Synthèse du rapport en anglais

Historically, the earliest inpainting methods focused on solving partial differential equations (PDEs) with boundary conditions defined by the edges of the damaged area. Although these techniques provided satisfactory results, advances revealed that better outcomes could be achieved by copying patches from intact regions. Recent approaches, particularly those based on deep learning, leverage this patch-based principle but typically require a rich training dataset. In contrast, our approach seeks to reproduce results using a highly frugal algorithm that relies solely on existing portions of the image, extracting patches from undamaged areas and selecting the best matches to fill in missing regions, operating in a semi-supervised manner without extensive external training data.

1 Nodes and Labels

To fill the missing region using patches, the algorithm needs to "paste" these patches onto the target area. To achieve this, we define a specific zone where the patches will be applied. This zone consists of a grid, with each grid intersection referred to as a Node. These Nodes represent the points where the center of each patch will ultimately be placed.

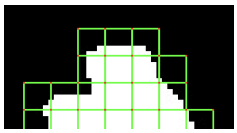


Figure 1: Nodes over mask

In Figure 1, the red dots represent the Nodes, while the green lines indicate the connections between them. The white region corresponds to the unknown or missing area that needs to be filled, and the black region represents the known, intact part of the image.

Nodes and Labels were coded as class with the following attributes :

- **Class**: `Label`
  - `Point` _point : Pixels coordinates of the label center
  - `int` potential : The label's potential value

- `double` messageFromLeft : Message from the left neighbor
- `double` messageFromRight : Message from the right neighbor
- `double` messageFromTop : Message from the top neighbor
- `double` messageFromBottom : Message from the bottom neighbor

- **Class**: `Node`
  - `int` index : Unique index of the node
  - `Point` nodePoint : Coordinates of the node
  - `int` leftNeighbor : Index of the left neighboring node
  - `int` topNeighbor : Index top neighbor
  - `int` rightNeighbor : Index right neighbor
  - `int` bottomNeighbor : Index bottom neighbor
  - `vector<Label>` nodeConfusionSet : candidate labels for this node

1.1 Nodes Construction

To construct the Nodes over the mask, we first need to determine their locations. A natural approach is to build a grid over the entire image and retain only the Nodes whose L1 norm distance from the known region is less than half the patch size, referred to as the radius. To achieve this, we extend the original mask along both the x and y axes by the radius r. In mathematical morphology, this process corresponds to a dilation by the structuring element $[-\mathrm{r},\mathrm{r}]^2$. Since this structuring element is separable, it can be decomposed into two 1D dilations with structuring element $[-r, r] \times 0$ and $0 \times [-r, r]$, one along each dimension.

We obtain the following algorithm :

Algorithm 1: ExtendMask

Input: An image with a binary mask M and a patch radius pr.
Output: An extended mask image EM.
begin
    Create EM, copy of M
    for y *in each row of* EM **do**
        for x *in each column of* EM **do**
            if $M(x, y) = 1$ **then**
                $EM(x - pr : x, y) \leftarrow 1$
                while $M(x, y) = 1$ *and* $x \in M$ **do**
                    $x++$
                if $x \in M$ **then**
                    $EM(x + 1 : x + pr, y) \leftarrow 1$
    Create EM_{temp}, copy of EM
    x in each column of EM y in each row of EM **if** $EM_{temp}(x, y) = 1$ **then**
        $EM(x, y - pr : y - 1) \leftarrow 1$
        while $EM(x, y) = 1$ *and* $y \in M$ **do**
            $y++$
        if $y \in M$ **then**
            $EM(x, y + 1 : y + pr) \leftarrow 1$
    Return EM

As a result, We obtain an extended mask that will be used throughout the remainder of the project:

(a) Original mask (b) Mask extended

Figure 2: Input and Output for the ExtendMask function

Finally, we need to retain only the nodes within the extended mask. By construction, these nodes will always intersect the original mask, indicating that the regions they represent are not entirely known. The grid of nodes that we aim to cover is illustrated in Figure 3.

