Image inpainting – Assignment 1

Michèle Wyss
10-104-123

Motivation

Sometimes it's a bad thing that some objects appear in a photograph, e.g. as a tourist I'd like to make a picture of a place but some people are standing around and occlude parts of the actual target image. A simple idea is to just remove the persons after the photograph has been taken. This is a typical example of where an image with holes and invalid regions appears. Image inpainting is one possible approach to generate sensible content for those regions that were occluded before.

Problem

Inpainting is an image processing method that aims for reconstruction of lost or corrupted parts of an image. The goal is to get a plausible-looking image without undefined or invalid pieces. Mathematically, the problem can be formulated by defining a cost function that has to be minimized.

Cost function

We formulate the following cost function to describe the problem:

$$E(u) = \frac{\lambda}{2}\|u - g\|_{\Omega}^2 + \|\nabla u\|_2.$$

Here, the value u that we optimize for is an image with sensible content, g is the input image containing undefined pieces, λ is a regularization parameter to control the tradeoff between exactness of fitting and smoothness of the result, and Ω is a mask that defines the missing data. The goal is to find the argument u that minimizes E, thus

$$\arg\min_u E(u).$$

Minimization – Gradient descent

To minimize the objective function E, a simple iterative approach called *gradient descent* is applied. This method proceeds as follows: Starting from an (randomly chosen) initial point, the following update rule is performed:

$$\theta_j := \theta_j - \alpha \cdot \frac{\partial}{\partial \theta_j} E(\theta), \tag{1}$$

where θ_j are the unknown parameters of the model, α is the *learning rate* and $E(\theta)$ is a cost function that is aimed to be minimized, normally called the *objective function*.

In the case of our above formulated objective function, the update rule will look as follows:

$$u[i,j] := u[i,j] - \alpha \cdot \frac{\partial}{\partial u[i,j]} E(u).$$

Derivation of gradient

Clearly, we need to know the partial derivative $\frac{\partial}{\partial u[i,j]} E(u)$ to apply gradient descent. It is given by

$$\begin{aligned}
\frac{\partial}{\partial u[i,j]}\left(\frac{\lambda}{2}\|u-g\|_\Omega^2+\|\nabla u\|_2\right) &= \frac{\partial}{\partial u[i,j]}\left(\frac{\lambda}{2}\left(\sum_{i,j}\Omega[i,j]\cdot(u[i,j]-g[i,j])^2\right)+\|\nabla u\|_2\right)\\
&= \frac{\lambda}{2}\cdot\frac{\partial}{\partial u[i,j]}\left(\sum_{i,j}\Omega[i,j]\cdot(u[i,j]-g[i,j])^2\right)+\frac{\partial}{\partial u[i,j]}\|\nabla u\|_2\\
&= \frac{\lambda}{2}\cdot\frac{\partial}{\partial u[i,j]}\Omega[i,j]\cdot(u[i,j]-g[i,j])^2+\frac{\partial}{\partial u[i,j]}\|\nabla u\|_2\\
&= \lambda\cdot\Omega[i,j]\cdot(u[i,j]-g[i,j])+\frac{\partial}{\partial u[i,j]}\|\nabla u\|_2.
\end{aligned}$$

The target image u is descrete and 2-dimensional. Using the forward difference scheme, we can write $\|\nabla u\|_2$ as

$$\|\nabla u\|_2=\sum_{i,j}\underbrace{\sqrt{(u[i+1,j]-u[i,j])^2+(u[i,j+1]-u[i,j])^2}}_{\tau[i,j]}. \tag{2}$$

Because for fixed i and j most of the summation terms in Equation (2) are constant, the exact derivative $\frac{\partial}{\partial u[i,j]}\|\nabla u\|_2$ of the discretized energy reduces to

$$\frac{\partial}{\partial u[i,j]}\|\nabla u\|_2=\frac{\partial\tau[i,j]}{\partial u[i,j]}+\frac{\partial\tau[i-1,j]}{\partial u[i,j]}+\frac{\partial\tau[i,j-1]}{\partial u[i,j]}.$$

