Assignment 1
Due date for the report: October 27, 2020
Due date for the implementation: November 10, 2020

Computer Vision
University of Bern
Fall 2020

Introduction

In this assignment you are asked to cast and solve a computer vision task as an energy minimization problem. It will allow you to learn about modern optimization techniques.

We consider the task of **Image Deblurring** with 4 different motion blurs. The objective is to implement an algorithm to output a sharp image (as close as possible to the original sharp one) from a blurry image.

This problem can be solved with any of the optimization **methods** that we have seen in class. In this assignment we consider only 3 such methods. The first step is to discretize the energy (the relevant lecture slides are 03-04). Then, you need to implement all of the following cases:

1. Gradient descent
2. Gradient linearization + Gauss-Seidel
3. Gradient linearization + successive over-relaxations.

Required work

- You need to solve the above task under 1 type of motion blur (out of 4) with **all three** methods.
- Your assigned visual task (i.e., which blur is applied to the sharp image) is determined by your matriculation number (see next pages).
- Implement your solution in Python. We provide a template Jupyter notebook.
- Write a report with your answers to the questions (see Assignment section).

Submission details

- **Important**: No group work allowed – it must be only your own coding and writing.
- The due date of the submission of the first half of the assignment (the report) is **October 27, 2020** and for the second half of the assignment (the code) is **November 10, 2020**.
- Submit your work through **ILIAS**.
- Submit the report with your answers in a single **PDF** file.
  The report will be accepted only in **PDF** format!
- Submit all files required to run your code in a zip file.
  - **Important**: Submit your Jupyter notebooks with all outputs displayed. To obtain credits, the code must run without crashing or producing errors and warnings.
  - Code errors will incur a penalty in the final assignment mark.

Task: Image Deblurring

Introduction Deblurring is the problem of undoing a blur artifact. For example, Fig. 1 (left) shows a blurry image captured by a camera moving horizontally and Fig. 1 (right) shows the desired output result (a sharp image) of a deblurring algorithm. When a camera captures an image, it opens the shutter for a given time interval (the exposure time). If the camera moves during this time, then multiple different views are accumulated on the camera sensor and produce a blurry image. Consider a camera that translates on the X-Y plane without rotating. in this case, the blur artifact will be the same everywhere in the image and it can be described via a convolution. Suppose that we list in a 2D grid all of the possible X-Y shifts occurring during the exposure interval. For each shift, we compute the time spent by the camera in that position. Then, we normalize each computed time by dividing it by the total exposure time. For example, if a camera sits half of the time in the center of the X-Y plane and the other half of the time 1 step to the right of the center, then the grid will have zero values at all locations except for $1/2$ at the center and $1/2$ at the element to its right. This resulting 2D grid is called the **blur kernel** $k \in R^{s\times s}$, where s is the total range of shifts. With these definitions, we can describe a blurry grayscale image $g \in R^{m\times n}$ as

$$g = u * k, \tag{1}$$

where $u \in R^{(m+s-1)\times(n+s-1)}$ is the sharp image and $*$ is the 2D convolution operator (with `valid` output size).

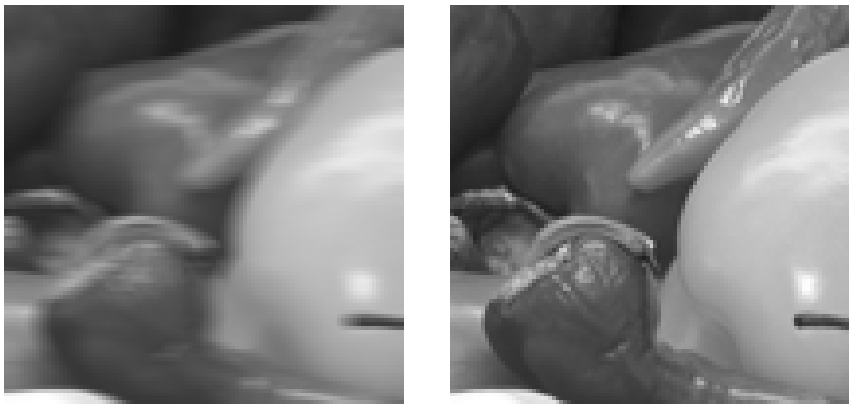

Fig. 1: The deblurring problem. Left: Input image with horizontal blur. Right: Restored sharp image (output).

Formulation The debluring problem can then be cast as the following optimization

$$\begin{aligned}\hat{u} &= \arg\min_u E[u], \\ E[u] &= |g - u * k|^2 + \lambda R[u].\end{aligned} \tag{2}$$

The first term in $E[u]$ is the *data term* and it encourages the reconstructed sharp image u to match the input blurry image g after being convolved with the blur kernel k. The second term, $\lambda R[u]$, is the *regularization term.* $\lambda \geq 0$ is the *regularization parameter.* The first term is defined as

$$|u * k - g|^2 = \sum_{i=0}^{m-1}\sum_{j=0}^{n-1}\left| g[i,j] - \sum_{p=0}^{1}\sum_{q=0}^{1} k[p,q]u[i-p+1, j-q+1]\right|_2^2, \tag{3}$$

where k is one of these 4 cases (notice that we choose $s = 2$):

- $k_0 = \begin{bmatrix} 1/2 & 1/2 \\ 0 & 0 \end{bmatrix}$,
- $k_1 = \begin{bmatrix} 1/2 & 0 \\ 1/2 & 0 \end{bmatrix}$,

- $k_2 = \begin{bmatrix} 1/2 & 0 \\ 0 & 1/2 \end{bmatrix}$,
- $k_3 = \begin{bmatrix} 0 & 1/2 \\ 1/2 & 0 \end{bmatrix}$.