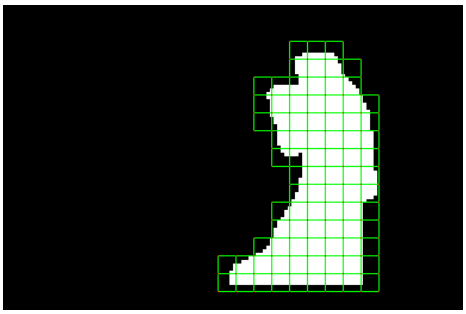


Figure 3: Nodes over mask

We used the following algorithm:

Algorithm 2: nodesOverMask

Input: A binary image M of size $w \times h$ M corresponding to the extended mask and a patch radius pr.
Output: A vector of nodes VN.
begin
  Initialize vector $VN \leftarrow$
  Initialize index $\leftarrow 0$
  for $y = 0 \dots h + pr$ **do**
    for $x = 0 \dots w + pr$ **do**
      if $EM(x, y) = 1$ **then**
        Initialize point p of coordinates (x, y)
        Initialize node n of coordinates (x, y)
        $VN \leftarrow VN \cup n$
        `Check for existing left neighbor`
        for *each node* n_i *in* VN **do**
          if n_i *x position equal to* $x - pr$ n_i *y position is equal to* n **then**
            Add node n as a right neighbor to node n_i
            Add node n_i as a left neighbor to node n
            break loop
        `Check for existing top neighbor`
        for *each node ni in* VN **do**
          if n_i *x position equal to* n n_i *y position is equal to* $y - pr$ **then**
            Add node n as a bottom neighbor to node n_i
            Add node n_i as a top neighbor to node n
            break
  Return VN

1.2 Label extraction

The patch candidates intended to cover this new region are referred to as labels. Unlike nodes, labels must be fully known, as they will be copy-pasted to create the final image. Therefore, labels cannot intersect with the mask and must be entirely within the image. Utilizing our previously defined extended mask, we can easily extract all potential label candidates by examining the black region. In fact, each pixel within the black region represents the center of a label (patch) that can be used to cover a node (Figure 4).

Figure 4: Black Pixels are centers of label candidate

2 Belief Propagation

2.1 Concept

To identify the best label candidates for covering the nodes, we need to assign a belief to each label for every node. For nodes located primarily at the edges of the mask, this process is straightforward, as we have ample information about the preferred candidates. However, for nodes that are entirely within the mask, this becomes significantly more challenging due to the lack of available information. To tackle this issue, we will employ belief propagation, where each node will send and receive messages concerning the likelihood of a label fitting that specific node.

2.2 Initialisation of belief propagation

To initiate belief propagation, we first need to assign preferred labels to the nodes situated at the boundary between the mask and the rest of the image. Based on how we constructed the nodes, there are two types that qualify as "threshold" nodes, as illustrated in Figure 5.

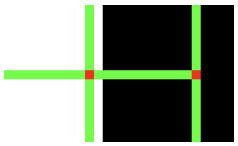


Figure 5: Threshold Nodes

Both types of nodes cover regions that are not exclusively within the mask, making them eligible to be considered threshold nodes. However, the node on the right only overlaps with 2 pixels in the x-direction of the known region, rendering it nearly an unknown node. Therefore to enhance the robustness of the algorithm, we restrict the definition of threshold nodes to those whose centers lie outside the mask (the left one in the figure). This approach allows for a more definitive selection of the initial candidates, which will influence the remainder of the new image.

2.3 Criteria for the selection of labels at initialisation

Once the threshold nodes are identified, we need to assign them label candidates, preferably those closest to the known region outside the mask. To do this, we use the Sum of Squared Differences (SSD) criterion to select the labels that best match the known area. The SSD potential for a node p with a patch candidate x_p is defined as:

$$V_p(x_p) = \sum_{dp \in \mathbf{patch}} \mathcal{M}(p+dp)\left(\mathcal{I}_0(p+dp) - \mathcal{I}_0(x_p+dp)\right)^2$$

Here, $\mathcal{M}$ is a binary mask, with 0 inside the mask and 1 outside, and $\mathcal{I}_0$ is the input image. Any patch comparison criterion, not limited to SSD, can be used in this context.

To make the algorithm more efficient and avoid unnecessary delays, we introduce two additional criteria aimed at discarding less useful candidates:

1. Threshold Confusion: We set a confusion threshold to filter out labels whose SSD exceeds this limit.

2. Threshold Similarity: We retain only candidates whose SSD with every other already selected candidate is above a set threshold.

These thresholds ensure that the algorithm focuses on the most relevant candidates, preventing the inclusion of those that would only slow down the process without adding value.

The label candidates for each node will be stored in a `std::vector`, where the order is determined by a measure called belief. This ensures that the most relevant candidates appear first. The belief for a node p with a label x_p is defined as:

$$b_p = -V_p(x_p)$$

This belief score reflects how well a label fits the node, with higher belief indicating better matches. The set of labels selected for each node is called the node's *confusion set*, and only the candidates within the threshold boundaries are retained. From now on, we will use belief to guide the selection of the confusion set.