There are now 3 derivative terms that are left to compute. This can be done straight forward:

$$
\begin{aligned}
\frac{\partial \tau[i,j]}{\partial u[i,j]} &= \frac{\partial}{\partial u[i,j]} \sqrt{(u[i+1,j]-u[i,j])^2 + (u[i,j+1]-u[i,j])^2} \\
&= \frac{1}{2} \cdot \frac{1}{\tau[i,j]} \cdot ((-2) \cdot (u[i+1,j]-u[i,j]) - 2(u[i,j+1]-u[i,j])) \\
&= \frac{-((u[i+1,j]-u[i,j]) + (u[i,j+1]-u[i,j]))}{\tau[i,j]} \\
&= \frac{2u[i,j]-u[i+1,j]-u[i,j+1]}{\tau[i,j]}, \\
\frac{\partial \tau[i-1,j]}{\partial u[i,j]} &= \frac{\partial}{\partial u[i,j]} \sqrt{(u[i,j]-u[i-1,j])^2 + (u[i-1,j+1]-u[i-1,j])^2} \\
&= \frac{1}{2} \cdot \frac{1}{\tau[i-1,j]} \cdot (2 \cdot (u[i,j]-u[i-1,j])) \\
&= \frac{u[i,j]-u[i-1,j]}{\tau[i-1,j]}, \\
\frac{\partial \tau[i,j-1]}{\partial u[i,j]} &= \frac{\partial}{\partial u[i,j]} \sqrt{(u[i+1,j-1]-u[i,j-1])^2 + (u[i,j]-u[i,j-1])^2} \\
&= \frac{1}{2} \cdot \frac{1}{\tau[i,j-1]} \cdot 2(u[i,j]-u[i,j-1]) \\
&= \frac{u[i,j]-u[i,j-1]}{\tau[i,j-1]}.
\end{aligned}
$$

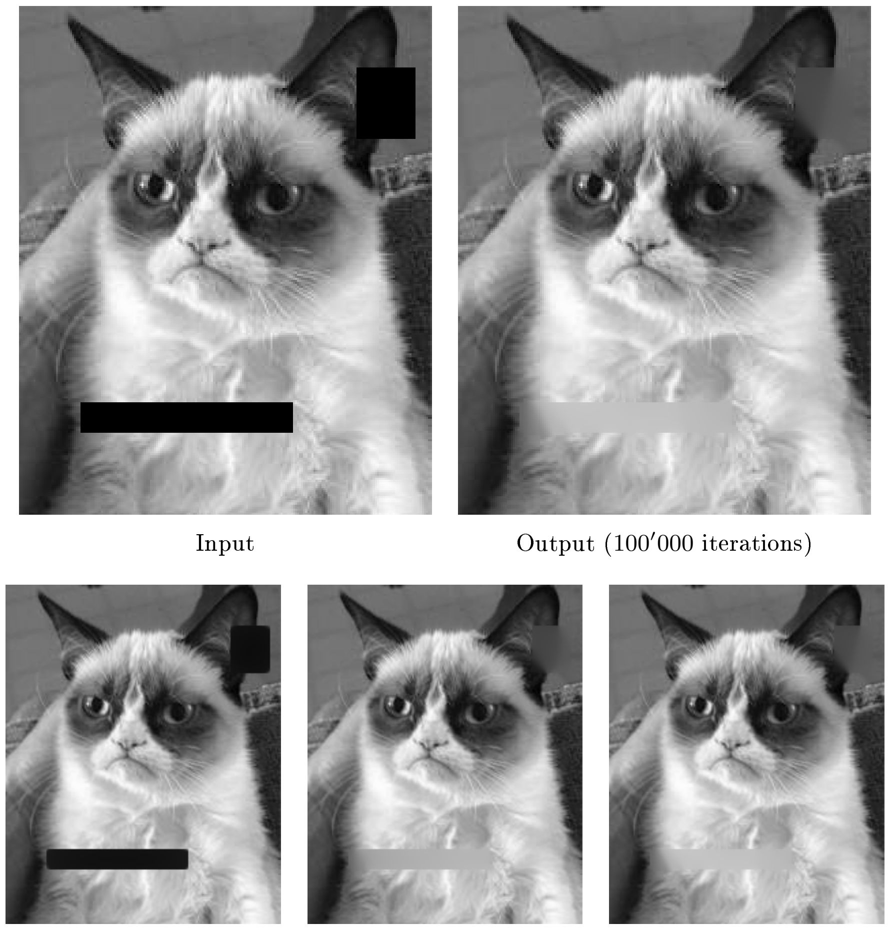
All in all we have therefore the following update rule:

