DIP PROJECT REPORT

Problem Statement

- Match patches using a randomized correspondence algorithm that can be used for structural image editing.
- We call this algorithm Approximate Nearest Neighbour Algorithm.
- Apply the A-NNF Algorithm to one practical application in image processing. (We chose this to be Hole Filling in Images using Patch Matching).

Motivation

- Previous research in graphics and vision has leveraged such nearest-neighbor searches to provide a variety of high-level digital image editing tools.
- However, the cost of computing a field of such matches for an entire image has eluded previous efforts to provide interactive performance.
- Our algorithm offers substantial performance improvements over the previous state of the art (20-100x).

Overview

- Illustration of the convergence of Approximate Nearest Neighbour Algorithm both theoretically and practically
- The theoretical proof is done by constructing appropriate use cases.
- For the practical proof:
    - Input:
        - Image A
        - Image B
    - Method:
        - Construct Nearest Neighbour Field (NNF) of Image B to Image A using our proposed algorithm.
        - Construct Image B from the matched patches of Image A stored in NNF previously constructed.Let this be Image C
    - Expected Output:
        - Show the similarity of Image B and Image C which in turn shows the convergence of our proposed algorithm.
- For the application: (Hole Filling of Images using NNF)
    - Input: Image with the hole to be filled given manually

    - Method: Search Space Constraints are given manually to approximate the NNF for patch match of the regions to be filled.
    - Output: Original Image with Holes filled appropraitely.

Theoretical Proof:

Step1:

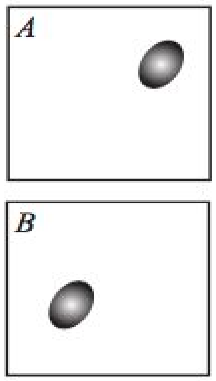
- We start by analyzing the convergence to the exact nearest neighbourhood field and then extend the solution to an approximate solution.
- Assume A and B have equal number of pixels M (same size)
- Probability that the random initialization for a pixel is the best offset=1/M
- Probability that the random initialization for all the pixels is not the best offset=$(1-1/M)^M$
- Probability that at least one pixel has the best offset in the random initialization=$1-(1-1/M)^M$
- Lim (p) =$1-e^{-1}$
  (M->infinity)

Step2:

- For an approximate solution, we can also consider a “correct” assignment to be any assignment within a small neighbourhood of size ‘C’ pixels around the offset.
- In that case p =$1-(1-C/M)^M$
- Lim (p) =$1-e^{-M}$
  (M->infinity)

Step-3:

- Consider a synthetic example in which both A and B have M pixels each but we are interested in matching a region of m pixels in A to a region of m pixels in B
- No further information is provided about M-m pixels in A and B



- The probability that any of the m pixels lands within the neighborhood *C* of the correct offset is given by p=$1-(1-C/M)^m$
- The probability that we did not converge on iterations 0, 1, ..., *t* 1 and

converge on iteration *t* is
p(1-*p*)^*t* .

- The probabilities thus form a geometric distribution, and the expected time of convergence is
E(t) = 1/*p* -1
- To simplify , let the relative feature size be gamma=m/M
- Take the limit as M->infinity

$$\langle t \rangle = [1-(1-C/M)^{\gamma M}]^{-1} - 1$$
$$\lim_{M\to\infty} \langle t \rangle = [1-\exp(-C\gamma)]^{-1} - 1$$

- By taylor expansion for small gamma,

$$\langle t \rangle = (C\gamma)^{-1} - \frac{1}{2} = M/(Cm) - \frac{1}{2}$$

- <t> is our expected number of iterations to convergence remains a constant for large image resolutions and a small feature size m relative to image resolution

Practical Proof:

Pseudo code for NNF:

```
Function [NNF]=PatchMatch(target , source, window)
//Defined in the paper
        radius=source_image_size/4 ;
        alpha=0.5;  // Values found by comparing accuracy and time taken for different values
        max_no_iters=-log(w)/log(alpha);
        //Random Initialization
        NNF->random_initialization_of_indices_from_source(leaving_padded_indices)
        Calculate L2 dists for each pixel window of source and its corresponding pixel window in
target linked by random NNF with and store in offset[ ]
        For max_iters:
            For each pixel in target image:(i,j)
               //Propagation->P
                Calculate idx=argmin(index)[offset(i,j),offset(i-1,j),offset(i,j-1)]
                 if(idx==2)
                        NNF(i,j,1)=NNF(i,j,1)+1;NNF(i,j,2)=NNF(i,j,2);Update offset of (i,j)
                 if(idx==3)
                        NNF(i,j,1)=NNF(i,j,1);NNF(i,j,2)=NNF(i,j,2)+1;Update offset of (i,j)
                //Random Search->S
                    ● Calculate range of search according to radius and max_no_iters
            ● Define the boundaries of search for each pixel according to the above
              range
            ● Update NNF and offsets if we find a better L2 distance for a window
              corresponding to (i,j)
        End
    End
End
//End of function
```

