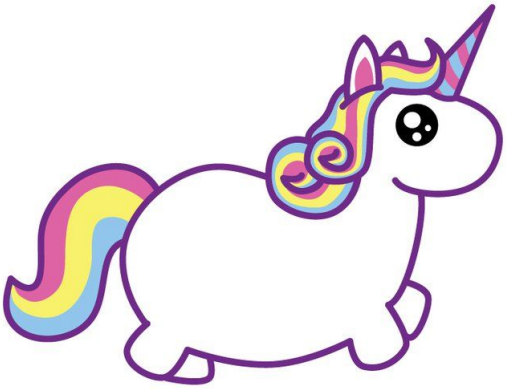
Separable Kernels and Edge Detection

CS1230

Disclaimer: For Filter, using separable kernels is optional. It makes your implementation faster, but if you can't get it to work, that's totally fine! Just use 2D kernels instead.



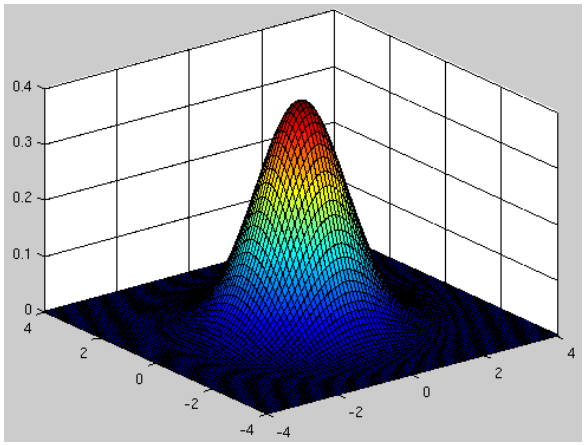
SEPARABLE KERNELS

The lecture slides concerning this subject are Image Processing IV, slides 35 - 39.

Let's start off by looking at a 2D Gaussian kernel with a standard deviation of $\frac{5}{3}$ and width of 5:

$$\begin{bmatrix} 0.018 & 0.031 & 0.037 & 0.031 & 0.018 \\ 0.031 & 0.052 & 0.062 & 0.052 & 0.031 \\ 0.037 & 0.062 & 0.074 & 0.062 & 0.037 \\ 0.031 & 0.052 & 0.062 & 0.052 & 0.031 \\ 0.018 & 0.031 & 0.037 & 0.031 & 0.018 \end{bmatrix}$$

This is what a 2D Gaussian generally looks like (not actually this one):



Here's what it looks like if we take the kernel and apply it to a picture of a unicorn:

This kernel is difficult to compute and also runs really slowly when we try to use it to convolve an image, so we probably want a better solution. Good news is, there is! A Gaussian kernel is **separable**. What this means is, if we break the kernel down into two 1D filters, do one pass in the horizontal direction, and another pass in the vertical direction, we will get the exact same result.

This is a little easier to explain with an example. Let's generate a 1D Gaussian again with a standard deviation of $\frac{5}{3}$:

$$\begin{bmatrix} 0.136 & 0.228 & 0.272 & 0.228 & 0.136 \end{bmatrix}$$

What we're going to do is take this kernel and first apply it to our image in the horizontal direction:

Now that the image is blurred horizontally, we'll take this intermediate image and apply the kernel in the vertical direction:

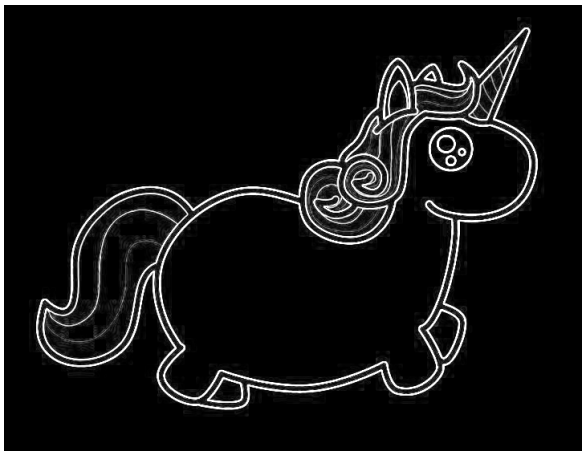
We can see that our resulting image is exactly the same as when we used the 2D Gaussian kernel! We don't necessarily have to do the passes in that specific order, either. You're free to do a vertical pass first, then a horizontal pass, and you'll get the same result.