We then define the InitialisationAlgorithm :

Algorithm 3: assignInitialLabels

Input: Binary image EM of size $w \times h$ corresponding to the extended mask, Binary image M of size $w \times h$ corresponding to the mask, RGB image I of size $w \times h$ corresponding to the initial image and a patch radius pr.

Output: A vector of nodes VN.

begin

- Initialize vector of Nodes VN with nodesOverMask
- **for** *each node n in* VN **do**
  - **if** *n is entirely inside* M **then**
    - continue
  - **for** $x = pr, \ldots, w - p_r$ **do**
    - **for** *y in the height of EM with y farther than pr from EM edges* **do**
      - **if** *EM(x,y)=0* **then**
        - **Initialize** label l with coordinates x and y
        - **if** *l satisfies 3.3 condition regarding n* **then**
          - **Push** l in node n
- **Sort** VN with respect to the belief

2.4 Starting the belief propagation

Once all nodes have their label candidates set, we assign to each a priority defined as the opposite of the size of the confusion set of the node, for node p it gives:

$$\text{priority}(p) = -|\mathbb{CS}(p)|.$$

This definition is suitable as nodes with fewer label candidates should be more certain of their candidates that nodes having more candidates. In this respect, any decreasing function applied to the cardinal of the confusion set would work.

Figure 6: Confusion set

In Figure 6, the patches are color-coded from red to black, with red representing nodes that have a smaller confusion set, thus have higher priority. We can observe that, in this example, the nodes with the highest priority are located within the sand region. This might seem counterintuitive, given the abundance of sand in the image. However, the sand's texture causes each patch to differ significantly from the others, leading to fewer valid candidates and thus higher priority for those nodes (see Figure 7).

Figure 7: 3 patch candidates for a sand patch in the bottom right region of the mask

In contrast, Figure 8 shows that the sky region has a lower priority, as many sky patches are highly inter-compatible due to their similarity.

Figure 8: 20 patch candidates for a region in the sky

Nodes that have a confusion set (only threshold nodes at the beginning) will be stored in a priority queue, ensuring that those with the highest priority are addressed first during the belief propagation process. This queue will continuously update throughout the entire process.

2.5 First messages sent

Starting from the node with the highest priority, we send a message to each of its neighbors.

In our example (Figure 9), the first threshold node (node 2) will send a message to the interior node (node 5) :

(a) message sent (b) overlap region

Figure 9: First message sent

The nature of the message sent is the likelihood for a given label to overlap any label of the confusion set. In other words the node receiving the message is trying to best fit the labels already selected in the confusion set of the node sending the message. Therefore the message sent is defined as follow:

$$m_{2,5}\left(x_5\right) = \min_{x_2 \in \mathcal{L}} \left\{ V_{2,5}\left(x_2, x_5\right) + V_2\left(x_2\right) \right\},$$

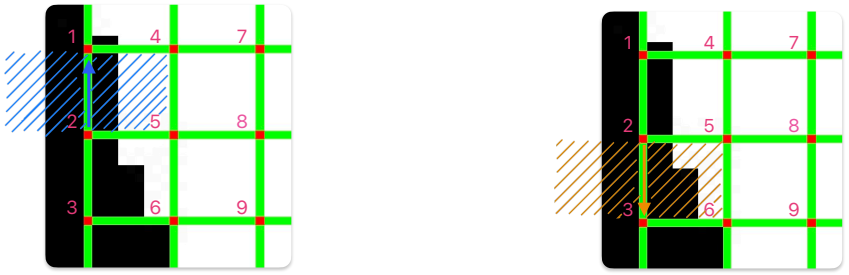
with 2 being the node sending the message, 5 the node receiving (as represented in figure 9a), and x_2 and x_5 labels candidates within the confusionset of the corresponding nodes. Finaly $V_{2,5}\left(x_2, x_5\right)$ is simply the SSD over the overlapping region represented in figure 9b). In other words, the SSD is calculated on the right half of the patch candidate identified at x_2 and the left half of the patch candidate identified at x_5.

With the newly received message, each label now has an associated belief, which combines the potential of selecting that label for the node and the message received from the neighbor regarding that label (in this case, node 5 has only received a message from node 2):

$$b_5(x_5) = -V_5(x_5) - m_{2,5}(x_5) = -m_{2,5}(x_5).$$

Thus node 5 has a confusion set and therefore a priority. We then update the priority list by adding node 5. Before continuing we remove node 2 from the list as it cannot send anymore message as long as everyone has not sent their messages.