Reconstruction of image:

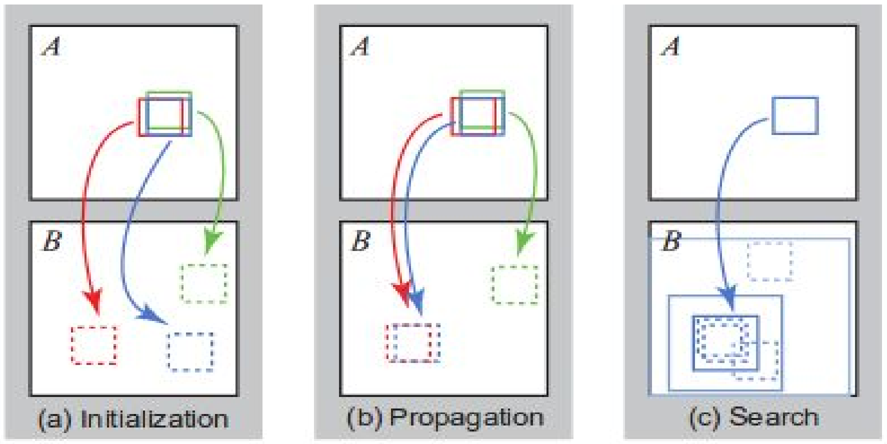


Figure 2: *Phases of the randomized nearest neighbor algorithm: (a) patches initially have random assignments; (b) the blue patch checks above/green and left/red neighbors to see if they will improve the blue mapping, propagating good matches; (c) the patch searches randomly for improvements in concentric neighborhoods.*

We move with a stride of window_size and reconstruct the image using the patch_matched from source image of the first pixel in each stride.

Halting criteria:

- Although different criteria for halting may be used depending on the application, in practice we have found it works well to iterate a fixed number of times. All the results shown here were computed with 4-5 iterations total, after which the NNF has almost always converged.

Application of NNF: Hole Filling using Patch Match:

- Step1:Hole to be filled in image A is taken as the input manually.The shape is hard coded to be rectangular.
- Step2:Search Space Constraint to search for the patches that have to used for hole filling are given manually.The shape is hard coded to be rectangular.
- Step3:NNF is computed between the hole to be filled and the Search Space.
- Step4: Image reconstruction is done using the above obtained NNF.

Experiments Performed:

- Halting Criteria, parametric values in the paper have been found out by conducting experiments for different values.
- Maximisation of speed for convergence has been the motivating criteria.
- Best suited Patch Size was estimated as a trade off between aesthetics of the output and speed of convergence
- The window size and search space constraint have to be manually given as the input to the code.This has to be done in a way such that it maximises the aesthetics of the output image.This experiment was performed for various images manually.

Work Performed:

- Code for implementing Approximate Nearest Neighbour Algorithm.This involves two functions and takes Image A, B as inputs
    - Propagation
    - Random Search
- Code for reconstructing Image B from Image A and the NNF constructed in the previous step.

- Detailed Analysis of the success cases and failure cases of the practical proof.
- Code to fill holes in images which take hole to be filled and search space constraints in the original image as manual inputs.
-

Results: (Successes)

