Assignment_1_Notebook

March 31, 2025

1 Computer Vision 2025 Assignment 1: Image filtering.

In this assignment, you will research, implement and test some image filtering operations. Image filtering by convolution is a fundamental step in many computer vision tasks and you will find it useful to have a firm grasp of how it works. For example, later in the course we will come across Convolutional Neural Networks (CNNs) which are built from convolutional image filters.

The main aims of the assignment are:

- to understand the basics of how images are stored and processed in memory;
- to gain exposure to several common image filters, and understand how they work;
- to get practical experience implementing convolutional image filters;
- to test your intuition about image filtering by running some experiments;
- to report your results in a clear and concise manner.

This assignment relates to the following ACS CBOK areas: abstraction, design, hardware and software, data and information, HCI and programming.

1.1 General instructions

Follow the instructions in this Python notebook and the accompanying file *a1code.py* to answer each question. It's your responsibility to make sure your answer to each question is clearly labelled and easy to understand. Note that most questions require some combination of Python code, graphical output, and text analysing or describing your results. Although we will check your code as needed, marks will be assigned based on the quality of your write up rather than for code correctness! This is not a programming test - we are more interested in your understanding of the topic.

Only a small amount of code is required to answer each question. We will make extensive use of the Python libraries

- numpy for mathematical functions
- skimage for image loading and processing
- matplotlib for displaying graphical results
- jupyter for Jupyter Notebooks

You should get familiar with the documentation for these libraries so that you can use them effectively.

2 The Questions

To get started, below is some setup code to import the libraries we need. You should not need to edit it.

[ ]:
```
# Numpy is the main package for scientific computing with Python.
import numpy as np

#from skimage import io

# Imports all the methods we define in the file a1code.py
from a1code import *

# Matplotlib is a useful plotting library for python
import matplotlib.pyplot as plt
# This code is to make matplotlib figures appear inline in the
# notebook rather than in a new window.
%matplotlib inline
plt.rcParams['figure.figsize'] = (10.0, 8.0) # set default size of plots
plt.rcParams['image.interpolation'] = 'nearest'
plt.rcParams['image.cmap'] = 'gray'

# Some more magic so that the notebook will reload external python modules;
# see http://stackoverflow.com/questions/1907993/
 ↪autoreload-of-modules-in-ipython
%load_ext autoreload
%autoreload 2
%reload_ext autoreload
```

2.1 Question 0: Numpy warm up! (Not Assesed. This part is for you to understand the basic of numpy)

Before starting the assignment, make sure you have a working Python 3 installation, with up to date versions of the libraries mentioned above. If this is all new to you, I'd suggest downloading an all in one Python installation such as Anaconda. Alternatively you can use a Python package manager such as pip or conda, to get the libraries you need. If you're struggling with this please ask a question on the MyUni discussion forum.

For this assignment, you need some familiarity with numpy syntax. The numpy QuickStart should be enough to get you started:

https://numpy.org/doc/stable/user/quickstart.html

Here are a few warm up exercises to make sure you understand the basics. Answer them in the space below. Be sure to print the output of each question so we can see it!

1. Create a 1D numpy array Z with 12 elements. Fill with values 1 to 12.
2. Reshape Z into a 2D numpy array A with 3 rows and 4 columns.
3. Reshape Z into a 2D numpy array B with 4 rows and 3 columns.
4. Calculate the *matrix* product of A and B.

5. Calculate the *element wise* product of A and B^T (B transpose).

[ ]:
```
import numpy as np

Z = np.arange(1,13)
print(Z)

print()

A = Z.reshape(3,4)
print(A)

print()

B = Z.reshape(4,3)
print(B)

print()

print(A @ B)

print()

print(A * np.transpose(B))
```

You need to be comfortable with numpy arrays because that is how we store images. Let's do that next!

2.2 Question 1: Loading and displaying an image (10%)

Below is a function to display an image using the pyplot module in matplotlib. Implement the `load()` and `print_stats()` functions in a1code.py so that the following code loads the mandrill image, displays it and prints its height, width and channel.