Node 2 sends also messages to its top (Node 1) and bottom (Node 3) neighbors :



(a) Message sent to Node 1

(b) Message sent to Node 3

Figure 10: Message sent by Node 2 to Node 1 and 3

In this case, unlike Node 5, the receiving nodes already possess a confusion set, and each label has an associated belief. Consequently, their beliefs are updated as follows:

$$b_1(x_1) = -V_1(x_1) \quad \longrightarrow \quad b_1(x_1) = -V_1(x_1) - m_{2,1}(x_1)$$

$$b_3(x_3) = -V_3(x_3) \quad \longrightarrow \quad b_3(x_3) = -V_3(x_3) - m_{2,3}(x_3)$$

This allows us to establish a more precise definition of belief as follows:

$$b_p\left(x_p\right) = -V_p\left(x_p\right) - \sum_{r:(r,p)\in\mathcal{E}} m_{rp}\left(x_p\right)$$

where $\mathcal{E}$ denotes the neighborhood of node p.

2.6 Rest of the forward pass

Once all the messages have been sent from the first node, we create a `std::stack` of commited node and push the node that just sent the first messages to it. From this point forward, any node that sends a message will be added to the pile once all its messages are sent. This approach ensures that no node is selected twice to send messages and that nodes receiving messages have not been already been commited.

Algorithm 4: Priority-Belief Propagation Initialization and First Message Passing

begin
- Initialize: Assign initial priorities with assignInitialPriority
- Declare all nodes as uncommitted
- Set initial messages $m_{pq}(x_q) = 0$ for all p, q in E
- **ForwardPass:**
- **for** *each uncommitted node p in order of descending priority* **do**
  - **for** *each label* x_p *in* L **do**
    - Compute $V_p(x_p)$ based on local image data and constraints
    - Initialize the message to a large value (e.g., infinity)
  - **for** *each neighboring node q of p* **do**
    - **for** *each label* x_q *in* L **do**
      - Calculate message from p to q $m_{pq}(x_q)$
      - Update $m_{pq}(x_q)$ based on the computed value
    - Send message $m_{pq}(x_q)$ to node q
  - Mark node p as committed

3 Backward Pass

The backward pass is essentially the reverse of the forward pass, where each node that received messages now sends messages. To facilitate this process, we created a queue of all the committed nodes responsible for sending messages during the forward pass. This phase relies on removing each element from the queue in a top-to-bottom manner, allowing each of these nodes to send their messages. It is important to note that, unlike in the forward pass, there is no update of priorities during the backward pass; the order of nodes sending messages is determined solely by the stack provided as input.

Algorithm 5: Backward Pass of Priority-Belief Propagation

Input: Committed pile of Nodes : forwardOrder
begin
- **for** *time* $= N$ *to* 1 **do**
  - $p \leftarrow$ forwardOrder[time]
  - $p \rightarrow$ committed $\leftarrow$ false
  - **for** *any "committed" neighbor q of node p* **do**
    - Send all messages $m_{pq}(\cdot)$ from node p to node q
    - Update beliefs $b_q(\cdot)$ as well

The remaining process of the inpainting algorithm is based on alternating between forward and backward passes. Although it may seem that increasing the frequency of these passes would yield better results, we observed that the size of the confusion set—which diminishes with each iteration—reaches l_{min} for

every node. As a result, further iterations become unnecessary. Consequently, we have limited the project to three iterations of the forward and backward passes.

4 Results

To improve the practicality and efficiency of the algorithm, we introduced the following parameters:

1. l_{min}: Minimum size for a confusion set.
2. l_{max}: Maximum size for a confusion set.

The lower bound l_{min} ensures that the confusion set is not empty, which is important because our thresholding method tends to reduce the set size by filtering out irrelevant label candidates. On the other hand, while l_{max} is not strictly necessary, it significantly speeds up the algorithm by limiting the confusion set to only the most relevant candidates. In our tests, we used $l_{min} = 3$ and $l_{max} = 50$. Although setting l_{max} to 50 might seem small, we observed no significant impact on the results when increasing l_{max} in our low-resolution examples ($\leq 10^5$ pixels).

