COMP37212 Convolution & kernels

Jeremy B. Roe

May 10, 2021

1 Introduction

This coursework aimed to implement convolution manually and then do some experiments to find out the effect of convolution on an image with the average and the weighted average smoothing kernels. The convolutions should be used to find the edge strength within the image for edge detection. I used the image *"kitty.bmp"* that was provided for the coursework.

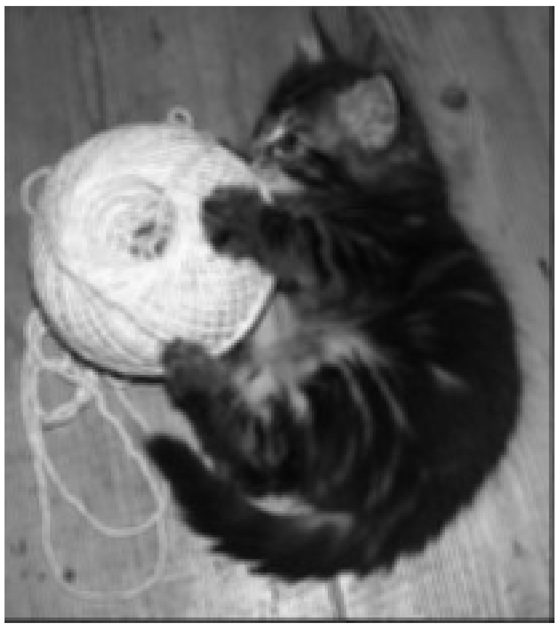


Figure 1: Kitty.bmp

2 Implementation

I created a python file *"kernels.py"*, which held all the kernels used in my experiments. This makes it easier to choose which kernels I want to use without hard coding them in the *"main.py"* file. These kernels included:

- a 3x3 mean kernel,
- a 3x3 Gaussian kernel,
- a 5x5 Gaussian kernel,
- a 7x7 Gaussian kernel,[1]
- two 3x3 Prewitt kernels (one for the x-axis, and one for the y-axis),
- and two 3x3 Sobel kernels (one for the x-axis, and one for the y-axis).

I also added the ability for the user to specify the image as a parameter, otherwise, it would use the default image provided.

I created a convolution function (*task 1 of the coursework*), which took the image, the kernel used, the image name, and Boolean values for whether you want to save and whether you wish to view the convoluted result. In order to pad the image, I checked the size of the kernel and then calculated the correct amount of padding to use. The padding has a value of zero. I used a cv2 function that adds padding for the image.[2]

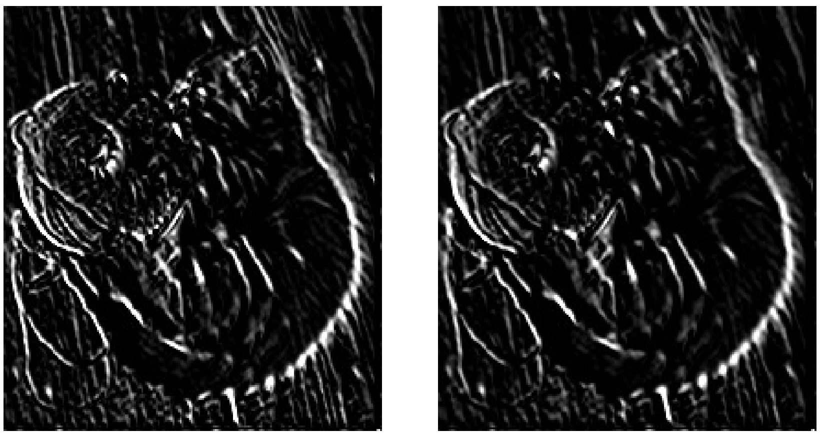
```
# add padding around image
n = len(kernel)
padding = int((n - 1) / 2)
padded_grey = cv2.copyMakeBorder(src=image,
                                 top=padding,
                                 bottom=padding,
                                 left=padding,
                                 right=padding,
                                 borderType=cv2.BORDER_CONSTANT,
                                 value=0)
```

The function would loop through each pixel in the image, taking a region of equal size to the kernel, and calculating the sum of all pixels with their weighting from the kernel added. In order to reduce numbers becoming too large or too small quickly, I divide by the sum of the kernel, provided that it is not zero-sum. I account for overflow and underflow by setting a maximum and minimum value of 255 and 0 respectively. The function then returns the convoluted image.