Now, not all kernels are separable. Some kernels that **are separable** are box, Gaussian, and Sobel (we will discuss this in a bit). Kernels that **are not separable** are cone and pyramid. You might think that a pyramid kernel is separable into two triangle filters, but that's not actually the case, which you can see on slide 37 of Image Processing IV. **For the purposes of Filter, however, making two passes with a triangle filter is close enough, so feel free to do so**.

EDGE DETECTION

The information in this section can be found in the Edge Detection section of the Filter assignment.

Earlier, we mentioned the Sobel kernels. The Sobel kernels are two kernels that are used for edge detection. Here's what edge detection on our unicorn looks like:



Pretty cool, right? Here's how we did it.

Edge detection is actually **a combination of two independent convolutions**! Let's go more in-depth as to how this works. What edge detection does is approximate the derivative of the pixel intensities in the image. We do this once in the horizontal direction and once in the vertical direction, then combine the two results together.

Taking derivatives of the image is actually very simple: it's simply just applying a convolution with a 3x3 kernel. The kernels we're going to need are what we call the Sobel kernels:

$$\text{kernel}_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}, \text{kernel}_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

where kernel_x is the kernel we need to get the derivative in the horizontal direction and

kernel_y is the kernel we need to get the derivative in the vertical direction.

Remember what we said about separable kernels in the first section? It turns out, both kernels are actually separable! The way we separate kernel_x is:

$$\begin{aligned}&\text{Convolve by } \begin{bmatrix}1 & 0 & -1\end{bmatrix} \text{ in the horizontal direction}\\&\text{Convolve by } \begin{bmatrix}1 & 2 & 1\end{bmatrix} \text{ in the vertical direction}\end{aligned}$$

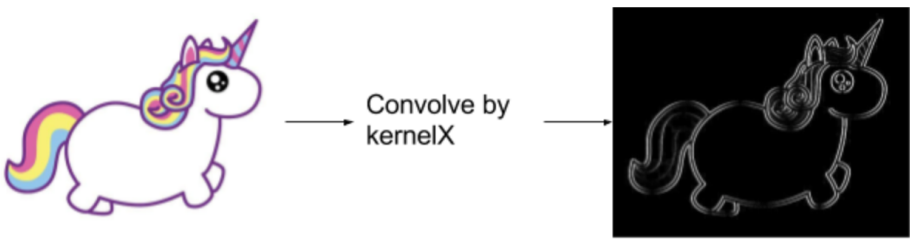
And the way we separate kernel_y is:

$$\begin{aligned}&\text{Convolve by } \begin{bmatrix}1 & 2 & 1\end{bmatrix} \text{ in the horizontal direction}\\&\text{Convolve by } \begin{bmatrix}1 & 0 & -1\end{bmatrix} \text{ in the vertical direction}\end{aligned}$$

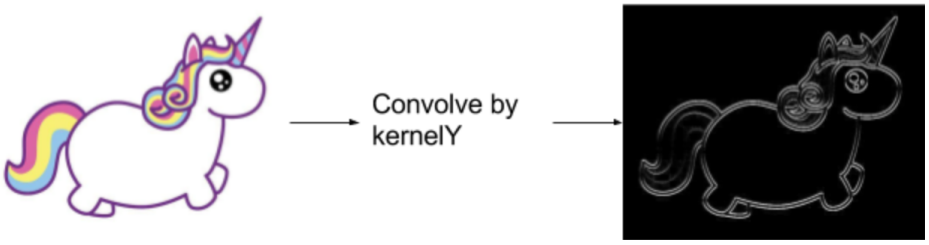
At the end, after obtaining both resulting images, we want to combine the results together into a single image. For each pixel, if we let $\boldsymbol{G}_x$ be the intensity in the image we used kernel_x with and let $\boldsymbol{G}_y$ be the intensity in the image we used kernel_y with, then our intensity in the final image will be:

$$\boldsymbol{G} = \sqrt{\boldsymbol{G}_x^2 + \boldsymbol{G}_y^2}$$

Let's show each of these steps with our unicorn again! When we apply kernel_x to the unicorn, we get:



Now, we **start over with the original image** and apply kernel_y:



And now with our two resulting images, we can combine the results:

Good luck with Filter!

No unicorns were harmed in the making of this document. Maybe.