(a) l_{max}=20 time =20s | (b) l_{max}=50 time =1m07s | (c) l_{max}=100 time =1m47s | (d) l_{max}=500 time =2m12s

Figure 11: size patch: $s_p = 25$ pixels, thresholdConfusion =-s_p^2*600, thresholdSimilarity=$s_p^2 * 1200$

4.1 Inpainting

Lighthouse

The lighthouse image is a 171×256 pixel picture of a red brick lighthouse, set against a backdrop of blue sky with some clouds. The mask covers the entire lighthouse, and the objective is to seamlessly remove it in a visually convincing way. The challenge lies in the fact that, while the sky appears relatively

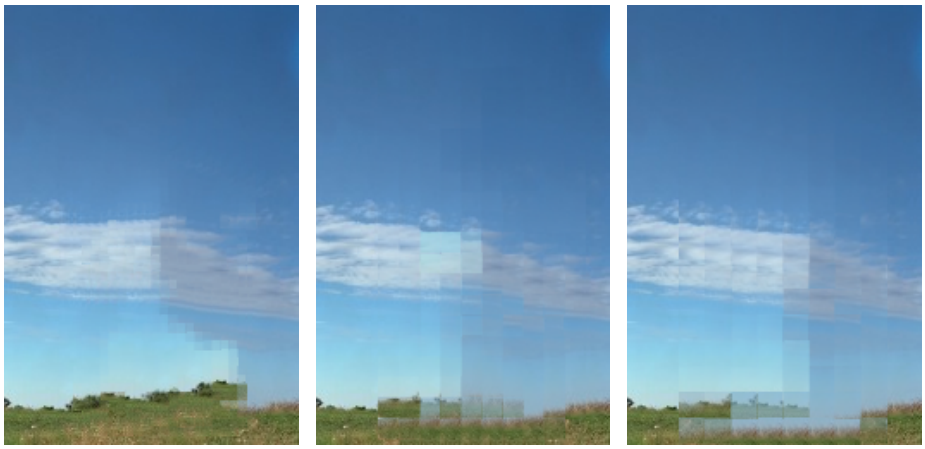
uniform, the presence of clouds complicates the filling process. Additionally, the intensity of the blue sky differs on each side of the lighthouse, with the left being significantly brighter than the right. Lastly, although the grass is easier to handle, there are few available grass patches in the unmasked areas of the image for use in the reconstruction.

(a) lighthouse initial image

(b) lighthouse mask

Figure 12: Lighthouse picture with its mask

In this exemple the size of patches seems to have a lot of effect and smaller patches seem to work better. The reason may lie from the fact than the cloud do not follow a horizontal or vertical line but rather a diagonal, this makes the process of following this line harder for bigger patches (Figure 13).



(a) patch size =11 (b) patch size =21 (c) patch size =31

Figure 13: $l_{max} = 50$,
threshold confusion $= -s_p^2 \times 600$, threshold similarity $= s_p^2 \times 1200$

The blurriness observed at the boundary between grass and sky when using larger patches can be attributed to the way the final image is reconstructed. This process involves averaging the pixels from overlapping patches, with each patch being weighted according to its priority (refer to section 3.4). Consequently, when the patches differ significantly from one another, the resulting weighted average can appear blurred.

We can also examine the effect of the threshold discussed in section 3.3, beginning with the threshold on similarity (Figure 14).

(a) threshold similarity = 0 (b) threshold similarity = $s_p^2 \times 600$ (c) threshold similarity = $s_p^2 \times 1200$

Figure 14: patch size = 11, $l_{max} = 50$, threshold confusion = $-s_p^2 \times 600$

The threshold for similarity does not significantly affect the results because it encourages a diversity of patches within the confusion set. In the case of the lighthouse, the colors are nearly homogeneous, which minimizes the impact of this diversity on the final result

Alternatively, adjusting the threshold for the confusion set, which establishes the minimum belief needed for a candidate to qualify, is relatively straightforward. Increasing this threshold makes it more challenging for labels to be included in the confusion set, thereby reducing its size. However, as noted in Section 5.0, the size of the confusion set cannot drop below l_{min}; thus, raising the threshold beyond this point yields no benefit. Conversely, decreasing the threshold allows more candidates to enter the confusion set, resulting in an increased size. However, for the same reason, the size cannot surpass l_{max}, which renders any further reduction in the threshold ineffective.

This consideration is particularly crucial during the initialization phase (as outlined in Algorithm 3), as a proper priority ranking is essential for the forward pass to function correctly. If this ranking is not established, the first node to begin sending messages would be chosen at random. Therefore, it is essential that, during initialization, at least one node has a confusion set smaller than l_{max}. Likewise, if all confusion sets are at l_{min}, the start of the forward pass would also be arbitrary. It is important to note that the values we obtain are significantly influenced by the l_{min} and l_{max} parameters we have defined in Section 5.0, set to 3 and 50, respectively.

