University of Michigan
EECS 504: Foundations of Computer Vision
Winter 2020. Instructor: Andrew Owens.

Problem Set 1: Image filtering

Posted: Thursday, January 9, 2020 **Due:** Tuesday, January 21, 2020

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.

The starter code can be found at:
`https://colab.research.google.com/drive/1hXN9fx8DaVm4BM4lm2vUj9Ut2_TMdyC5`

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 (i) is commutative, i.e. $f * g = g * f$ and (ii) associative, i.e. $(f * g) * h = f * (g * h)$ (1 point).

(b) Construct a matrix multiplication that produces the same result as 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 (1 point).

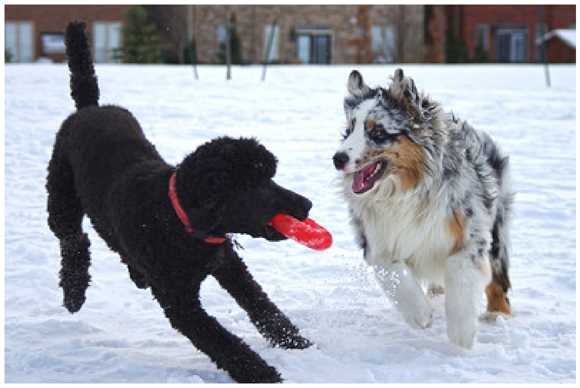
(c) In class, we showed how convolution with a 2D Gaussian filter can be performed efficiently as a sequence of convolutions with 1D Gaussian filters. This idea also works with other kinds of filters. We say that a 2D filter $f \in \mathbb{R}^{N\times N}$ is separable if $f = uv^\top$ for some $u, v \in \mathbb{R}^N$. 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).

(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). *Note: we will not grade this problem. It really is totally optional! Only do this problem if it interests you.*

Problem 1.2 *Pet edge detection*


(a)

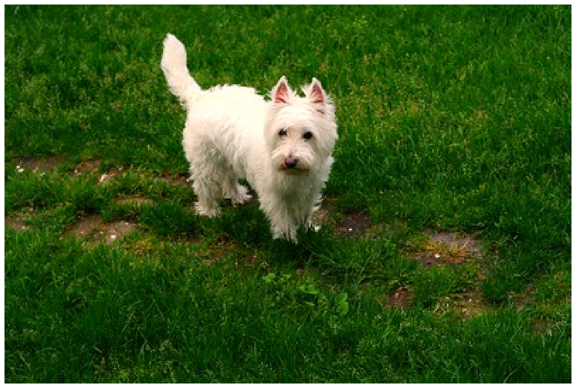

(b)

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 the occasional *sponsored* problem where, regrettably, we must provide a problem that has substantial industrial implications for our sponsor[1]. Our friends at Petco see an opportunity to use computer vision to pull ahead of their rivals, Petsmart™, with whom they are locked in bitter competition. Rather than laboriously marking the edges of dogs and cats in pictures by hand—the current industry practice—they have turned to you.

(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, 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 (such as the ones in `scipy`). Instead, implement the convolution as a series of nested `for` loops (you may also use `numpy.dot`, but it is not necessary). Compute the total edge strength as $I_x^2 + I_y^2$, and visualize it using `matplotlib`, following the sample code. Create visualizations of I_x, I_y, and the combined response, following the sample code.

Please handle out-of-bounds pixels in the image by assuming that they are 0. 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 manually fix any errors (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).

[1]Be thankful for this sponsorship, though. Recall that in previous semesters, the class shared a single GPU, which the course staff had to personally shuttle between students' homes.

(c) While the edge detector you submitted works well on some pets, engineers are reporting a large number of failures, and Petco execs are *not* happy. Rumor is that it's failing on pets that are playing in grass, especially for dogs with fluffy coats (Figure 1b) — a particularly lucrative market for our sponsor. 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 larger-scale edges. Do this by first blurring the image with a Gaussian filter, before computing gradients. Implement both the Gaussian filter and the gradient filter using a black-box convolution function `scipy.ndimage.convolve`, 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 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 that yields the same response. Visualize the filter using the provided code (1 point).

(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 Petsmart at their own game using steerable filters.

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