For the second term we will explore three different choices:

1. no regularization ($\lambda = 0$),
2. the *Gaussian prior*, where
$$R[u] = |\nabla u|_2^2 = \sum_{i=0}^{m} \sum_{j=0}^{n} |\nabla u[i,j]|_2^2 \,, \tag{4}$$
3. the *anisotropic* total variation
$$R[u] = |\nabla u|_1 = \sum_{i=0}^{m} \sum_{j=0}^{n} |\nabla u[i,j]|_1 \,. \tag{5}$$

The discretization of eq. 4 needs some care, because of the boundary conditions. For example, when one uses forward differences to approximate the gradient, the regularization term in eq. 4 becomes

$$R[u] = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} \left(u[i+1,j] - u[i,j]\right)^2 + \left(u[i,j+1] - u[i,j]\right)^2 + \sum_{j=0}^{n-1} (u[m,j+1] - u[m,j])^2 + \sum_{i=0}^{m-1} (u[i+1,n] - u[i,n])^2. \tag{6}$$

Also in the anisotropic version, it is important to take care of the boundary conditions. We leave that derivation as part of the assignment.

Assignment

Your assigned blur kernel is k_j where j = mod(last digit of your matriculation number, 4). For example, if your matriculation number is 12-345-678, then j = mod(8, 4)=0 and your blur is k_0,

Report (due on October 27, 2020). Describe the problem formulation in detail and derive your personal implementation (write the analytic formulas) for the following two aspects: **[Total 40 Points]**

1. Finite difference approximation of the objective function E: Choose forward differences for the discretization. In particular, use eq. (6) for the Gaussian prior implementation and derive the corresponding discretization for the anisotropic prior. Write the main steps of your calculations in the report. **[10 Points]**
2. Calculation of the exact gradient of the discretized E. **[30 Points]**
   - Compute the gradient $\nabla_u E$ of the objective function for all three choices of regularization ($\lambda = 0$, Gaussian prior, anisotropic TV). Choose forward differences for the discretization of the gradient.
   - Write the main steps of your calculations in the report and then the final formulas (that is, show how you obtained your formulas)
   - You must handle all the **special cases at the boundaries**, or otherwise you will have serious artifacts in your reconstructions (and this will incur in a score penalty).

   Hint: We highly recommend to use forward differences in your calculations.

Implementation (due on November 10, 2020). Write the python code (with comments) to implement the following three numerical methods: **[Total 60 Points]**

1. **Gradient descent** **[15 Points]**
   Implement the **gradient descent method** for your assigned blur kernel in Python. The function declaration must match the following:

```
def GD(g, lambda):
    """
    g: grayscale blurry image of size (M, N)
    lambda: regularization parameter

    :returns u: sharp image of size (M+1, N+1)
    """
    returns u
```

   Add comments in your code to explain the most important parts of your algorithm.

   (a) First, include only the gradient of the data term ($\lambda = 0$). Document your observation in the report.

   (b) Try gradient descent with the two regularization terms defined above (Gaussian prior, anisotropic TV). Choose λ in the range $[0.001, 0.1]$ Which of the three versions yields the best result?

2. **Linearization + Gauss-Seidel** **[15 Points]**
   Linearize the gradient and then use Gauss-Seidel to solve the linear system iteratively. **Here we will only consider the case for the Gaussian prior regularization** in eq. 4. Implement the code for your assigned blur kernel in Python. The function declaration must match the following:

```
def LGS(g, lambda):
    """
    g: grayscale blurry image of size (M, N)
    lambda: regularization parameter

    :returns u: sharp image of size (M+1, N+1)
    """
    return u
```

   Add comments in your code to explain the most important parts of your algorithm.

   Hint: We provide the implementation for the Hessian matrix.

3. **Linearization + SOR** **[15 Points]**
   Linearize the gradient and then use SOR to solve the linear system iteratively. **Here we will only consider the case for the Gaussian prior regularization** in eq. 4. Implement the code for your assigned blur kernel in Python. The function declaration must match the following:

```
def LSOR(g, lambda, w):
    """
    g: grayscale blurry image of size (M, N)
    lambda: regularization parameter
    w: SOR parameter

    :returns u: sharp image of size (M+1, N+1)
    """
    return u
```

   Add comments in your code to explain the most important parts of your algorithm. Use $w = 1.5$.

   Hint: We provide the implementation for the Hessian matrix.

Hint: Have a look at the `scipy.sparse` package for splitting sparse matrices in the SOR and Gauss-Seidel methods: `triu()`, `tril()`, `diags()`.

Code for all three algorithms must be submitted in the form of a Jupyter notebook. In addition, in your Jupyter notebook you need to provide the following: **[15 Points]**

1. Choose one of the above 3 solvers and show reconstructed images obtained in 3 cases: by very high, very low and a "reasonable" λ (a reasonable λ is one that yields a realistic sharp image – which you can check through visual inspection). Explain what the effect of λ is.
2. Let us denote with $\hat{u}_\lambda$ the solution obtained with a given λ. Since we synthetically generate the data, we can compare the solution $\hat{u}_\lambda$ to the original sharp image u. Choose one of the above 3 solvers and find manually a good guess of the optimal λ that minimizes the Sum of Squared Distances (SSD)

$$\text{SSD}(\lambda) = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} (\hat{u}_\lambda[i,j] - u[i,j])^2 \tag{7}$$

   by trying a few λ values (at least 3). Plot the SSD for the computed λ values (show the SSD values in the ordinate axis and the chosen λs in the abscissa axis). Describe (in the code) the range of λ values, where the optimal value might be and justify your answer.