Assignment 1
Due date: November 12, 2019

Computer Vision
University of Bern
2019

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 and build experience on modern optimization techniques.

We consider two different **tasks**:

A. Inpainting

B. Image blending

Each 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 (see the slides of lecture 03-04 to see how to do this). 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 only **one** of the two visual tasks, but with **all three** methods.
- Your assigned visual task will be picked randomly. A list will be posted on ILIAS.
- Implement your solution in Python. We provide a template in form of Jupyter notebooks.
- Write a report with your answers to the questions in the exercises section.

Submission details

- **Important**: No group work allowed – must be your own coding and writing.
- The due date of the submission is **November 12, 2019**.
- 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. The code must run without crashing or producing errors and warnings.
  - Code errors will incur a penalty in the final assignment mark.

Visual task (A) 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

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

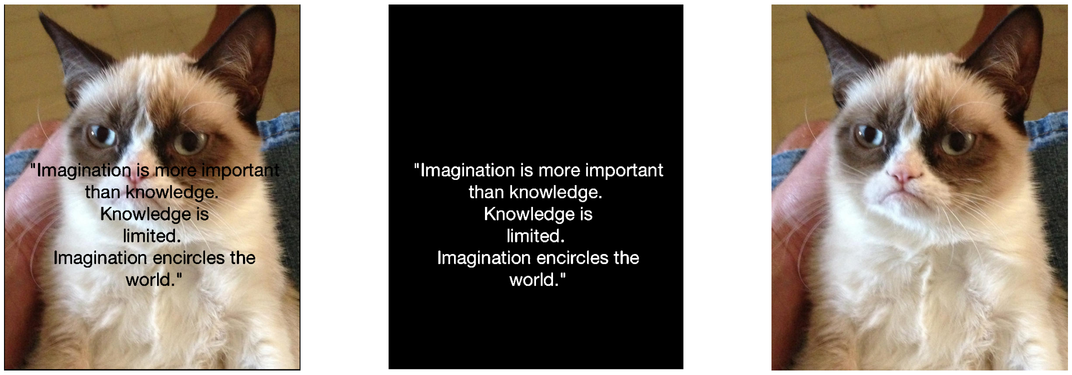
"Imagination is more important
than knowledge.
Knowledge is
limited.
Imagination encircles the
world."
"Imagination is more important
than knowledge.
Knowledge is
limited.
Imagination encircles the
world."

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

$$|u_C - g|^2_\Omega = \sum_{i,j} \Omega[i,j] |u_C[i,j] - g_C[i,j]|^2_2, \qquad C \in \{R,G,B\}. \tag{3}$$

The second term (total variation) is defined as

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

Visual task (B) Image blending

Introduction Image blending enables advanced editing of images. We would like to remove some texture from a *background* image b (see Fig. 2 left) and substitute it with texture from a *foreground* image f (see Fig. 2 middle). A function Ω defines which region in the background image will be substituted (see Fig. 2 right) as follows

$$\begin{cases} \Omega[i,j] = 1 & \text{if color at } (i,j) \text{ pixel should come from } f \\ \Omega[i,j] = 0 & \text{otherwise.} \end{cases} \tag{5}$$

As shown in Fig. 3, image blending consists of composing the two images b and f in a seamless way.

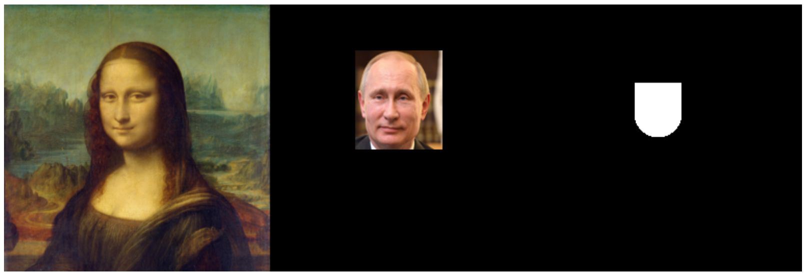

Fig. 2: Left: background image. Middle: foreground image. Right: mask.

