Introduction to Computer Vision (Spring 2022)
Assignment 1

Release date: March 13, due date: March 27, 2022 11:59 PM

The assignment includes 5 tasks: implementing convolution operation, Canny edge detector, Harris corner detector, plane fitting using RANSAC, and backpropagation for a toy classification network, which sums up to 100 points and will be counted as 10 points towards your final score of this course. This assignment is fully covered by the course material from Lecture 2 and 3.

The objective of this assignment is to get you familiar with coding and implementing the basic modules and algorithms of classic computer vision and deep learning **in a tensor style**. Here the tensor style refers to avoiding using any for loops and, instead, *implement everything via matrix or tensor operations, e.g. matrix multiplication*. This exactly resembles how people code in PyTorch or Tensorflow for the sake of parallel computing. We offer starting code for all the tasks and you are expected to implement the key functions using Python and Numpy.

Policy on using for loop/while: if a question doesn't indicate for loops are allowed, using for loop/while in your code will be penalized (1 point for 1 use). However, if your code is correct and shorter or more parallel than TA's answer, you will have a bonus. Some useful Numpy functions are included in Appendix 5 for your information.

After completing the tasks, please compress your code along with your results to *Name_ID.zip* following the original path structure, then submit to course.pku.edu.cn. Feel free to post in the discussion panel for any questions and we encourage everyone to report the potential improvements of this assignment with a bonus of up to 5 points.

1. **Convolution operation (15 points):**

   Convolution is one of the most common operations used in both deep learning and image processing. In this section, you are required to implement 2D convolution in two different ways. You will be able to examine your convolution operator in the Gaussian filter and Sobel filter. Please complete the *HM1_convolve.py*.

   a)[5 points] Implementing Padding Function

   You are required to implement a padding function that takes inputs a 2D image, a padding size (the number of circles around the image), and a padding mode (zero padding or replication padding). Here replication padding refers to pad the input tensor using the replication of the input boundary (see Appendix 2).

   b)[5 points] Implementing 2D Convolution via Toeplitz Matrix

   The convolution can be implemented as a single matrix multiplication, which is called Toeplitz matrix (see Appendix 1). For a 6×6 input 2D array and a 3×3 filter, you are requested to construct the Toeplitz matrix and then perform the convolution on the 2D array using your Toeplitz matrix. Here we assume that the convolution uses zero padding so that the output size remains the same.

Hint: you will need to construct a 36×36 *Toeplitz matrix if your input array is not padded or a* 36×64 *Toeplitz matrix if you pad the input first. You can avoid using for loops since Numpy allows you to index many matrix elements at the same time and assign values to them together.*

c)[5 points] Convolution by Sliding Window

Convolution implemented by Toeplitz matrix is fully parallel, but it requires you to construct a highly sparse Toeplitz matrix (nearly $N^2 \times N^2$) for an image with $N \times N$ size. An alternative for implementing convolution is through sliding window, which allows you to simply compute the dot product between your image values and the kernel inside each window. In this question, you are requested to implement a convolution operator that can take an 2D image with an arbitrary shape as input and perform convolution with 3×3 kernel. You can assume no padding when implementing this convolution operator.

Hint: to avoid using any loop operations, you can use np.meshgrid to generate indices.

d)[0 points] Gaussian Filter

With the sliding-window convolution, you can simply construct a Gaussian filter. In the code, we will use your implemented padding function in replication mode in a) and your implemented sliding-window convolution in c) to construct a Gaussian filter.

e)[0 points] Sobel Filter

In Lecture 2, we talked about using the finite difference to approximate image gradient. A more commonly used way is actually to perform Sobel filter (see Appendix 3). In the code, we will use your implemented padding function with reflection mode in a) and the sliding-window convolution function in c) to construct a Sobel filter.