Source Image

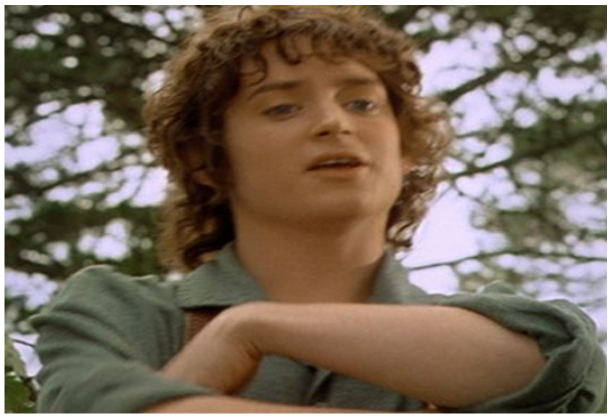
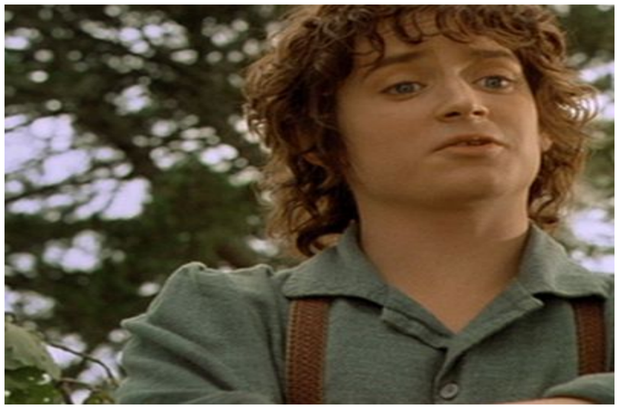
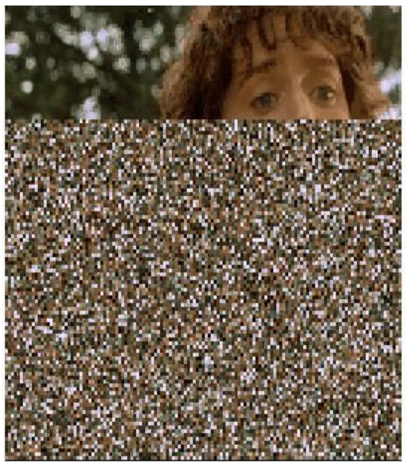


