Introduction to Computer Vision: Assignment 1

Instructions

- This homework is **due before the class on monday march 31th, 2025**.
- The submission includes two parts:
  1. Submit a `zip` file of all of your code.
     We have indicated questions where you have to do something in code in red.

     Your zip file should contain a single directory which has the same name of your student number, e.g. the zip file should contain a single folder `2023123123/` containing all required files.

  2. Submit a `pdf` file as your write-up, including your answers to all the questions and key choices you made.
     We have indicated questions where you have to do something in the report in blue.
     You might like to combine several files to make a submission. Here is an example online link for combining multiple PDF files: https://combinepdf.com/.
     The write-up must be an electronic version. **No handwriting, including plotting questions**

Python Environment We are using Python 3.7 for this course. You can find references for the Python standard library here: https://docs.python.org/3.7/library/index.html. To make your life easier, we **recommend** you to install Anaconda for Python 3.7.x (https://www.anaconda.com/download/). This is a Python package manager that includes most of the modules you need for this course.

We will make use of the following packages extensively in this course:

- Numpy (https://docs.scipy.org/doc/numpy-dev/user/quickstart.html)
- SciPy (https://scipy.org/)
- Matplotlib (http://matplotlib.org/users/pyplot tutorial.html)

1 Patches [8 pts]

Task 1: Image Patches (8 pts) A patch is a small piece of an image. Sometimes we will focus on the patches of an image instead of operating on the entire image itself.

(a) **Complete the function** `image_patches` in `filters.py`. This should divide a grayscale image into a set of non-overlapping 16 by 16 pixel image patches. Normalize each patch to have zero mean and unit variance.

Plot and put in your report three 16x16 image patches from `grace_hopper.png` loaded in grayscale. (3 pts)

(b) **Discuss in your report** why you think it is good for the patches to have zero mean. (2 pts)

Hint: Suppose you want to measure the similarity between patches by computing the dot products between different patches to find a match. Think about how the patch values and the resulting similarity obtained by taking dot products would be affected under different lighting/illumination conditions. Say in one case a value of dark corresponds to 0 whereas bright corresponds to 1. In another scenario a value of dark corresponds to -1 whereas bright corresponds to 1. Which one would be more appropriate to measure similarity using dot products?

(c) Early work in computer vision used patches as descriptions of local image content for applications ranging from image alignment and stitching to object classification and detection.

Discuss in your report in 2-3 sentences, why the patches from the previous question would be good or bad for things like matching or recognizing an object. Consider how those patches would look like if we changed the object's pose, scale, illumination, etc. (3 pts)

2 Image Filtering [46 pts]

Foreword: There's a difference between convolution and cross-correlation: in cross-correlation, you compute the dot product (i.e., `np.sum(F*I[y1:y2,x1:x2])`) between the kernel/filter and each window/patch in the image; in convolution, you compute the dot product between the *flipped* kernel/filter and each window/patch in the image. We'd like to insulate you from this annoying distinction, but we also don't want to teach you the wrong stuff. So we'll split the difference by pointing where you have to pay attention.

We'll make this more precise in 1D: assume the input/signal f has N elements (i.e., is indexed by i for $0 \leq i < N$) and the filter/kernel g has M elements (i.e., is indexed by j for $0 \leq j < M$). In all cases below, you can assume *zero-padding* of the input/signal f.

1D Cross-correlation/Filtering: The examples given in class and what most people think of when it comes to filtering. Specifically, 1D cross-correlation/filtering takes the form:

$$h[i] = \sum_{j=0}^{M-1} g[j] f[i+j], \tag{1}$$

or each entry i is the sum of all the products between the filter at j and the input at $i+j$ for all valid j. If you want to think of doing this in terms of matrix products, you can think of this as $\mathbf{h}_i = \mathbf{g}^T \mathbf{f}_{i:i+M-1}$. Of the two options, this tends to be more intuitive to most people.

