University of Michigan
EECS 442: Computer Vision
Fall 2021. Instructor: Andrew Owens.

Problem Set 1: Image filtering

Posted: Monday, August 30, 2021 **Due:** Wednesday, September 15, 2021

For Problem 1.1, please submit your written solution to Gradescope as a `.pdf` file. For Problem 1.2, please submit your solution to Canvas as a notebook file (`.ipynb`), containing the visualizations that we requested. Please also follow our instructions to convert the your Colab notebook to a PDF file and submit the PDF file to Gradescope. For your convenience, we have included the PDF conversion script at the end of the notebook.

The starter code can be found at:
`https://colab.research.google.com/drive/1EkMRJ7HsILBvAZrLd3OX2MitI2ZnvDVF?usp=sharing`

We recommend editing and running your code in Google Colab, although you are welcome to use your local machine instead.

Problem 1.1 *Properties of convolution*

Recall that 1D convolution between two signals $f, g \in \mathbb{R}^N$ is defined:

$$(f * g)[n] = \sum_{k=0}^{N-1} f[n-k]g[k]. \tag{1}$$

(a) Show that convolution is commutative, i.e. $f*g = g*f$. You may assume that out-of-bounds indices are 0, e.g. $f[-1] = 0$, or whatever is convenient. **(1 point)**.

Hint: Write the equation for $f * g$, and figure out how to "rename" the variable used in the summation to arrive at $g * f$.

(b) Show that convolution is associative, i.e. $(f * g) * h = f * (g * h)$ (optional). *Note: we will not grade this problem. You will* ***not*** *get bonus points for completing it. It is totally optional. Only do this problem if it interests you!*

(b) Construct a matrix multiplication that produces the same result as a 1D convolution with a given filter. In other words, given a filter f describe a matrix H such that $f * g = Hg$ for any input g. Here Hg denotes matrix multiplication between the matrix H and vector g **(1 point)**.

(c) In class, we showed how convolution with a 2D Gaussian filter can be performed efficiently as a sequence of convolutions with 1D Gaussian filters. This idea also works with other kinds of filters. We say that a 2D filter $F \in \mathbb{R}^{N \times N}$ is separable if $F = uv^\top$ for some $u, v \in \mathbb{R}^N$, i.e. F is the outer product of u and v. Show that if F is separable, then the 2D convolution $G * F$ can be computed as a sequence of two one-dimensional convolutions **(1 point)**.

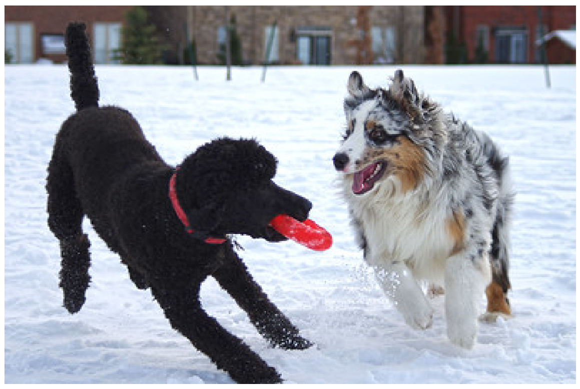
Hint: Start by writing down the expression for $G * F$ and then, inside the double summation, write F in terms of u and v.

(d) ***(Optional)*** Show that cross-correlation,

$$h[n] = \sum_{k=0}^{N-1} f[n+k]g[k], \tag{2}$$

is *not* commutative (0 points).

Problem 1.2 *Pet edge detection*


(a)

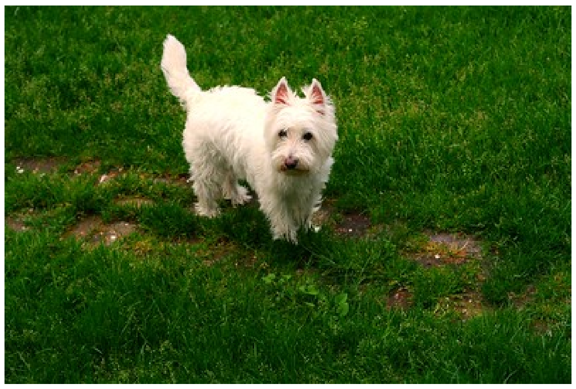

(b)

Figure 1: (a) A pet photo helpfully delivered by our sponsor, (b) a failure case for a simple edge detector. These images are provided in the starter code. Photo credits for this problem set can be found here.

As you know, this course is largely funded by grants from Petco, Inc.™ One of the strings attached to this funding is that we must occasionally assign *sponsored* problems that, regrettably, cover topics with significant industrial implications for our sponsor instead of the usual course material[1]. Our partners at Petco see an opportunity to use computer vision to pull ahead of their bitter rivals, Petsmart™. Rather than laboriously marking the edges of dogs and cats in pictures by hand—the current industry practice—they have turned to you for an automated solution.