[ ]:
```
def display(img, caption=''):
    # Show image using pyplot
    plt.figure()
    plt.imshow(img)
    plt.title(caption)
    plt.axis('off')
    plt.show()
```

[ ]:
```
image1 = load('images/whipbird.jpg')

display(image1, 'whipbird')

print_stats(image1)
```

Return to this question after reading through the rest of the assignment. Find **at least 2 more images** to use as test cases in this assignment for all the following questions and display them below. Use your print_stats() function to display their height, width and number of channels. Explain *why* you have chosen each image.

```
[ ]: ### Your code to load and display your images here
image2 = load("images/child.png")
display(image2, "child")
print_stats(image2)

image3 = load("images/pengbrew.png")
display(image3, "penguin")
print_stats(image3)

image4 = load("images/galaxy.jpg")
display(image4, "galaxy")
print_stats(image4)

image5 = load("images/house.png")
display(image5, "house")
print_stats(image5)

image6 = load("images/grainy.jpg")
display(image6, "grainy")
print_stats(image6)

image7 = load("images/mandrill.jpg")
display(image7, "mandrill")
print_stats(image7)
```

image1 (child.png) - The first image I selected had transparent sections, and therefore the typical 3 channels of colour would not suffice for this image. I was curious as to whether or not the number of channels would reflect the transparency in the image. The fourth channel (the alpha channel) is used to reflect transparency, but in this case the transparent sections are converted to black and thus the number of channels is 3. I would need to modify the image read function to account for this fourth channel but I believe it is out of the scope of this assignment to do so.

image2 (pengbrew.png) - The second image I selected was a black-and-white image of a penguin, I selected this image such that I could ensure the number of channels present within the image would equate to a value of 1. Selecting this image is useful as it ensures that the shape function for an image is correctly identifying the number of channels present within an image, and can distinguish between other variation of channel counts.

image3 (galaxy.jpg) - The third image I selected was a large image of the galaxy, I selected this image to ensure the print_stats function could appropriately identify the size of images regardless of their size. In selecting this image and achieving the correct result I can validate the correctness of the width and height output of the print_stats function.

image4 (house.png) - The fourth image I selected was a cartoon picture depicting a house with a white background, I selected this image such that I could test the edge detection in the convolution section later on within this assignment. The image possesses strong edges with large variations in colour intensity that should be very simple to identify with an edge detection method.

image5 (grainy.jpg) - The fifth image I selected was a very small grainy image of a lakeside walkway, I selected this image such that I could test the noise-reduction filters like the gaussian filter. This image is incredibly pixelated and therefore should produce a significantly clearer result when exposed to gaussian blurring.

2.3 Question 2: Image processing (30%)

Now that you have an image stored as a numpy array, let's try some operations on it.

1. Implement the `crop()` function in a1code.py. Use array slicing to crop the image.
2. Implement the `resize()` function in a1code.py.
3. Implement the `change_contrast()` function in a1code.py.
4. Implement the `greyscale()` function in a1code.py.
5. Implement the `binary()` function in a1code.py.

What do you observe when you change the threshold of the binary function?

Apply all these functions with different parameters on your own test images.

When I change the threshold of the binary function, I am effectively adjusting the level at which a pixel is considered background or foreground. In a brighter image, there are generally more pixels that will fall above the threshold mark because they have naturally higher intensity values, which convert them to white pixels. Comparatively, in darker images it is far more likely for pixels to fall below the threshold mark which because they have comparatively lower intensity values, which convert them to black pixels.

The level I set the threshold to also impacts the output image, if I set the threshold high I am far more likely to recieve an image containing more black pixels as opposed to white pixels. On the other hand, if I set the threshold too low I am far more likely to receive an image containing more white pixels as opposed to black pixels.

