Computer Vision Assignment # 2
Universität Bern

Introduction

In this assignment, you need to upload a zip file to ILIAS which includes: 1) A Jupyter Notebook file Assignment2.ipynb that contains your python code and 2) a Jupyter Notebook exported to HTML (File / Export Notebook as / HTML) and 3) a report in PDF format with your answers to the questions in the exercises section. The zip file name must be FirstName LastName.zip. If your implementation requires auxiliary functions, you must implement those functions inside a corresponding .py file. Please state your name at the beginning of the notebook.
Important: No group work allowed – must be your own coding and writing.
Contact: hamadi.chihaoui@unibe.ch
Due date: 18/11/2024

Notes on code and submission quality

In addition to answering the different questions, you are also expected to provide well written submissions. Here are some recommendations to take into consideration.

- Please answer the question in the same order as in the assignment and use the same question numbers.
- Make sure the right execution order of the notebook cells is from top to bottom. A TA should be able to reproduce your results by simply clicking "Run All" without having to guess which cells should be executed first.
- Poorly written submissions might result in points deduction.

Image Inpainting

Introduction Inpainting is the problem of filling in regions of an input image g where data is missing. For example, in Fig. 1 (left) the texture is masked by black text. Suppose that we are given a function Ω that defines where pixels are missing as follows

$$\begin{cases} \Omega[i,j] = 0 & \text{if the } (i,j) \text{ pixel is missing} \\ \Omega[i,j] = 1 & \text{otherwise} \end{cases} \tag{1}$$

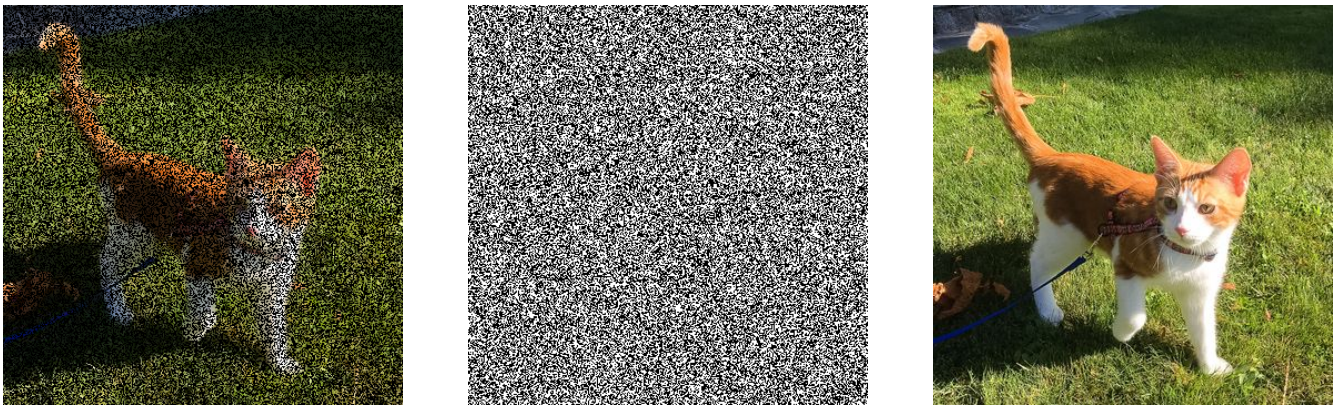

Fig. 1: Inpainting problem. Left: input image with missing pixels (at the black text). Center: mask Ω defining missing pixels. Right: inpainted image.

Formulation The inpainting problem can be cast as the following optimization

$$\begin{aligned} \hat{u}_C &= \arg\min_{u_C} E[u_C], \qquad C \in \{R, G, B\} \\ E[u_C] &= |u_C - g_C|^2_\Omega + \lambda |\nabla u_C|^2_{2,1}. \end{aligned} \tag{2}$$

The first term in $E[u_C]$ is the *data term* and it encourages the reconstructed image u_C to match the input image g_C in Ω. The second term is the *regularization term* (*total variation*). $\lambda > 0$ is the *regularization parameter*. We solve this optimization for all color channels C separately. The first term is defined as

$$|u_C - g|^2_\Omega = \sum_{i,j} \Omega[i,j](u_C[i,j] - g_C[i,j])^2, \qquad C \in \{R, G, B\}. \tag{3}$$

The second term (total variation) is defined as

$$|\nabla u_C|^2_{2,1} = \sum_{i,j} |\nabla u_C[i,j]|^2_2, \qquad C \in \{R, G, B\}. \tag{4}$$

Hint: In case of forward difference, $|\nabla u_C[i,j]|^2_2 = (u_C[i+1,j] - u_C[i,j])^2 + (u_C[i,j+1] - u_C[i,j])^2$

Optimization techniques and questions for the report

1. **[10 pts]** Finite difference approximation of the objective function E.
   - Choose forward differences for the discretization.
   - Write the main steps of your calculations in the report.
2. **[40 pts]** Calculation of the exact gradient of the discretized E.
   - Compute the gradient $\nabla_u E$ of the objective function. The gradient $\nabla_u E$ is the same for every channel $C \in \{R, G, B\}$, so you need to compute it only once. You must handle all the special cases at the boundaries.
   - Write the main steps of your calculations in the report.
3. Solvers
   (a) **[15 pts] Gradient descent**
   Implement the **gradient descent method** for your assigned visual task in Python. The function declaration must match the following:

```
def GD(g, omega, lmbda):
    """
    g: color image of size (M, N, 3)
    omega: mask of size (M, N)
    lmbda: regularization parameter

    :returns u: inpainted image of size (M, N, 3)
    """
    return g
```

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

   (b) **[25 pts] Linearization + Gauss-Seidel**
   Linearize the gradient and then use Gauss-Seidel to solve the linear system iteratively. We provide a numerical approximation of the Hessian matrix (see `hessian_matrix.py`). Implement the code in Python. The function declaration must match the following:

```
def LGS(g, omega, lmbda):
    """
    g: color image of size (M, N, 3)
    omega: mask of size (M, N)
    lmbda: regularization parameter

    :returns u: inpainted image of size (M, N, 3)

    """
    return g
```

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

   Hint: Have a look at the `scipy.sparse` package for splitting sparse matrices in the Gauss-Seidel method: `triu()`, `tril()`, `diags()`.
4. **[10 pts]** For every solver, show reconstructed images obtained 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.