Image need to be reconstructed

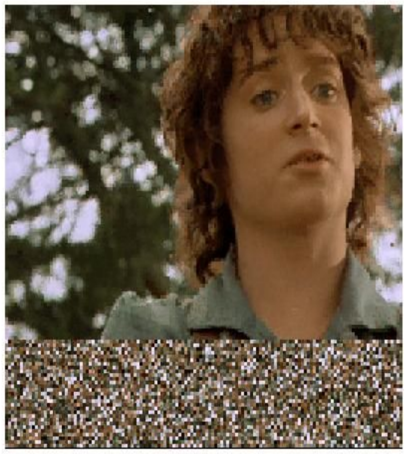


Output Images

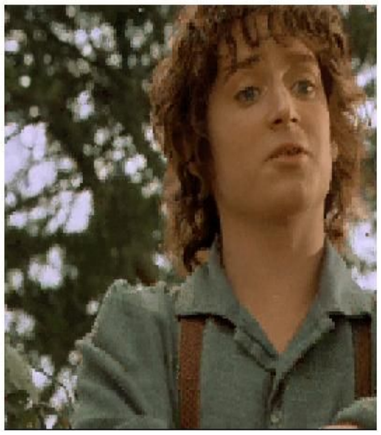
Random Initialisation



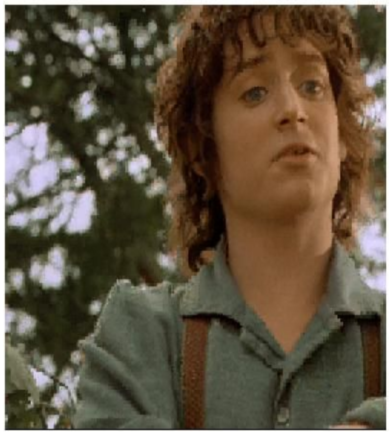
1/4th iteration



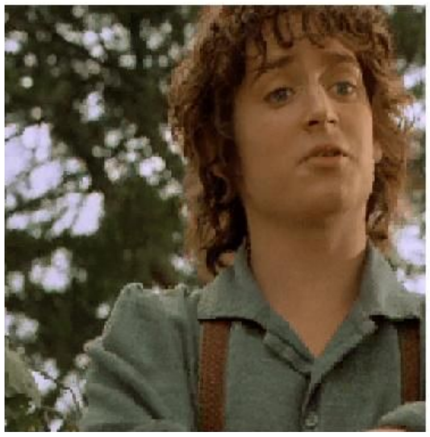