1D Convolution: When we do 1D convolution, on the other hand, we re-order the filter last-to-first, and then do filtering. In signal processing, this is usually reasoned about by index trickery. By definition, 1D convolution takes the form:

$$(f * g)[i] = \sum_{j=0}^{M-1} g[M-j-1]f[i+j], \tag{2}$$

which is uglier since we start at 0 rather than 1. You can verify that as j goes $0 \to (M-1)$, the new index $(M-j-1)$ goes $(M-1) \to 0$. Rather than deal with annoying indices, if you're given a filter to apply and asked to do convolution, you can simply do the following: (1) at the start of the function and only once, compute `g = g[::-1]` if it's 1D or `G = G[::-1,::-1]`; (2) do filtering with this flipped filter.

The reason for the fuss is that convolution is commutative ($f * g = g * f$) and associative ($f * (g * h) = (f * g) * h$). As you chain filters together, its is nice to know things like that $(a * b) * c = (c * a) * b$ for all a, b, c. Cross-correlation/filtering does not satisfy these properties.

You should watch for this in three crucial places.

- When implementing convolution in Task 2(b) in the function `convolve()` in `filters.py`.
- When dealing with non-symmetric filters (like directional derivatives $[-1, 0, 1]$). A symmetric filter like the Gaussian is unaffected by the distinction because if you flip it horizontally/vertically, it's the same. But for asymmetric filters, you can get different results. In the case of the directional derivatives, this flips the sign. This can produce outputs that have flipped signs or give you answers to question that are nearly right but need to be multiplied by -1.

  Bottom-line: if you have something that's right except for a -1, and you're using a directional derivative, then you've done it with cross-correlation.
- In a later homework where you implement a convolutional neural network. Despite their name, these networks actually do cross-correlation. Argh! It's annoying.

  Here's my key: if you're trying to produce a picture that looks clean and noise-free or you're talking to someone who talks about sampling rates, "convolution" is the kind of convolution where you reverse the filter order. If you're trying to recognize puppies or you're talking to someone who doesn't frequently say signal, then "convolution" is almost certainly filtering/cross-correlation.

Task 2: Convolution and Gaussian Filter (21 pts

(a) A Gaussian filter has filter values that follow the Gaussian probability distribution. Specifically, the values of the filter are

$$\text{1D kernel} : G(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{x^2}{2\sigma^2}\right) \qquad \text{2D kernel} : G(x,y) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2+y^2}{2\sigma^2}\right)$$

where 0 is the center of the filter (in both 1D and 2D) and σ is a free parameter that controls how much blurring takes place. One thing that makes lots of operations fast is that applying a 2D Gaussian filter to an image can be done by applying two 1D Gaussian filters, one vertical and the other horizontal.

Show in your report that a convolution by a 2D Gaussian filter is equivalent to sequentially applying a vertical and horizontal Gaussian filter. (3 pts)

Advice: Pick a particular filter size k. From there, define a 2D Gaussian filter $\mathbf{G} \in \mathbb{R}^{k \times k}$ and two Gaussian filters $\mathbf{G}_y \in \mathbb{R}^{k \times 1}$ and $\mathbf{G}_x \in \mathbb{R}^{1 \times k}$. A useful fact that you can use is that for any k, any

vertical filter $\mathbf{X} \in \mathbb{R}^{k \times 1}$ and any horizontal filter $\mathbf{Y} \in \mathbb{R}^{1 \times k}$, the convolution $\mathbf{X} * \mathbf{Y}$ is equal to $\mathbf{XY}$. Expanded out for $k = 3$, this just means

$$\begin{bmatrix} X_1 \\ X_2 \\ X_3 \end{bmatrix} * \begin{bmatrix} Y_1 & Y_2 & Y_3 \end{bmatrix} = \begin{bmatrix} X_1Y_1 & X_1Y_2 & X_1Y_3 \\ X_2Y_1 & X_2Y_2 & X_2Y_3 \\ X_3Y_1 & X_3Y_2 & X_3Y_3 \end{bmatrix} \tag{3}$$

You may find it particularly useful to use the fact that $\mathbf{Y} * \mathbf{Y}^T = \mathbf{Y}\mathbf{Y}^T$ and that convolution is associative. Look at individual elements. If you do this correctly, the image does not have to be involved at all.

If you have not had much experience with proofs or need a refresher, this guide will help you get started. Here is another link that will help you write readable and easy to follow solutions. But in general, the key isn't formality, but just being precise.