3 Convolution

For edge detection, I decided to do Sobel edge detection (*task 2 of the coursework*). Sobel uses two filters, one for the x-axis, and another for the y-axis. The results of these two images are then combined to show the gradient magnitude. I achieved this by going through each pixel and using the formula below to calculate the magnitude at each pixel.

$$|I'| = \sqrt{x'^2 + y'^2}$$



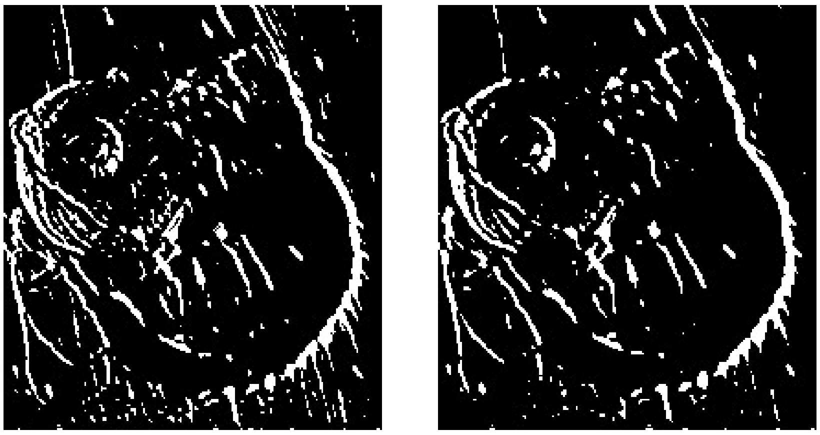
(a) Original (b) With 3x3 Gaussian Smoothing

Figure 2: Sobel filter edge strength of *"kitty.bmp"*

I discovered that smoothing the original image makes it easier to do edge detection as it reduces variation in the image. Small changes in depth discontinuity, surface colour discontinuity, and illumination discontinuity become less apparent, meaning only larger changes are left. This is apparent when you look at the difference in the Sobel filters when used against the original image and an image with a weighted-mean smoothing.

4 Thresholding

I implemented thresholding, where all values in the result of the Sobel image are either 0 or 255 based on whether it is above the threshold value.



(a) Original (b) With 3x3 Gaussian Smoothing

Figure 3: Threshold of 80 on Sobel Image of *"kitty.bmp"*

This shows that at a threshold of 80, the image with Gaussian smoothing has less noise in the image, as small magnitude pixels would have a value below the pixel, and are thus not shown. When they are all set at the same threshold value, you can see that those with blurring have fewer lines at that threshold, which makes it easier for detecting edges.