3/4th iteration



1st iteration



2nd iteration



4th iteration

Source Image

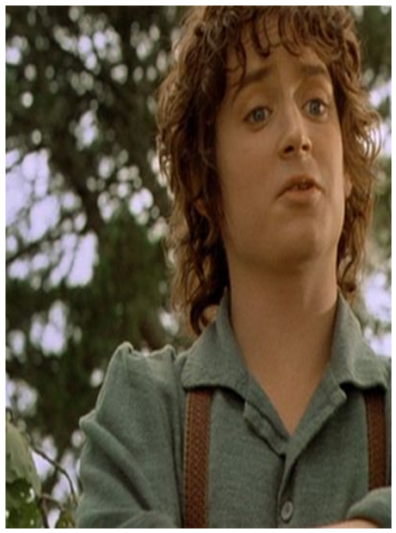
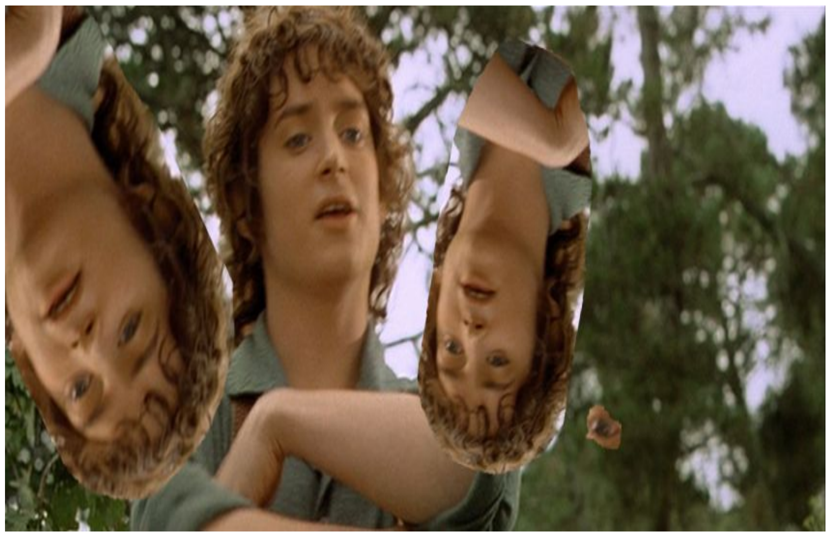
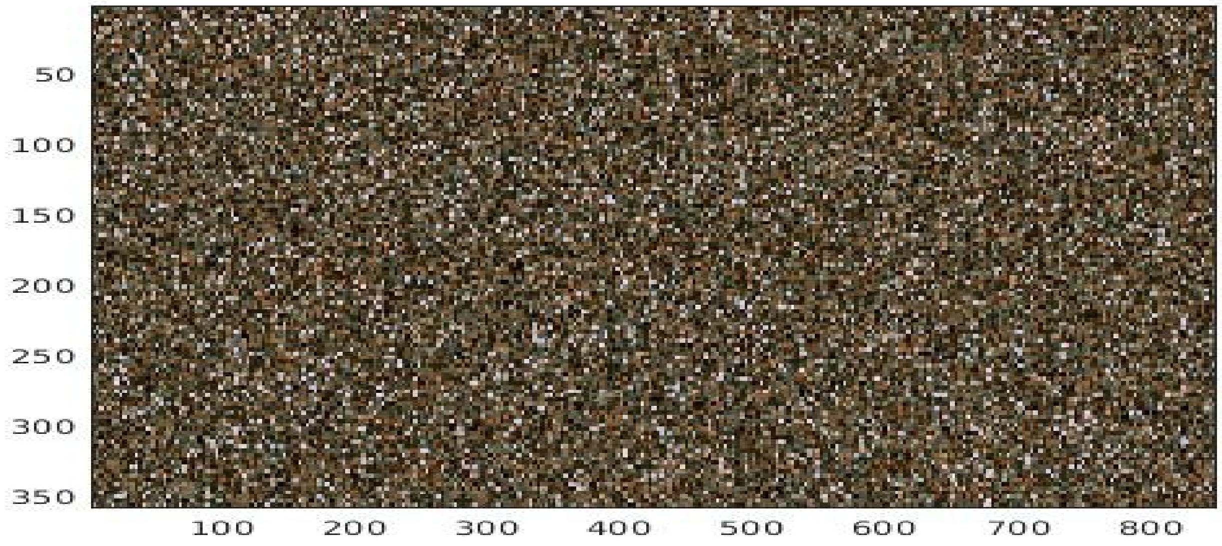
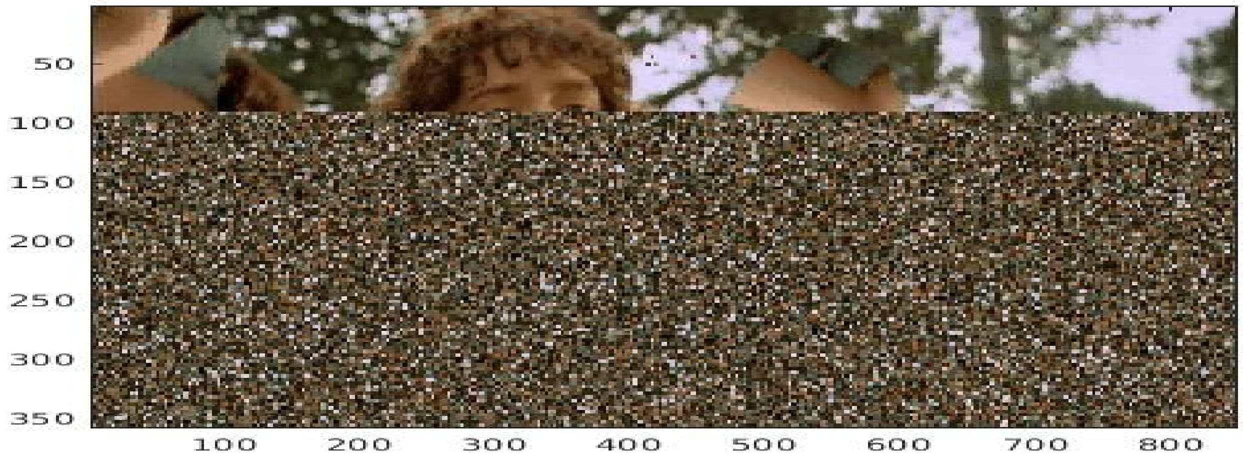