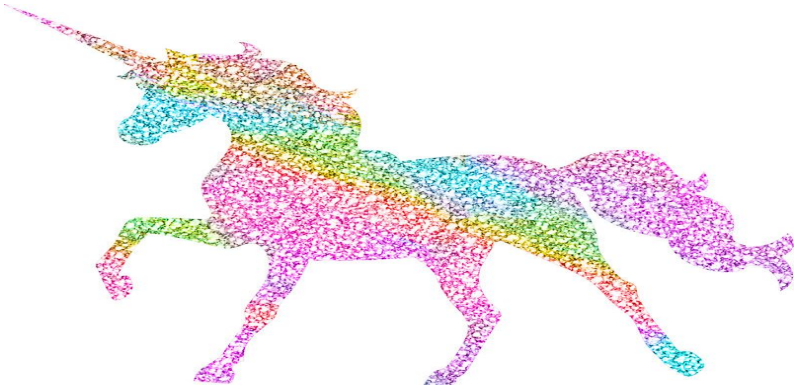
How to Scale

Hey everyone ! It seems like there's some confusion on how to scale, so I'm gonna try to describe it a little here.

First, we have our original source image (the pre-scaled image) and our destination image (the scaled image).

Original source image

1. Scale in the x-direction. This means the height of the image will remain the same (for now) but the width of the image will shrink/grow.



Original image scaled in the x-direction

2. Use the image you got from scaling in the x-direction to scale in the y-direction.

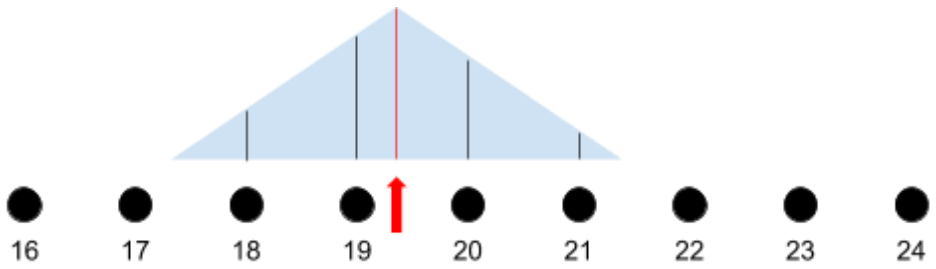
X-scaled image scaled in the y-direction
Also, our destination image!

For now, we'll just do an example about scaling in the x direction but the steps will be similar for scaling in the y direction.

For every pixel in our **x-scaled image**, we want to do these following steps:

1. Use the back map function (in lecture slides / algos) to figure out what pixel we want to sample from in our **original image.**
    - If we are calculating the pixel value at `row` and `col` in our **x-scaled image**, then we use the backmapping function to find what column we want to sample from in our original image. For scaling in the x-direction, we sample from the same row in our original image, since our height hasn't changed yet.
      (For scaling in y-direction, we would use backmapping to find the row to sample from.)
    - Using the backmap function, we want to sample from row `row` and column $\frac{col}{a} + \frac{1-a}{2a}$.

2. Next, as in the algos, we place our "triangle" filter over the pixel (`row`, $\frac{col}{a} + \frac{1-a}{2a}$) in our **original image**.



Here, this represents the pixels of row `row` *in our original image.*
The red arrow is the value of $\frac{col}{a} + \frac{1-a}{2a}$ *, which is the center of our triangle.*

- Though the value for the column may be fractional, we are still only sampling discrete pixel values.
  - Notice how column 17 and column 22 fall out of range of our support width?

3. Then, we will be sampling the pixels that fall underneath the triangle support width.
   - For the image above, that would be the pixels in columns 18 to 21.

4. For each pixel, we want to multiply the pixel value with the height of the triangle above that point.
   - Use the equation from slide 25 in Viewing IV to figure out the height of the triangle for each point!

5. After sampling pixels and multiplying them with the triangle height, add them up!
   - That is the value of the pixel in (`row, col`) of your **x-scaled** image!
     - So in our image above, that would be:

$$image(row, 18) * height_{18} + image(row, 19) * height_{19} + image(row, 20) * height_{20} + image(row, 21) * height_{21}$$

6. Continue calculating all the pixels of your x-scaled image, and you'll have an image scaled in the x-direction!

For scaling in the y-direction, similar steps will be applied. But make sure you backmap to the x-scaled image instead of the original image!

Good luck with Filter!