has 3 channels), and filters all but the blue channel. Note that `np.shape(img) == (320, 640, 3)`. **[1 sentence]**

Retrieve the blue channel of `img` while preserving all of `img`'s dimensions and intensity values.

```
# TODO: Your expression here
```

(f) **[1 point]** Let's say you're trying to code a green-filter function. This will take in an RGB image-array, `img`, and filters out the green channel. Note that `np.shape(img) == (320, 640, 3)`.

Retrieve the red and blue channels of `img` while preserving all of `img`'s dimensions and intensity values. **[1 sentence]**

```
# TODO: Your expression here
```

(g) **[1 point]** Same-Padding is a useful tool to ensure that the dimensions of the output image from a convolution operation matches the dimensions of the input image.

Given an RGB image-array, `img`, pad it with two columns of zeros on the left and right edges of the image, and three rows of zeros on the top and bottom edges of the image. Don't add zeros to the color channel dimension (the front and back faces of the image-array). **[1 sentence]**

```
# TODO: Your expression here
```

Only A5 (f) - (g) should be on this page

Feedback? (Optional)

Please help us make the course better. If you have any feedback for this assignment, we'd love to hear it!