[ ]:
```python
# This should crop the bird from the  image; you will need to adjust the␣
↪parameters for the correct crop size and location
crop_img = crop(image1, 120, 250, 210, 170)
display(crop_img)
print_stats(crop_img)

resize_img = resize(crop_img, 500, 600)
display(resize_img)
print_stats(resize_img)

contrast_img = change_contrast(image1, 0.5)
display(contrast_img)
print_stats(contrast_img)

contrast_img = change_contrast(image1, 1.5)
display(contrast_img)
```

```python
print_stats(contrast_img)

grey_img = greyscale(image1)
display(grey_img)
print_stats(grey_img)

binary_img = binary(grey_img, 0.3)
display(binary_img)
print_stats(binary_img)

binary_img = binary(grey_img, 0.7)
display(binary_img)
print_stats(binary_img)

# Add your own tests here...
crop_img2 = crop(image5, 200, 200, 300, 300)
display(crop_img2)
print_stats(crop_img2)

resize_img2 = resize(crop_img2, 400, 500)
display(resize_img2)
print_stats(resize_img2)

contrast_img2 = change_contrast(image2, 0.2)
display(contrast_img2)
print_stats(contrast_img2)

contrast_img3 = change_contrast(image3, 2.0)
display(contrast_img3)
print_stats(contrast_img3)

grey_img2 = greyscale(image4)
display(grey_img2)
print_stats(grey_img2)

binary_img2 = binary(grey_img2, 0.2)
display(binary_img2)
print_stats(binary_img2)

binary_img3 = binary(grey_img2, 0.8)
display(binary_img3)
print_stats(binary_img3)

binary_img4 = binary(image3, 0.5)
display(binary_img4)
print_stats(binary_img4)
```

```
resize_img3 = resize(image4, 300, 400)
display(resize_img3)
print_stats(resize_img3)

binary_img5 = binary(grey_img, 0.5)
display(binary_img5)
print_stats(binary_img5)
```

2.4 Question 3: Convolution (30%)

2.4.1 3.1(a) 2D convolution

Using the definition of 2D convolution from week 1, implement the convolution operation in the function `conv2D()` in a1code.py.

```
[ ]: test_conv2D()

kernel = np.array(
    [
        [1,0,0],
        [0,1,0],
        [0,0,0]
    ])

display(image5)

# Normal size image
image_filtered_1 = conv(image5, gauss2D(3,1))
image_filtered_2 = conv(image5, gauss2D(3,5))
image_filtered_3 = conv(image5, gauss2D(9,9))
image_filtered_4 = conv(image5, gauss2D(9,1))

display(image_filtered_1)
display(image_filtered_2)
display(image_filtered_3)
display(image_filtered_4)

# Compute the difference
diff_1 = np.abs(image5 - image_filtered_1)
diff_2 = np.abs(image5 - image_filtered_2)
diff_3 = np.abs(image5 - image_filtered_3)
diff_4 = np.abs(image5 - image_filtered_4)

display(diff_1)
display(diff_2)
display(diff_3)
display(diff_4)
```

```
image5Small = resize(image5, 200, 200)

# Normal size image
image_filtered_1 = conv(image5Small, gauss2D(3,1))
image_filtered_2 = conv(image5Small, gauss2D(3,5))
image_filtered_3 = conv(image5Small, gauss2D(9,9))
image_filtered_4 = conv(image5Small, gauss2D(9,1))

display(image_filtered_1)
display(image_filtered_2)
display(image_filtered_3)
display(image_filtered_4)

# Compute the difference
diff_1 = np.abs(image5Small - image_filtered_1)
diff_2 = np.abs(image5Small - image_filtered_2)
diff_3 = np.abs(image5Small - image_filtered_3)
diff_4 = np.abs(image5Small - image_filtered_4)

display(diff_1)
display(diff_2)
display(diff_3)
display(diff_4)
```

2.4.2 3.1(b) RGB convolution

In the function `conv` in a1code.py, extend your function `conv2D` to work on RGB images, by applying the 2D convolution to each channel independently.

2.4.3 3.2 Gaussian filter convolution

Use the `gauss2D` function provided in a1code.py to create a Gaussian kernel, and apply it to your images with convolution. You will obtain marks for trying different tests and analysing the results, for example:

- try varying the image size, and the size and variance of the filter
- subtract the filtered image from the original - this gives you an idea of what information is lost when filtering

What do you observe and why?