$$
\begin{aligned}
u[i,j] := u[i,j] - \alpha \cdot \Bigg(\lambda\Omega[i,j] \cdot (u[i,j]-g[i,j]) &+ \frac{2u[i,j]-u[i+1,j]-u[i,j+1]}{\tau[i,j]} \\
&+ \frac{u[i,j]-u[i-1,j]}{\tau[i-1,j]} \\
&+ \frac{u[i,j]-u[i,j-1]}{\tau[i,j-1]} \Bigg)
\end{aligned}
$$

Implement gradient descent for inpainting

I implemented inpainting using the mentioned gradient descent method. The gradients were computed as derived above. A quite small learning rate of $\alpha = 0.0005$ ensures that the iterative approach converges – however, this also leads to the need of many iterations to get an acceptable result (Figures 1 and 2).

Therefore some of the later shown examples are computed with larger learning rates.

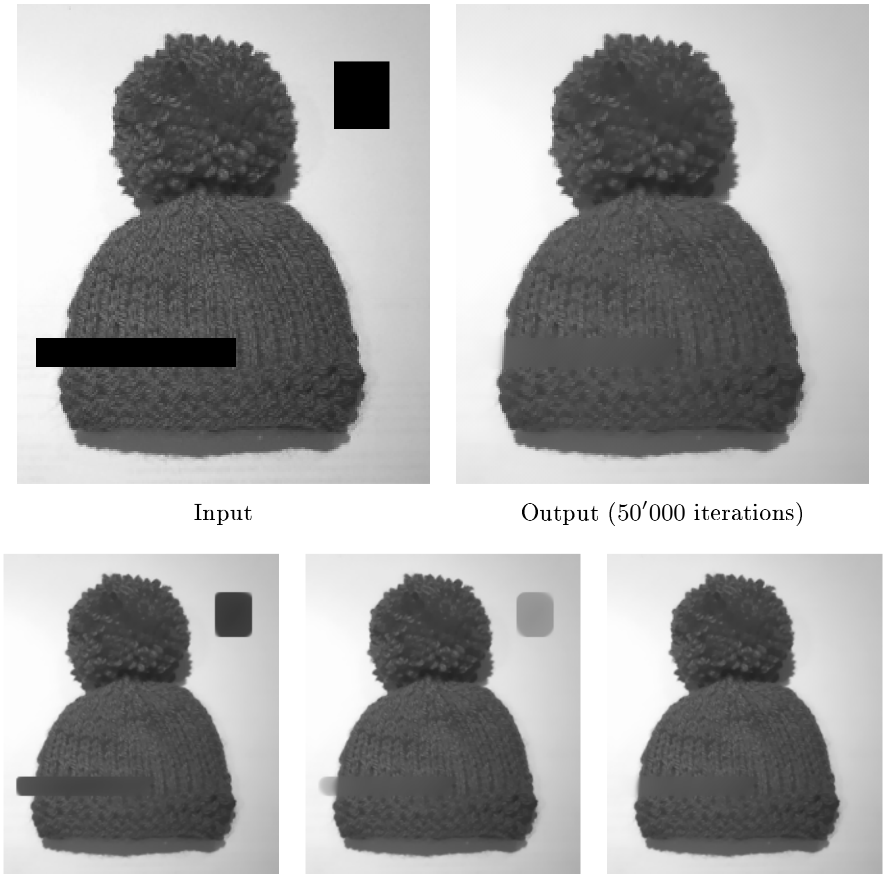


Input — Output (100′000 iterations)