Image to be reconstructed

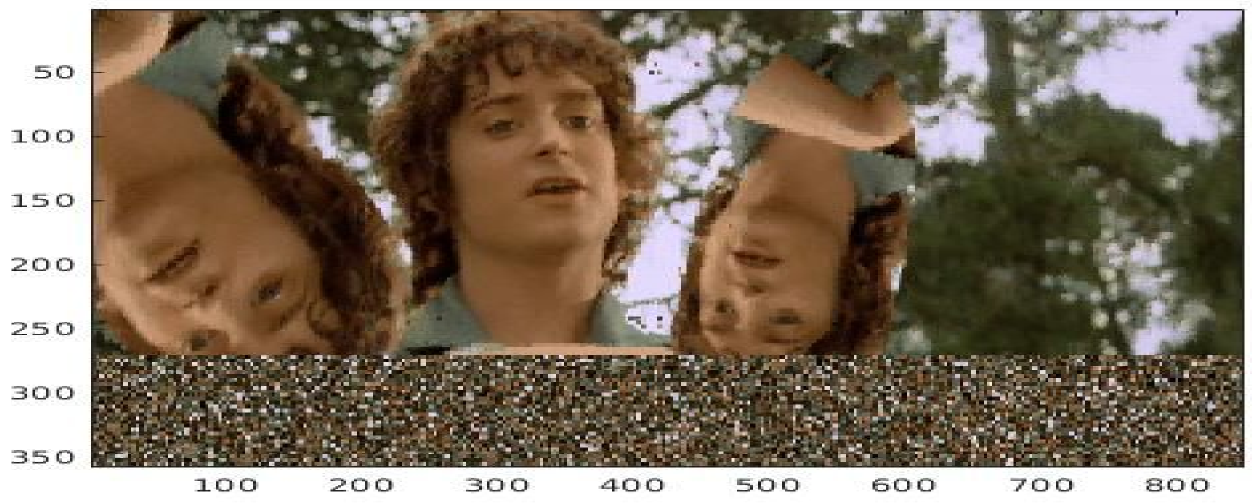




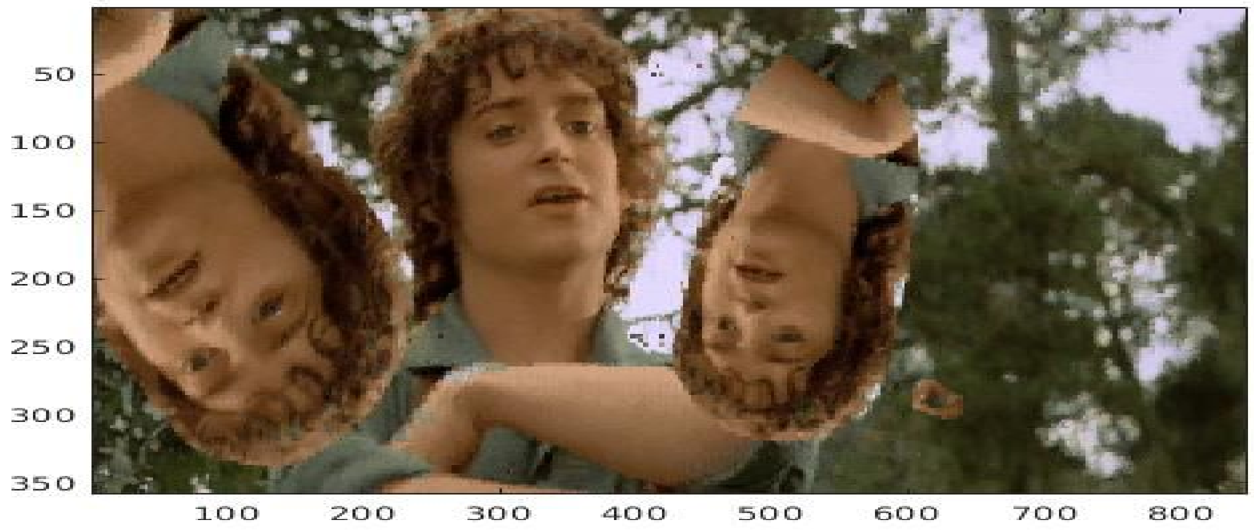
Random Initialisation



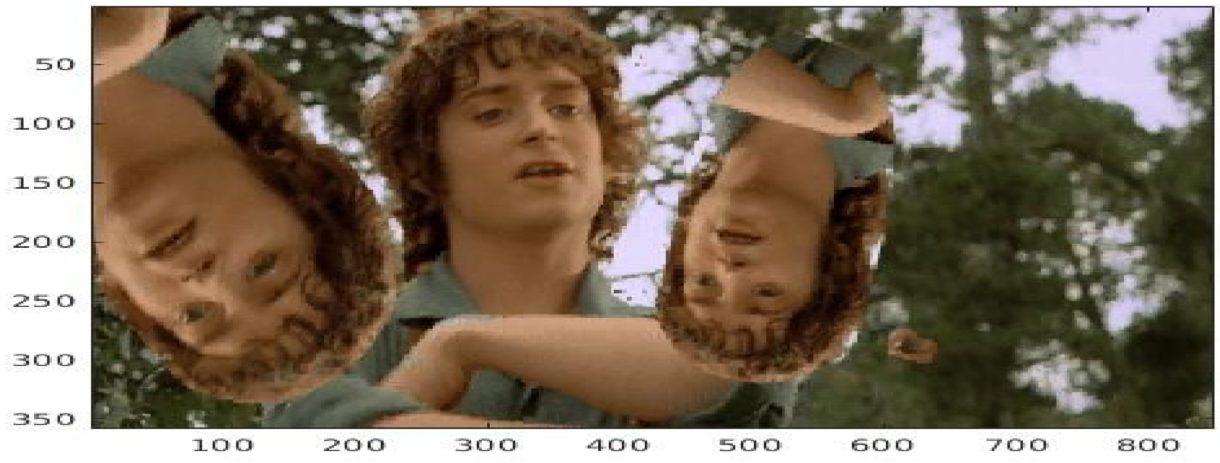
1/4th iteration



3/4th iteration



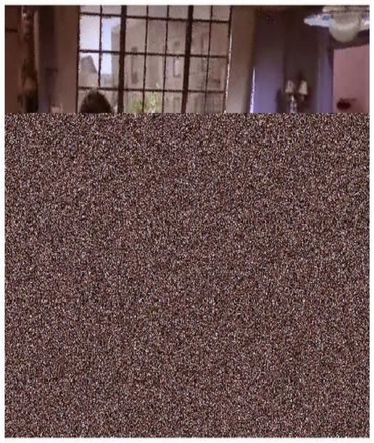
1st iteration

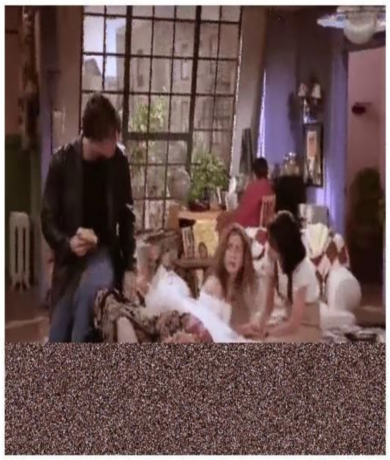


4th iteration

Source Image