Varying the image size, and the size and variance of the filter had some interesting effects on the resulting image. The following could be identified:

Effects of image size: - Enlarging the image while keeping the Gaussian filter size the same resulted in less noticeable blurring, as the filter covered a smaller proportion of the image. - Conversely, reducing the image size before applying the Gaussian filter made the blurring effect more noticeable, as the filter influences more of the image at once.

Therefore, larger images require proportionally larger Gaussian filters to achieve the same level of blurring.

Effects of filter size: - A smaller Gaussian filter provided slight smoothing and blurring when convolved with an image. - On the other hand, a larger Gaussian filter provided greater smoothing and blurring, removing significantly more finer detail within an image.

Therefore, a larger Gaussian filter provided greater blurring and greater loss of finer detail.

Effects of variance of the filter: - A smaller variance produced less blurring as it kept the Gaussian distribution more concentrated. - On the other hand, a larger variance spread the blurring effect over a larger area, increasing blurring.

Therefore, a greater variance produced greater smoothing and loss of details.

2.4.4 3.3 Sobel filters

Define a horizontal and vertical Sobel edge filter kernel and test them on your images. You will obtain marks for testing them and displaying results in interesting ways, for example:

- apply them to an image at different scales
- considering how to display positive and negative gradients
- apply different combinations of horizontal and vertical filters as asked in the Assignment sheet.

[ ]:
```
# Your code to answer 3.3, 3.4 and displaay results here.
sobel_x = np.array([
    [-1, 0, 1],
    [-2, 0, 2],
    [-1, 0, 1]
])

sobel_y = np.array([
    [-1, -2, -1],
    [ 0,  0,  0],
    [ 1,  2,  1]
])

# Apply to an image at different scales
conv_x_1 = conv(image5, sobel_x)
conv_x_2 = conv(resize(image5, 800, 800), sobel_x)
conv_x_3 = conv(resize(image5, 500, 500), sobel_x)

conv_y_1 = conv(image5, sobel_y)
conv_y_2 = conv(resize(image5, 800, 800), sobel_y)
conv_y_3 = conv(resize(image5, 500, 500), sobel_y)

display(conv_x_1, "Sobel X")
display(conv_x_2, "Sobel X Smaller")
display(conv_x_3, "Sobel X Smallest")
display(conv_y_1, "Sobel Y")
display(conv_y_2, "Sobel Y Smaller")
display(conv_y_3, "Sobel Y Smallest")

# Apply different combinations of horizontal filters
combined_1 = conv(conv_x_1, sobel_y)
combined_2 = conv(conv_x_2, sobel_y)
combined_3 = conv(conv_x_3, sobel_y)
combined_4 = conv(conv_y_1, sobel_x)
combined_5 = conv(conv_y_2, sobel_x)
combined_6 = conv(conv_y_3, sobel_x)

# Display combined results
display(combined_1, "Y on X")
display(combined_2, "Y on X")
display(combined_3, "Y on X")
display(combined_4, "X on Y")
display(combined_5, "X on Y")
display(combined_6, "X on Y")
```

From the above generated images, it is clear the effect of the horizontal and vertical sobel filter on an image.

The horizontal Sobel filter (Sobel X filter) detects verticle edges, where there is a transition of intensity along the horizontal axis. In the above images, when the Sobel X filter was convolved with the original image, it effectively highlights all verticle edges in the original image.

The vertical Sobel filter (Sobel Y filter) detects horizontal edges, where there is a transition of intensity along the vertical axis. When applied in the above images, it effectively highlights all horizontal edges within the original image.

When applying the filters on images of different scale, the edge detection present was altered significantly. The original images, when convolved with the Sobel filter, produced much finer and sharper edges, whereas when the Sobel filter was convolved with smaller images produced edges that contained far more blurring. This can be attributed to the smaller number of pixels in smaller images, as it provides less information for the filter to utilise in its convolution.

When I apply the vertical Sobel filter to an image that has already been convonvled with a horizontal Sobel filter or vice versa, the resultant image will be that which satisfies an appropriate combination of the two. The resulting image will emphasise both vertical and horizontal edges in an image, while the order of application of the filters does very little to change the outcome. As seen above, the order changes very little of the output image and rather captures the gradient of the image in both directions.