1000 iterations — 10′000 iterations — 50′000 iterations

Fig. 1: "Grumpy cat" – different stages of the minimization using gradient descent with a learning rate of $\alpha = 0.0005$

Another example:



Input — Output ($50'000$ iterations)

$2'000$ iterations — $5'000$ iterations — $10'000$ iterations

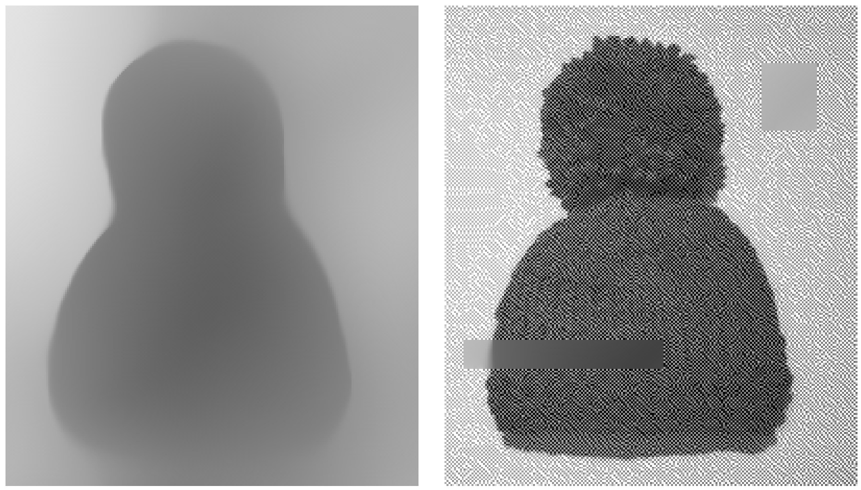
Fig. 2: "Wooly hat" – different stages of the minimization using gradient descent with a learning rate of $\alpha = 0.0005$

The effect of λ

In the objective function

$$E(u) = \frac{\lambda}{2}\|u - g\|_\Omega^2 + \|\nabla u\|_2$$

the parameter λ is a *regularization parameter*. It controls the tradeoff between the two constraints of fitting the image and keeping the result smooth. A very low λ, e.g. $\lambda = 0.1$ leads to an oversmoothed image whereas a very large value for λ may possibly fail to reach a nice-looking (smooth) image (see Figure 3). Moreover, a large λ-value generally requires a smaller learning rate α, which can be seen directly by considering the computed final update rule: If λ is very large, the first term gets a high weight and may cause the gradient descent method to diverge. Reasonable results can be obtained with λ-values that lie in between (see Figure 4).



$\lambda = 0.1$, $10'000$ iterations $\quad$ $\lambda = 1'990$, $10'000$ iterations

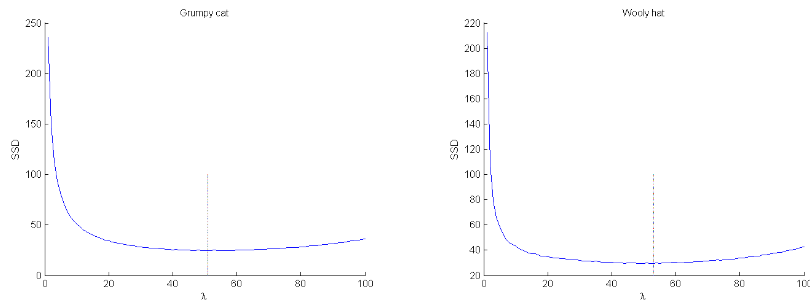
Fig. 3: Results obtained with extreme values for λ. Left: low λ; the result is oversmoothed, Right: high λ; too much structured

$\lambda = 200$ $\quad$ $\lambda = 500$ $\quad$ $\lambda = 850$

Fig. 4: Better results obtained with reasonable values for λ.