(b) **Complete the function** `convolve()` in `filters.py`. Be sure to implement convolution and not cross-correlation/filtering (i.e., flip the kernel as soon as you get it). For consistency purposes, please use **zero-padding** when implementing convolution. (4 pts)

Advice: You can use `scipy.ndimage.convolve()` to check your implementation. For zero-padding use `mode='constant'`. Refer to documentation for details. For Part 3 Feature Extraction and Part 4 Blob Detection, directly use scipy's convolution function with the same settings, ensuring zero-padding.

(c) **Plot the following output and put it in your report** and then describe what Gaussian filtering does to the image in one sentence. Load the image `grace_hopper.png` as the input and apply a Gaussian filter that is 3×3 with a standard deviation of $\sigma = 0.572$. (2 pts)

(d) **Discuss in your report** why it is a good idea for a smoothing filter to sum up to 1. (3 pts)

Advice: As an experiment to help deduce why, observe that if you sum all the values with of the Gaussian filter in (c), you should get a sum close to 1. If you are very particular about this, you can make it exactly sum to 1 by dividing all filter values by their sum. When this filter is applied to `'grace_hopper.png'`, what are the output intensities (min, max, range)? Now consider a Gaussian filter of size 3×3 and standard deviation $\sigma = 2$ (but do not force it to sum to 1 – just use the values). Calculate the sum of all filter values in this case. What happens to the output image intensities in this case? If you are trying to plot the resulting images using `matplotlib.pyplot` to compare the difference, set `vmin = 0` and `vmax = 255` to observe the difference.

(e) Consider the image as a function $I(x, y)$ and $I : \mathbb{R}^2 \to \mathbb{R}$. When working on edge detection, we often pay a lot of attention to the derivatives. Denote the "derivatives":

$$I_x(x, y) = I(x + 1, y) - I(x - 1, y) \approx 2\frac{\partial I}{\partial x}(x, y)$$

$$I_y(x, y) = I(x, y + 1) - I(x, y - 1) \approx 2\frac{\partial I}{\partial y}(x, y)$$

where I_x is the twice the derivative and thus off by a factor of 2. This scaling factor is not a concern since the units of the image are made up. So long as you are consistent, things are fine.

Derive in your report the convolution kernels for derivatives (3 pts):

(i) $k_x \in \mathbb{R}^{1\times 3}$: $I_x = I * k_x$

(ii) $k_y \in \mathbb{R}^{3\times 1}$: $I_y = I * k_y$

(f) Follow the detailed instructions in `filters.py` and **complete the function** `edge_detection()` in `filters.py`, whose output is the gradient magnitude. (3 pts)

(g) Use the original image and the Gaussian-filtered image as inputs respectively and use `edge_detection()` to get their gradient magnitudes. **Plot both outputs and put them in your report**. **Discuss in your report** the difference between the two images in no more than three sentences. (3 pts)

Task 3: Sobel Operator (9 pts) The Sobel operator is often used in image processing and computer vision.

(a) The Sobel filters S_x and S_y are given below and are related to a particular Gaussian kernel G_S:

$$S_x = \begin{pmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{pmatrix} \qquad S_y = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix} \qquad G_S = \begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix}.$$

Show in your report the following result: If the input image is I and we use G_S as our Gaussian filter, taking the horizontal-derivative (i.e., $\frac{\partial}{\partial x} I(x, y)$) of the *Gaussian-filtered image*, can be approximated by applying the Sobel filter (i.e., computing $I * S_x$). (5 pts)

Advice: You should use the horizontal filter k_x that you derive previously – in particular, the horizontal derivative of the Gaussian-filtered image is $(I * G_S) * k_x$. You can take advantage of properties of convolution so that you only need to show that two filters are the same. If you do this right, you can completely ignore the image.

(b) **Complete the function** `sobel operator()` in `filters.py` with the kernels/filters given previously. (2 pts)

(c) **Plot the following and put them in your report:** $I * S_x$, $I * S_y$, and the gradient magnitude. with the image `'grace hopper.png'` as the input image I. (2 pts)

In the `zip` file you submit to PLATO, the directory named after your student number should include the following files:

- `filters.py`
- `corners.py`
- `blob detection.py`
- `common.py`
- `image patches`: directory with the detected images patches in it.
- `gaussian filter`: directory with filtered image and edge responses.
- `sobel operator`: directory with sobel filtered outputs.

The rest should be all included in your `pdf` report submitted to PLATO.