2.5 Question 4: Image sampling and pyramids (30%)

2.5.1 4.1 Image Sampling

- Apply your `resize()` function to reduce an image (I) to 0.5*height and 0.5*width

- Repeat the above procedure, but apply a Gaussian blur filter to your original image before downsampling it. How does the result compare to your previous output, and to the original image? Why?

2.5.2 4.2 Image Pyramids

- Create a Gaussian pyramid as described in week2's lecture on an image.
- Apply a Gaussian kernel to an image I, and resize it with ratio 0.5, to get I_1. Repeat this step to get I_2, I_3 and I_4.
- Display these four images in a manner analogus to the example shown in the lectures.

4.1 When the Gaussian blur filter is applied to the image prior to downsampling it, it appears slightly blurred. When this output is compared to the image that was simply downsampled, the pixelation that has occurred within the image becomes very apparent. When both of these images are compared to the original image, the one that had the Gaussian blur fliter applied to it looks far cleaner in its appearance.

The reason the image that had the Gaussian blur filter applied to it looks far cleaner is because the image that didn't have blur applied experiences aliasing, which causes jagged edges, colour distortion, and other effects. By blurring the image before downsampling, you effectively remove any sharp objects in the image, which creates a far more visually consistent image.

```
[ ]: # Your answers to question 4 here
imageShape = image7.shape

# Original
display(image7)

# Resize no blur
resizeImage1 = resize(image7, int(0.5 * imageShape[0]), int(0.5 *␣
 ↪imageShape[1]))
display(resizeImage1, "Resize no blur")

# Resize and blur
convImage1 = conv(image7, gauss2D(3, 1))
resizeImage1 = resize(convImage1, int(0.5 * imageShape[0]), int(0.5 *␣
 ↪imageShape[1]))
display(resizeImage1, "Resize with blur")

# Gaussian pyramid
pyramid = [image7]

plt.figure(figsize=(15, 5))
plt.subplot(1, 5, 1)
plt.imshow(pyramid[0])
plt.title("Level 0")
plt.axis('off')
```

```
for i in range(4):
  convImage = conv(pyramid[-1], gauss2D(3, 1))
  resizeImage = resize(convImage, int(0.5 * imageShape[0]), int(0.5 *␣
↪imageShape[1]))

  pyramid.append(resizeImage)

  plt.subplot(1, 5, i + 2)
  plt.imshow(pyramid[i + 1])
  plt.title("Level " + str(i + 1))
  plt.axis('off')
```

The Gaussian pyramid becomes progressively more blurred as the process is repeated, however the image maintains its relative quality. This occurs because each level of the pyramid applies a Gaussian blur applied to it, which assists in removing any sharp edges. Without this Gaussian blurring, the image would become incredibly pixelated and hard to identify.

By smoothing the image before downsampling, it ensures that the downsampled images have a more gradual transition in scaling, reducing pixelation.

2.6 Question 5: (optional, assesed for granting up to 20% bonus marks for the A1)

Image filtering lectures, particularly Lecture 2, have covered the details related to this question. This is a bonus question for the students to get opportunities to recover lost marks in the other parts of the assignment. **Note that the overall marks will be capped at 100%**.

2.6.1 5.1 Apply and analyse a blob detector

- Create a Laplacian of Gaussian (LoG) filter in the function `LoG2D()` and visualize its response on your images. You can use the template function (and hints therein) for the task if you wish.
- Modify parameters of the LoG filters and apply them to an image of your choice. Show how these variations are manifested in the output.
- Repeat the experiment by rescaling the image with a combination of appropriate filters designed by you for these assignment. What correlations do you find when changing the scale or modifying the filters?
- How does the response of LoG filter change when you rotate the image by 90 degrees? You can write a function to rotate the image or use an externally rotated image for this task.

[ ]:
```
# Your code to answer question 5 and display results here
filter = LoG2D(3,1)
print(filter)
```