The optimal λ

Performing 1'000 iterations and using a learning rate of $\alpha = 0.01$, we have the following optimal values for λ: In the case of the wooly hat the optimal value was $\lambda = 53$, in the grumpy cat example it was $\lambda = 51$ which can be seen from the plots in Figure 5. The vertical axis shows the summed squared distance between the ground truth image and the obtained results, on the horizontal axis is λ.



Grumpy cat Wooly hat

Fig. 5: λ vs. SSD plot. The vertical line denotes the optimal value for λ.

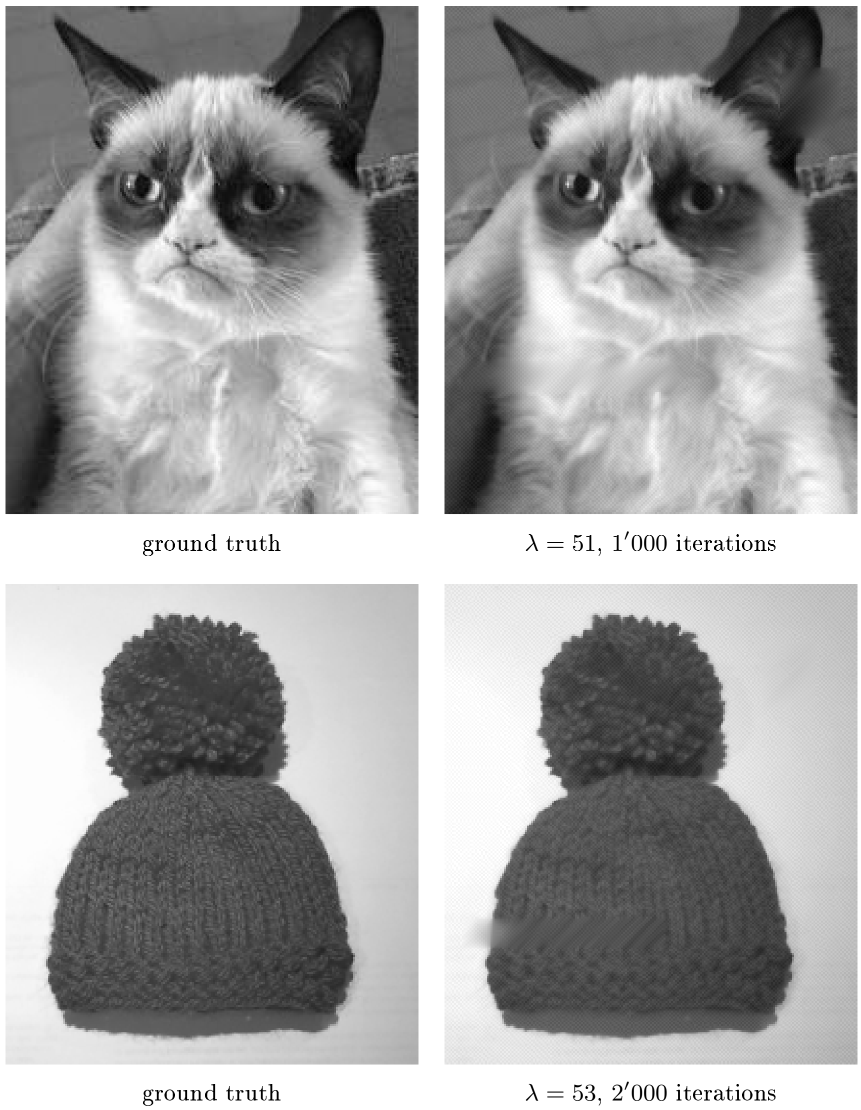


Fig. 6: Top row: The optimal λ leads to reasonable results. Bottom row: The optimal λ does not lead to satisfying quality of the resulting image. Choosing a lower learning rate leads to better results. This will, however, also lead to another optimal value for λ.