Image to be Reconstructed

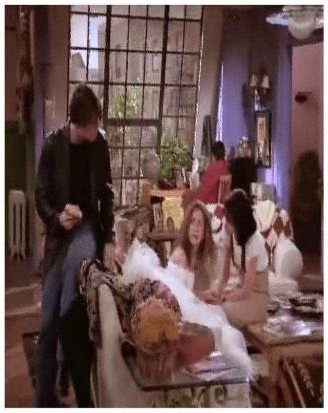


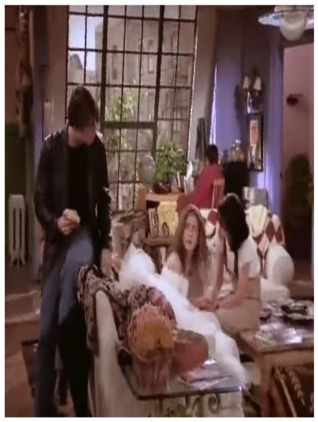


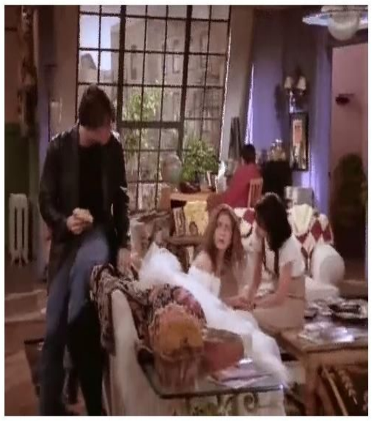
Random Initialisation

1/4th iteration

3/4th iteration







1st iteration

2nd iteration

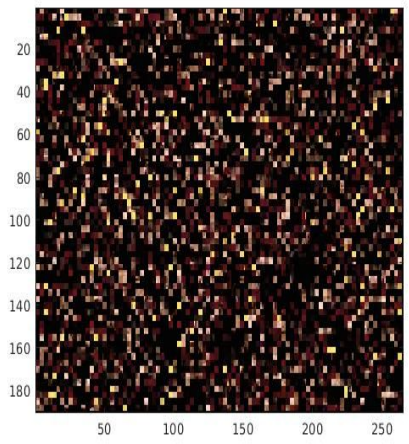
4th iteration

Failure Cases: (Based on Window Size)