2. **Canny Edge Detector (20 points):**

In this question, you will build your own Canny edge detector. Specifically, there are three key functions to be implemented, image gradient computing, non-maximal suppression and edge linking (with hysteresis). Please complete the *HM1_Canny.py*, and you will see a similar result as Fig. 1.

Figure 1: Canny edge detector with Lenna

a)[5 points] Compute the Image Gradient.

We will use Sobel filter to approximate the image gradient – x-derivative $\frac{\partial I}{\partial x}$ and y-derivative $\frac{\partial I}{\partial y}$. Then, we can obtain the magnitude $\mathbf{M}$ and the orientation $\mathbf{D}$ of the gradient as:

$$\begin{aligned} \mathbf{M} &= \sqrt{\left(\frac{\partial I}{\partial x}\right)^2 + \left(\frac{\partial I}{\partial y}\right)^2}, \\ \mathbf{D} &= \arctan\left(\frac{\partial I}{\partial y} \cdot \frac{\partial x}{\partial I}\right). \end{aligned} \tag{1}$$

The magnitude and direction give a rough estimation about the edges, including the possibility of a pixel lying in the edge (magnitude) and the normal of the edge (direction). You will complete the *compute_gradient_magnitude_direction* function.

b)[5 points] Non-Maximal Suppression (NMS)

After obtaining the magnitude and direction of gradient, you should check each pixel and remove any unwanted pixels which may not constitute the edge. In other words, you have to find the local maximum along the gradient direction, which could lead to "thin edges". Please complete the *non_maximal_suppressor* function.

c)[10 points] Edge Linking with Hysteresis Threshold At this stage, we have to remove the noise/fake edges which are discontinuous or have low magnitude. To keep only the high-quality edges, we have to filter out the pseudo-edges and link the potential discontinuous edges. Specifically, you are required to utilize the low/high threshold to categorize the pixels of edges. Finally, you should remove and link the pixels based on the pixel category. Please complete the *hysteresis_thresholding* function **("for loop" is allowed in this question)**.

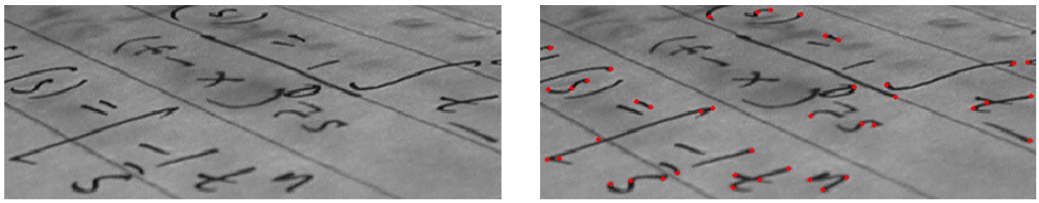

Figure 2: Harris corner detector with handwriting.png

3. **Harris Corner Detector (10 points):**

One of the key topics in computer vision is to find the salient and accurate point feature of given images. One early attempt to find these features is called Harris corner detector. In this question, you are requested to implement the corner response function to investigate the property of the intensity changes of a given window. You can simply build the corner response function within each window to estimate the θ as:

$$\theta = det(\mathbf{M}) - \alpha trace(\mathbf{M})^2. \tag{2}$$

Please complete the *HM1_HarrisCorner.py*, and you will see a similar result as Fig. 2.

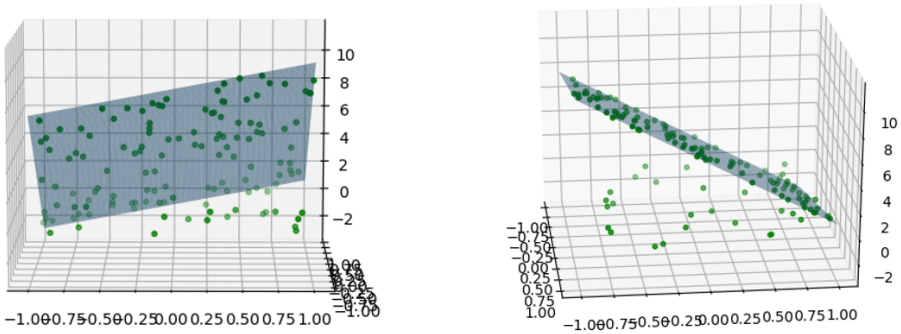

Figure 3: Plane (blue) fitting by RANSAC; Points (green)

