C2 – Optimisation in Computer Vision
Part 1 –Inpainting

Team nr. 4

Members:
Miruna-Diana Jarda
Gunjan Paul
Diana Tat

Image Inpainting

•Inpainting is a process of filling in missing or damaged parts of an image with plausible content based on the surrounding information.

•This technique is often used in compute vision to restore or complete an image when certain areas are obscured, damaged, or missing.

Fig 1.1. Damaged image

Fig 1.2. Mask on the damaged part

Fig 1.3. Inpainted image

Tasks

1. Explanation of the problem
2. Explanation of the solution
3. A few slides with the main parts of the code, with simple explanations
4. Results (including with your own examples, be creative)
5. Discussion & conclusions

Task 1: Explanation of the problem

We have an image U=(A,B) and we produce a new image V

Criteria: In A, the inpainted image should be the same in V as in U

In B, the inpainted image should be smooth

$V(x,y) = U(x,y)$ at each (x,y) in A

$4V(x,y)-(V(x-1,y)+V(x-1,y)+V(x,y-1)+V(x,y+1))=0$ at each (x,y) in B

Task 2: Explanation of the solution Mathematically approach

- We divide the image in two parts.
- Region A: The known region where no inpainting is needed.
- Region B: The region which needs to be inpainted.

- For the pixels at the edge of an image, they don't have four neighbours in all directions because they're on the boundary. To work with these boundary pixels, we can use a technique called padding.
- Padding involves adding extra, virtual pixels around the edges of the image. These virtual pixels are sometimes called "ghost cells." Think of them as creating a one-pixel-wide frame around the image.

Fig 2. Image U splitted in 2 regions

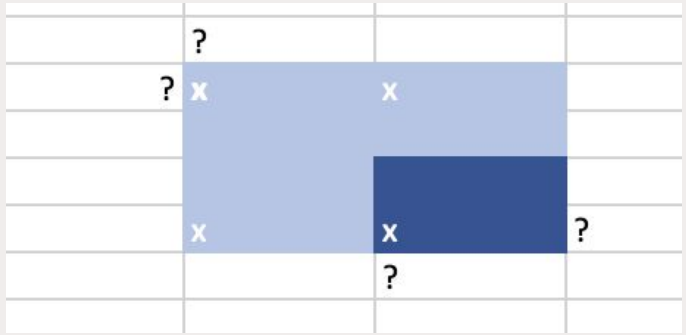

Fig 3. Boundary pixels

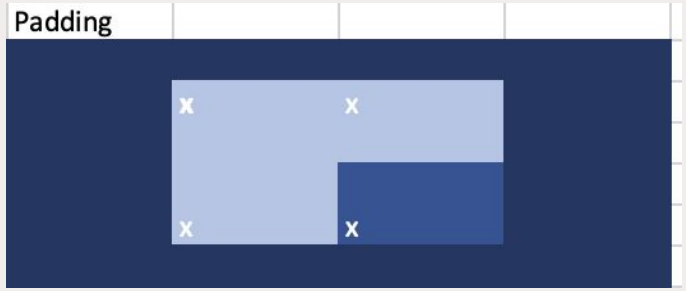

Fig 4. Padded image

•If we assume as an input image U(x, y), the inpainted image V(x,y) can be obtained in the following way:

•1. In region A: $U_{i,j} = V_{i,j}$

•2. In region B: $4 * V_{i,j} - (V_{i+1,j} + V_{i-1,j} + V_{i,j+1} + V_{i,j-1}) = 0$

Because we added the padding, our image is now of size (ni+2, nj+2) and the boundary conditions are:

•North boundary condition: $U_{(i,j)} - U_{(i-1,j)} = 0$

•South boundary condition: $U_{(ni+1,j)} - U_{(ni,j)} = 0$

•West boundary condition: $U_{(i,1)} = U_{(j,2)}$

•East boundary condition: $U_{(i,nj+1)} = U_{(i,nj)}$

x	1	x		
1	-4	1		
x	1	x		
		pixel needed to be inpainted		

Fig 5. Example an inapinting situation

Assembling the matrix A

For inner pixels where inpainting is required:
The diagonal (corresponding to the pixel itself) has a value of 4.
The immediate neighbours (boundaries) have a value of -1.
For inner pixels outside the inpainting region the diagonal has a value of 1.

$$A = \begin{pmatrix} 4 & -1 & 0 & \dots & -1 \\ -1 & 4 & -1 & \dots & 0 \\ 0 & -1 & 4 & \cdots & 0 \\ \dots & \dots & \dots & \cdots & \dots \\ -1 & 0 & 0 & \cdots & 4 \end{pmatrix} * V = \begin{pmatrix} v1,1 \\ v2,1 \\ v3,1 \\ \dots \\ vni-2,1 \\ v1,2 \\ v2,2 \\ v3,2 \\ \dots \\ vni-2,2 \\ \dots \\ vni-2,nj2 \end{pmatrix} = \begin{pmatrix} 0 \\ \dots \\ ui,j \\ \dots \\ 0 \\ \dots \\ 0 \end{pmatrix}$$

3. A few slides with the main parts of the code

```
# South side boundary conditions

# This sets i to the last row of the padded image (with ghost boundary),
# which corresponds to the south boundary
i = ni + 2

# we are iterating over each column j of the padded image,
# covering the entirety of the south boundary
for j in range(1, nj + 3, 1):
    p = (j - 1) * (ni + 2) + i

    # u(i,j) is influenced by u(i,j) own and above one u(i-1,j)

    # so on for row, we can say both are one column
    idx_Ai.extend([p, p])
    idx_Aj.extend([p, p - 1])
    # 1 for u(i,j) own and -1 for u(i-1,j) because
    # u(i,j)-u(i-1,j)=0 (boundary condition)
    a_ij.extend([1, -1])
    b[p - 1] = 0
```

Fig 6. South side boundary code

- The south side boundary represents the last row of the padded image. In order to compute each pixel belonging to this boundary, we store 1 for the pixel itself and -1 for the pixel on the same column and row above as coefficients at at their corresponding indices in the sparse matrix (here represented by idx_Ai and idx_Aj).
- The value of all boundary pixels in vector b is set 0.

```
# East side boundary conditions
j = nj + 2
for i in range(1, ni + 3, 1):
    p = (j - 1) * (ni + 2) + i

    idx_Ai.extend([p, p])
    idx_Aj.extend([p, p - ni - 2])
    a_ij.extend([1, -1])
    b[p - 1] = 0
```

```
# West side boundary conditions
j = 1
for i in range(1, ni + 3, 1):
    p = (j - 1) * (ni + 2) + i

    idx_Ai.extend([p, p])
    idx_Aj.extend([p, p + ni + 2])
    a_ij.extend([1, -1])
    b[p - 1] = 0
```

Fig 7. East and West boundary code

- The west (left) side and east (right) side boundaries are similarly computed by adding the coefficients 1 for itself and -1 for the pixel on its right, respectively on its left to the list of the coefficients of the sparse matrix.

```
# Inner points
### we are iterating over image pixel values(excluding ghost boundary)
for j in range(2, nj + 2, 1):
    for i in range(2, ni + 2, 1):
        p = (j - 1) * (ni + 2) + i
        # checks if the current pixel (i,j) is within B region to be inpainted
        if dom2Inp_ext[i - 1, j - 1] == 1:
            # For the Laplace equation, we can use a five-point stencil
            idx_Ai.extend([p, p, p, p, p])
            idx_Aj.extend([p, p + 1, p - 1, p + ni + 2, p - ni - 2])

            # coefficient values of laplacian equation
            a_ij.extend([4, -1, -1, -1, -1])
            # for laplacian eqn, we have 0 on b
            b[p - 1] = 0
        else:
            # for pixel in A region
            idx_Ai.append(p)
            idx_Aj.append(p)

            # for pixel in A region, we add 1 in matrix A
            a_ij.append(1)
            # we are exact pixel value of A in b
            b[p - 1] = f_ext[i - 1, j - 1]
```