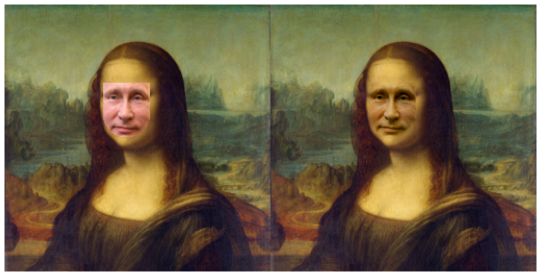

Fig. 3: Image blending. Left: naïve blending. Right: smooth blending.

Formulation The image blending task can be cast as the following optimization

$$\hat{u}_C = \arg\min_{u_C} E[u_C], \qquad C \in \{R, G, B\}$$
$$E[u_C] = |u_C - b_C|^2_{\Omega\backslash} + \lambda |\nabla u_C - \nabla f_C|^2_{\Omega} \tag{6}$$

where the first term in $E[u_C]$ encourages u_C to match the background image b in the complementary region $\Omega\backslash[i,j] = 1 - \Omega[i,j]$. The second term encourages the gradients of u_C to match those of the foreground image f in Ω. $\lambda > 0$ is the *regularization parameter*. We solve this optimization for all color channels C separately. The first term in E is defined as

$$|u_C - b_C|^2_{\Omega\backslash} = \sum_{i,j} (1 - \Omega[i,j]) |u_C[i,j] - b_C[i,j]|^2_2, \qquad C \in \{R, G, B\}. \tag{7}$$

The second term is defined as

$$|\nabla u_C - \nabla f_C|^2_{\Omega} = \sum_{i,j} \Omega[i,j] |\nabla u_C[i,j] - \nabla f_C[i,j]|^2_2, \qquad C \in \{R, G, B\}. \tag{8}$$

Optimization techniques and questions for the report

1. Finite difference approximation of the objective function E.
   - Clearly state the explicit discretization choices you make (forward/backward difference etc.)
   - Write the main steps of your calculations in the report
2. Calculation of the exact gradient of the discretized E.
   - Compute the gradient $\nabla_u E$ of the objective function (for each color channel).
   - Write the main steps of your calculations in the report
3. Solvers

   (a) **Gradient descent**
   Implement the **gradient descent method** for your assigned visual task in Python. The function declaration must match one of the following:

```
def GD(f, b, omega, lmbda):
    """
    b: backgound color image of size (M, N, 3)
    f: foreground color image of size (M, N, 3)
    omega: foreground mask of size (M, N)
    lmbda: parameter

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

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) **Linearization + Gauss-Seidel**
   Linearize the gradient and then use Gauss-Seidel to solve the linear system iteratively. Implement the code for your assigned visual task in Python. The function declaration must match one of the following:

```
def LGS(f, b, omega, lmbda):
    """
    b: backgound color image of size (M, N, 3)
    f: foreground color image of size (M, N, 3)
    omega: foreground mask of size (M, N)
    lmbda: parameter

    :returns u: blended image of size (M, N, 3)
    """
    return b

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.

(c) **Linearization + SOR**
Linearize the gradient and then use SOR to solve the linear system iteratively. Implement the code for your assigned visual task in Python. The function declaration must match one of the following:

```
def LSOR(f, b, omega, lmbda):
    """
    b: backgound color image of size (M, N, 3)
    f: foreground color image of size (M, N, 3)
    omega: foreground mask of size (M, N)
    lmbda: parameter

    :returns u: blended image of size (M, N, 3)
    """
    return b

def LSOR(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 SOR and Gauss-Seidel methods: `triu()`, `tril()`, `diags()`.

4. Choose one of the above 3 solvers and show images obtained by very high, very low and a reasonable λ (you can find a reasonable λ by looking at the solution). Explain what the effect of λ is.

5. **Inpainting:** Let us denote with $\hat{u}_\lambda$ the solution obtained for all 3 channels with a given λ. Since we synthetically generate the data, we can compare the solution $\hat{u}_\lambda$ to the original image u. Choose one of the above 3 solvers and find the optimal λ that minimizes the sum of squared distances (SSD)

$$SSD(\lambda) = \sum_{i,j} (\hat{u}_\lambda[i,j] - u[i,j])^2.$$

Add a figure in the report with the graph of the SSD (ordinate axis) vs. λ (abscissa axis).

Image blending: Choose one of the above 3 solvers and create three image compositions. Use your own photos or images from the internet. Show the foreground, background, mask and final result.