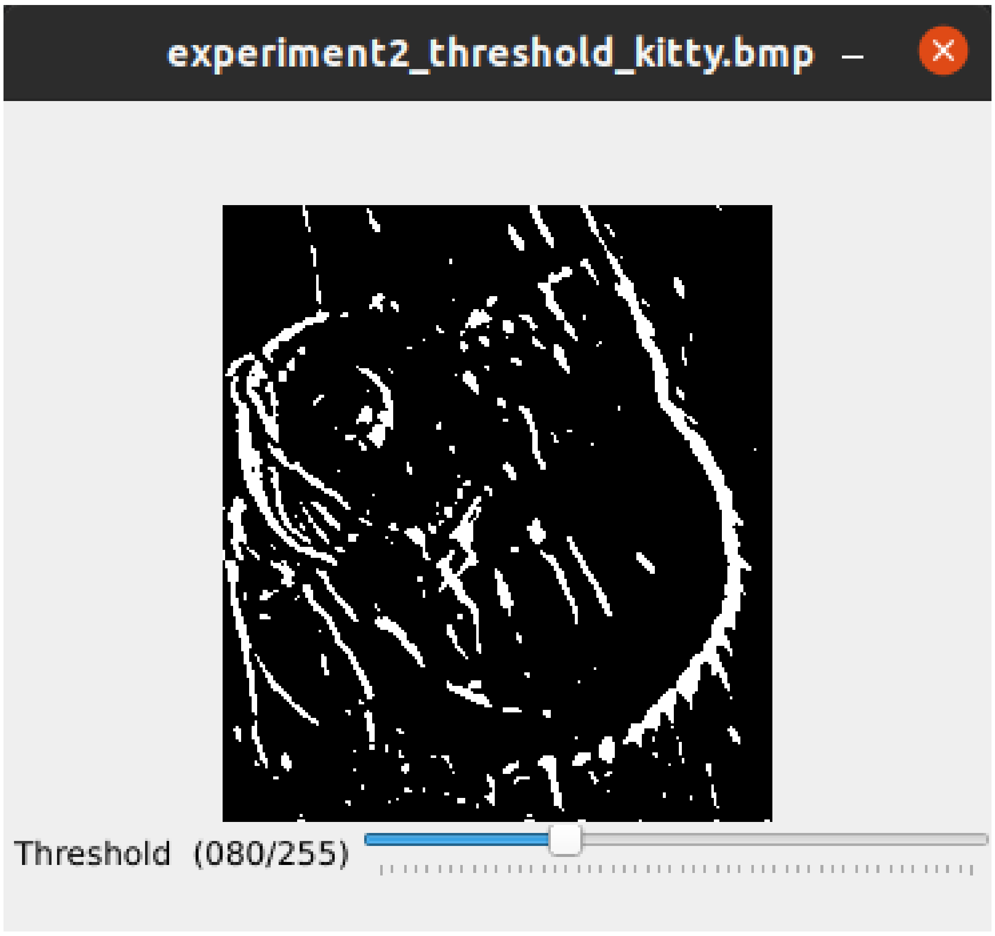


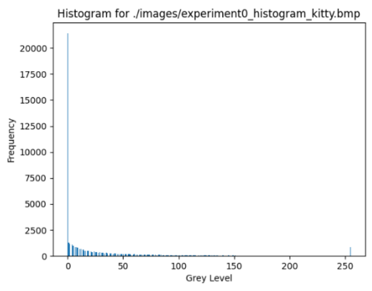
Figure 4: thresholding on 3x3 Gaussian blurred Sobel image

I made a track bar to change the threshold which allows me to change the threshold values and dynamically see what the image looks like at each value. This helps me find the best threshold that has the least amount of noise and only shows the bigger edges that I care about.

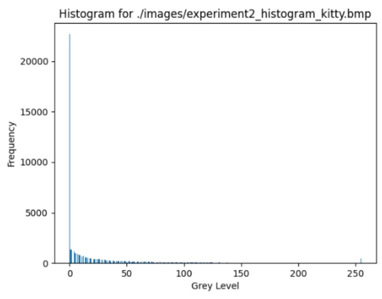
5 Results

One way I could have improved upon this is to use adaptive thresholding. This means that I can calculate threshold values for smaller regions. Thus, if there is something that I do not want to show in the final image, I can give it a higher threshold so that fewer lines show up in that specific area.

The threshold value that I found worked best for the original image's thresholding was around 150. However, the threshold value for the 3x3 smoothing kernel was around 120.



(a) Original



(b) With 3x3 Gaussian Smoothing

Figure 5: Histogram of Sobel filter of *"kitty.bmp"*

From this coursework, I have learnt that smoothing an image makes less variation, which in turn makes edge detection better.

If you would like to run my code, please run it in the command line as follows:

```
python3 main.py <image file path>
```

"kernels.py" and *"main.py"* need to be in the same directory for this to work. And the images should be in a folder in the same directory as the files.

References

[1] Oleg Shipitko and Anton Grigoryev. "Gaussian filtering for FPGA based image processing with High-Level Synthesis tools". In: *Institute for Information Transmission Problems (Kharkevich Institute) – IITP RAS, Bol'shoy Karetnyy per. 19, Moscow, Russia, 127051* (2018).

[2] openCV. *openCV documentation: copyMakeBorder()*. URL: `https://docs.opencv.org/3.4/d2/de8/group__core__array.html#ga2ac1049c2c3dd25c2b41bffe17658a36`.