4. **Plane Fitting using RANSAC (25 points):**

   In this question, you are expected to implement a plane fitting algorithm with RANSAC to find a plane from a noisy 3D point cloud. Specifically, you need to perform the following steps.

   1) hypothesis generation: For one hypothesis, you need to randomly select a group of seed points and then estimate the plane parameters. The number of hypotheses should be decided such that with more than 99.9% probability at least one hypothesis does not contain any outliers. You are expected to generate all hypotheses in parallel.

   2) verification: find and count the inliers for all the hypotheses.

   3) final refinement: select the hypothesis with the largest number of inliers as your best hypothesis and estimate the final plane parameters with its inliers using the least-square method, which minimizes the sum of squared perpendicular distances between the points and the plane.

   Please complete the *HM1_RANSAC.py*, and you will see a similar result as Fig. 3.

5. **Backpropagation for an MLP (30 points):**

   Backpropagation is a fundamental technique for training neural networks. In this question, you will experience how to write the backpropagation for a toy MLP to achieve binary classification. Specifically, you will have 10 images from the MNIST dataset and the MLP learns to predict whether the given image is "0" or not.

```
iteration: 1, loss: 9.779285
iteration: 2, loss: 4.440574
iteration: 3, loss: 3.978661
iteration: 4, loss: 3.626848
iteration: 5, loss: 3.319223
iteration: 6, loss: 3.042046
iteration: 7, loss: 2.792467
iteration: 8, loss: 2.569340
iteration: 9, loss: 2.370970
iteration: 10, loss: 2.194997
...
iteration: 40, loss: 0.526443
iteration: 42, loss: 0.494772
iteration: 43, loss: 0.480096
iteration: 44, loss: 0.466126
iteration: 45, loss: 0.452815
iteration: 46, loss: 0.440120
iteration: 47, loss: 0.428005
iteration: 48, loss: 0.416432
iteration: 49, loss: 0.405369
iteration: 50, loss: 0.394786
```

Figure 4: A result of our BP implemetntion. You may obtain a even better result by tuning parameters.

   We have already implemented the forward pass including the sigmoid activate function and the cross-entropy loss. Once you have figured out the workflow of the forwarding, try to implement your own backpropagation. Note that, you are only recommended to use Numpy and without using any loop operations. You can check more details in the slides of lecture 3 and Appendix 4.

Appendix

1. 1D Convolution with Toeplitz Matrix.
https://en.wikipedia.org/wiki/Toeplitz_matrix#Discrete_convolution

2. Replication padding in PyTorch.
https://pytorch.org/docs/stable/generated/torch.nn.ReplicationPad2d.html

3. OpenCV tutorial on Sobel derivative.
https://docs.opencv.org/3.4/d2/d2c/tutorial_sobel_derivatives.html

4. Backpropagation for a Linear Layer from Justin Johnson. http://cs231n.stanford.edu/handouts/linear-backprop.pdf

5. We recommend some handy Numpy functions which may help your tensor-style coding.

- meshgrid, https://numpy.org/doc/stable/reference/generated/numpy.meshgrid.html
- concatenate, https://numpy.org/doc/stable/reference/generated/numpy.arange.html
- where, https://numpy.org/doc/stable/reference/generated/numpy.where.html
- argmax, https://numpy.org/doc/stable/reference/generated/numpy.argmax.html
- linalg.svd, https://numpy.org/doc/stable/reference/generated/numpy.linalg.svd.html
- arctan2, https://numpy.org/doc/stable/reference/generated/numpy.arctan2.html