(a) Using the starter code provided, apply the horizontal and vertical gradient filters [1 -1] and $[1 \ \ -1]^\top$ to the picture of the provided pet[2] producing filter responses I_x and I_y. Write a function `convolve(im, h)` that takes a grayscale image and a 2D filter as input, and returns the result after convolution. Please do not use any "black-box" filtering functions for this,

[1]Be thankful for this sponsorship, though. Recall that in previous semesters, the whole class shared a single GPU, which the course staff shuttled between students' homes in a rented van.

[2]These convolution filters are 1×2 and 2×1 matrices respectively. Even though they are not square, your convolution function should have no problem using them.

such as the ones in `scipy`[3]. Instead, implement the convolution as a series of nested `for` loops (you may also use `numpy.dot`, but it is not necessary). Compute the *edge strength* as $I_x^2 + I_y^2$. After that, create visualizations of I_x, I_y, and the edge strength, following the sample code.

The filter response that your function returns should be the same dimensions as the input image. Please use *zero padding*, i.e. assume that out-of-bounds pixels in the image are zero. Also please make sure to implement convolution, *not* cross-correlation. Note that this simple filtering method will have a fairly high error rate — there will be true object boundaries it misses and spurious edges that it erroneously detects. The team at Petco, thankfully, has volunteered to painstakingly fix any errors by hand **(2 points)**.

(b) ***(Optional)*** This method detects edges fairly well on one of the dogs, but not the other. Our sponsor would like us to investigate why this is happening. First, convert your gradient outputs to edge detections using a hand-chosen threshold τ (i.e. set values at most τ to 0 and those above to 1). Point out 2 errors in the resulting edge map — that is, edge detections that do not correspond to the boundary of an object — and explain what causes these errors (0 points).

(c) While the edge detector you submitted works well on some pets, engineers are reporting a large number of failures, and Petco higher ups are *not* happy. Rumor is that it's failing on pets that are playing in grass, especially for dogs with fluffy coats — a particularly lucrative market for our sponsor (Figure 1b). It appears that the gradient filter is firing on small, spurious edges.

Kindly address our funders' problem by creating an edge detector that only responds to edges at larger spatial scales. Do this by first blurring the image with a Gaussian filter, before computing gradients. Implement your Gaussian filter on your own. Please do not use any "black-box" Gaussian filter function for this, such as `scipy.ndimage.gaussian_filter`. Apply both the Gaussian filter and the gradient filter using a black-box convolution function `scipy.ndimage.convolve` on the image, rather than your hand-crafted solution.

(i) Compute the edges without blurring, so we can look at the before-and-after results.

(ii) Compute the blurred image using $\sigma = 2$ and an 11×11 Gaussian filter.

(iii) Instead of blurring the image with a Gaussian filter, use a box filter (i.e., set each of the 11×11 filter values to $1/11^2$).

(iv) Compute edges on the two blurred images.

(v) ***(Optional)*** Do you see artifacts in the box-filtered result? Describe how the two results differ. Include your written response in the notebook.

Visualize the edges in the same manner as Problem 1.2(a). Your notebook should contain edges that were computed using no blurring, Gaussian blurring, and box filtering. Please also show the two blurred images **(2 points)**.

(d) Instead of convolving the image with two filters to compute I_x (i.e. a Gaussian blur followed by a gradient), create a *single* filter Gx that yields the same response. You can reuse the function in part (c). Visualize this filter using the provided code **(1 point)**.

[3]Our sponsor has been prohibited from from using these functions as part of the terms of an lawsuit that—best we can tell from the outside—involves a dog bite and a key member of the `scipy` development team.

(e) Good news. Petco execs are pleased with your work, and they've renewed our funding for several more problem sets. At our last grant meeting, however, they made an additional request. Their competitors at Petsmart are now computing *oriented* edges of their pets: rather than just estimating just the horizontal or the vertical gradients, they now provide gradients for an arbitrary angle θ. Petsmart's method for doing this, however, is computationally expensive: they construct a new filter for each angle, and filter the image with it. Petco sees an opportunity to beat their rivals using steerable filters.

Write a function `oriented_grad`(I_x, I_y, θ) that returns the image gradient steered in the direction θ, given the horizontal and vertical gradients I_x and I_y. Use this function to compute your gradients on a blurred version of the input image at $\theta \in \{\frac{1}{4}\pi, \frac{1}{2}\pi, \frac{3}{4}\pi\}$, using the same Gaussian blur kernel as (c). Visualize these results in the same manner as Problem 1.2 (a) **(1 point)**.