We found the threshold for the confusion set must stay within [-$patchSize^2$*1200;-

$patchSize^2$*100], Figure 15 shows some results with different values for this threshold.

(a) threshold confusion = $-s_p^2 \times 100$ (b) threshold confusion = $-s_p^2 \times 600$ (c) threshold confusion = $-s_p^2 \times 1200$

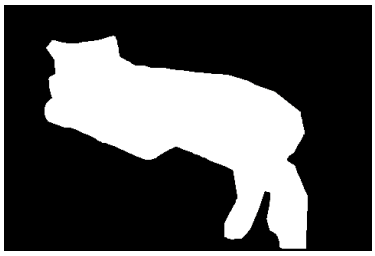
Figure 15: $l_{max} = 50$, patch size = 11 pixels, threshold similarity = $s_p^2 \times 1200$

In our analysis of the lighthouse image, we found that it exhibits the best consistency when modifying parameters. Although the final results display notable defects, particularly in the gradient of the sky and the continuity of the grass, the outcomes remain satisfactory. This suggests that the patch-based method, when used without any additional manipulation, yields results that cannot be improved upon through manual patch pasting.

Leopard

The leopard image has dimensions of 481×321 pixels and depicts a leopard resting on a tree branch, set against a softly blurred gradient of green, brown, and yellow in the background. Since the mask obscures the leopard, the algorithm must effectively extend the branch in a realistic manner while preserving the gradient background. A significant challenge arises from the limited availability of branch patches in the source image, compounded by the fact that the branch is neither perfectly horizontal nor straight.

(a) leopard initial image



(b) leopard mask

Figure 16: Leopard picture with its mask

Using the parameters specified for the lighthouse here are some results obtained for the leopard with its mask

(a) patch size=11

(b) patch size=21

(c) patch size=31

Figure 17: $l_{max} = 50$
threshold confusion $= -s_p^2 \times 600$, threshold similarity $= s_p^2 \times 1200$

In contrast to the lighthouse example, larger patches appear to give better results in this case. The final outcome with a patch size of 31 is quite satisfactory and would be nearly perfect with improved post-processing to blur the edges of the patches.

Beach player

The baseball image is a 386x256 pixel depiction of individuals playing baseball on the beach, with the mask covering the blue player in the center right. This example presents significant challenges due to the lack of uniform regions; the sky features a blue gradient, the horizon is mostly straight but obscured by players on the beach, and the sand varies greatly in texture (see Figure 7). The most difficult aspect of this example is preventing the propagation of anomalies. For instance, if the algorithm extends the sand area but there are players within that region, it may inadvertently paste portions of those players into the area being filled, complicating the reconstruction process.

(a) baseball initial image

(b) baseball mask

Figure 18: Baseball image and its mask

(a) patch size=11

(b) patch size=21

(c) patch size=31

Figure 19: $l_{max} = 50$
threshold confusion $= -s_p^2 \times 600$, threshold similarity $= s_p^2 \times 1200$

The results obtained are highly inconsistent. Although a patch size of 11 pixels yields the best outcome, this appears largely due to chance, as further investigation revealed that patch sizes of 9 or 13 produce similar results. This suggests that the propagation of anomalies into the filling region remains a persistent issue.

4.2 Texture

Minor modifications were made to the algorithm to facilitate texture completion. Although this is not its primary purpose, it can be beneficial and requires minimal changes to the inpainting process. All results presented here were achieved using the optimal parameters specified in the inpainting section (5.1). The algorithm now accepts three arguments: the initial image, along with the desired width and height for the final image. In this case, a mask is unnecessary; instead, the initial image is used to create a larger image of the desired dimensions, filling the empty spaces with black pixels. The objective is to complete these black pixels to extend the original texture.

(a) Flowers texture mask (b) Flowers texture extesnsion

Figure 20: Flowers texture and its extension

(a) Lizard texture (b) Lizard texture extension

Figure 21: Lizard texture and its extension

5 Conclusion

While this method can give good results it is particularly sensitive to the choice of certain parameters particulary the size of patch where a certain value make the algorithm work for an exemple and not for the other. For complex examples it tends to spread anomalies during the belief propagation, this arises from the fact that we limit the number of patches within the confusion set to a rather small number while this improve computational time it sometimes can exclude very good patches from the confusion set. On the other hand it works fine for texture completion.