Fig 8. Inner points code

- The computation of the inner pixels of the image depends on whether the image belong to the A or B (faulty) region.
- As described in "Assembling the matrix A", for those pixels in A region, the coefficient is just 1 for the pixel itself. For those in B region, the coefficients are 4 for the pixel itself and -1 for its neighbors at their corresponding indices in the sparse matrix.
- For the A region, the value of b is pixel's value from the original image, while for the B region is 0.

4. Results

Fig 9. Image 5 before and after inpainting

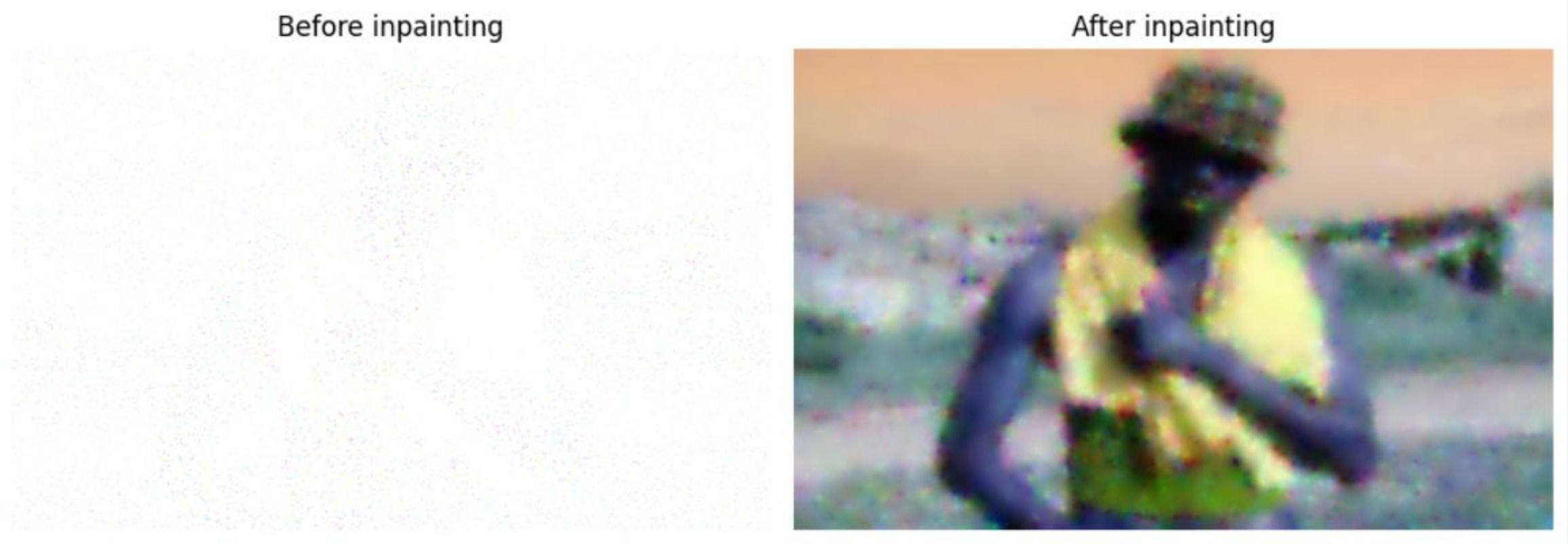

Fig 10: Image 6 before and after inpaiting

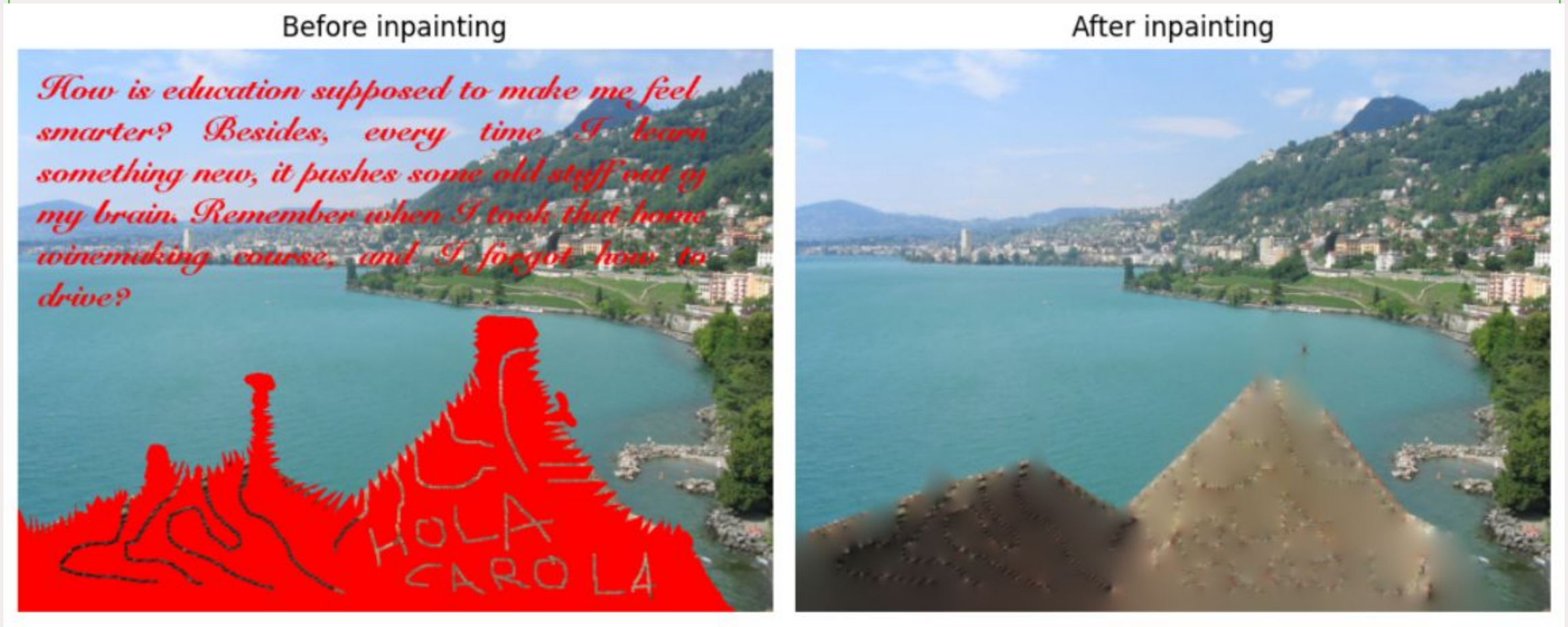

Fig 11. Carola Picture before and after

Fig 12. Personal example before and after inpaiting

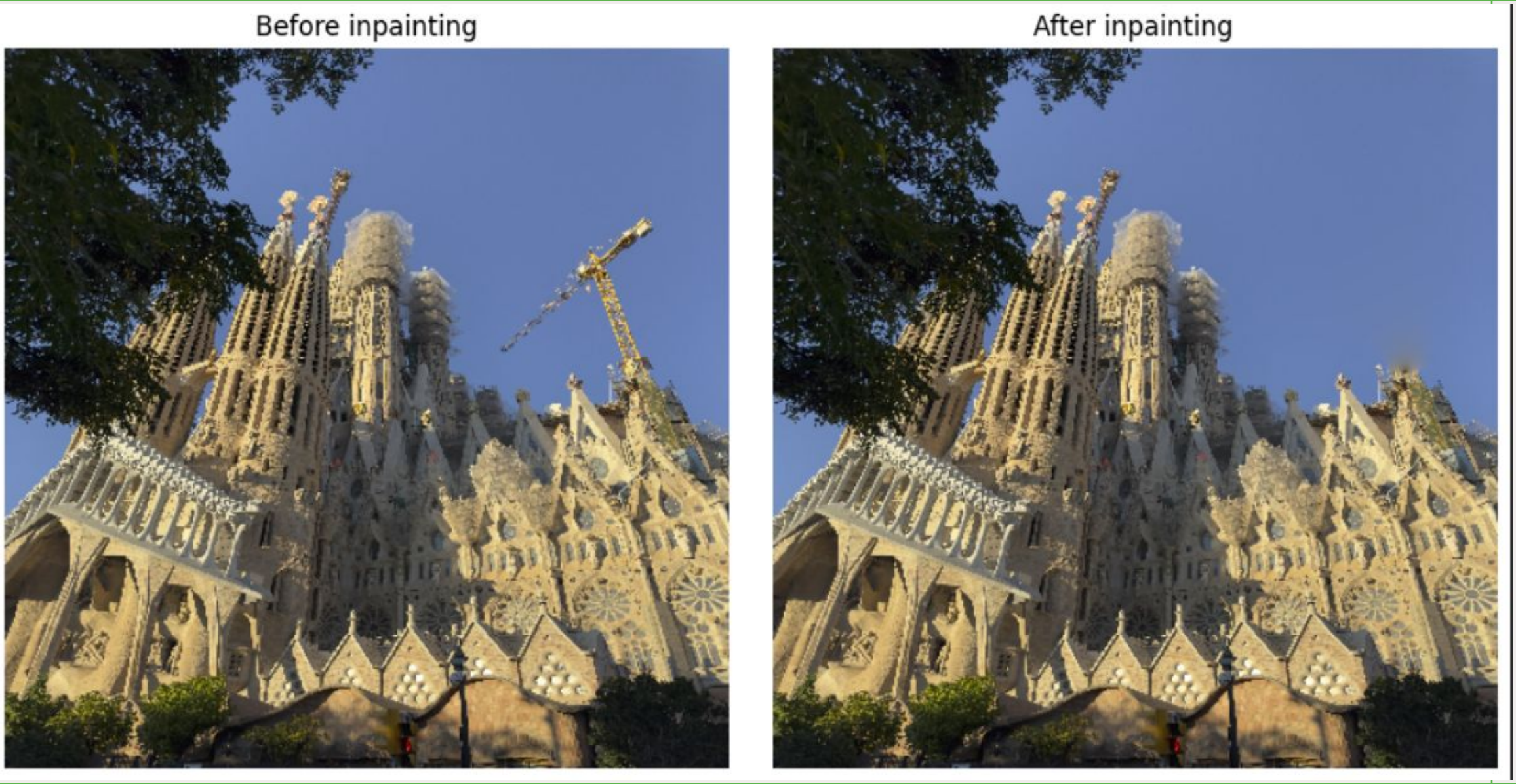

Fig 13. Personal example before and after inpaiting

5. Discussion & conclusions

• The inpainting process, which involves finding the right values for the pixels in a faulty region of an image that would yield credible content based on the surrounding information, can be achieved by solving a Laplace's equation in two dimensions. This results in the diagonal value of 4 and the off-diagonal values of -1 for the coefficients of the faulty region in the A matrix of the linear system $Au=b$, where u is the original image.

•In order to combat the issue of missing neighbours at the boundaries, we use padding to add ghost boundaries to the image.

•The quality of the results depends on various aspects such as the amount of information and the variation of change in it in the original image, the ratio between the area of the faulty region and the region where no inpainting is need, etc.