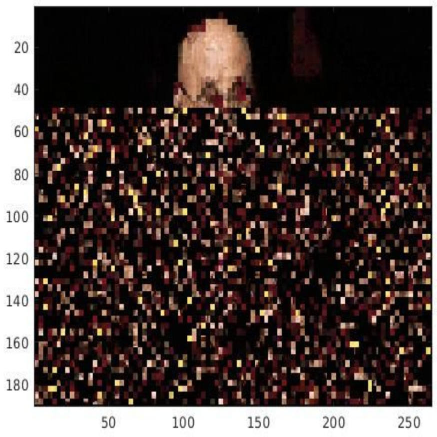
To be Constructed

Source Image

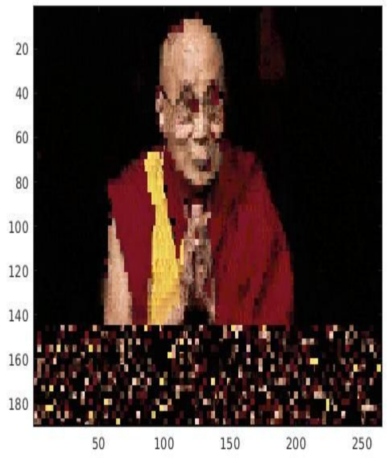
Random search



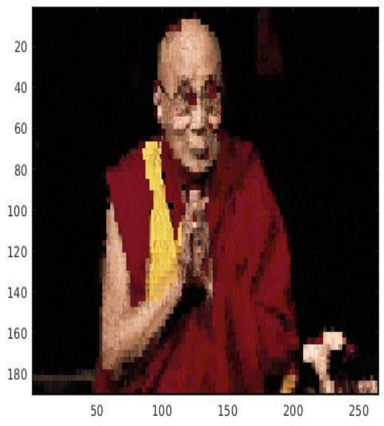
After 1/4rth iteration



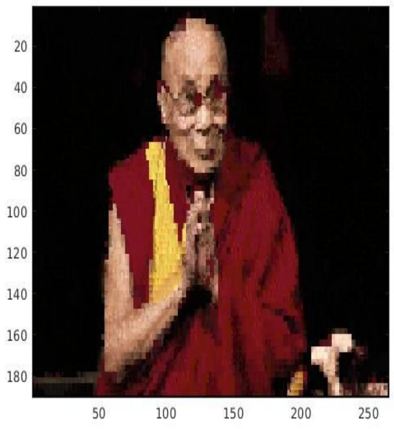
After 3/4rth iteration



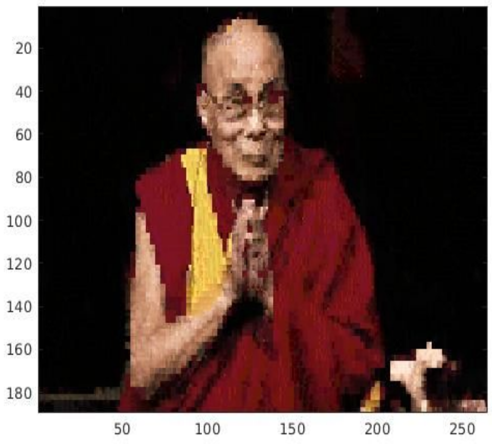
After 1st iteration



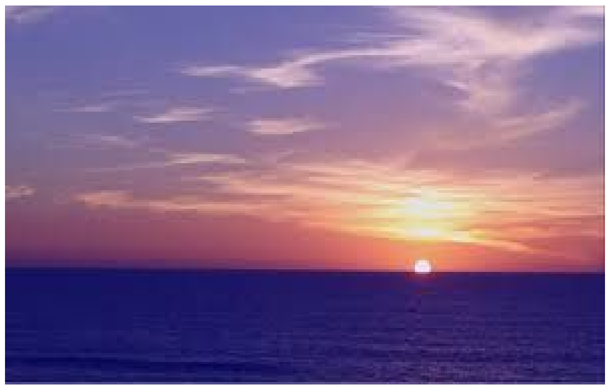
After 2nd iteration



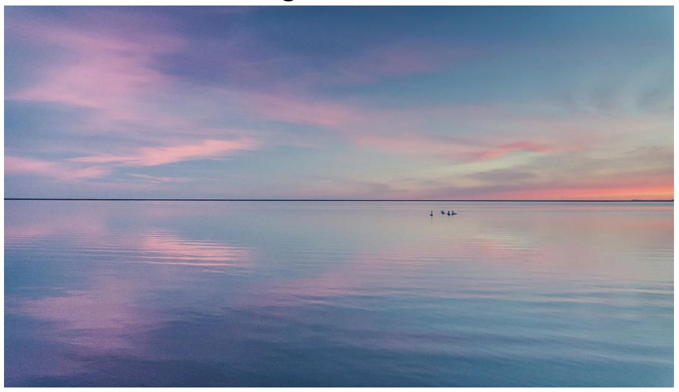
After 4rth iteration



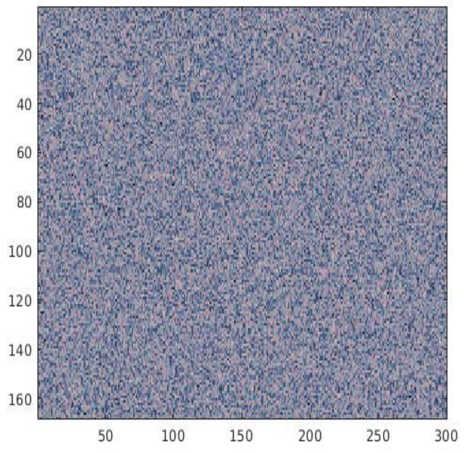
To be Reconstructed



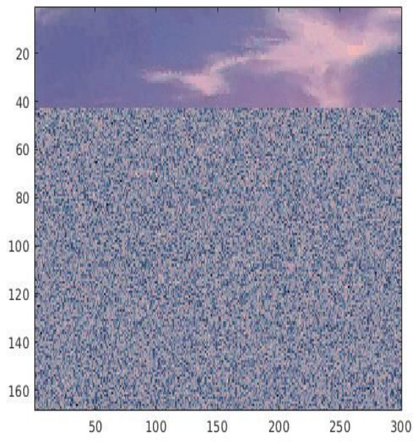
Source Image



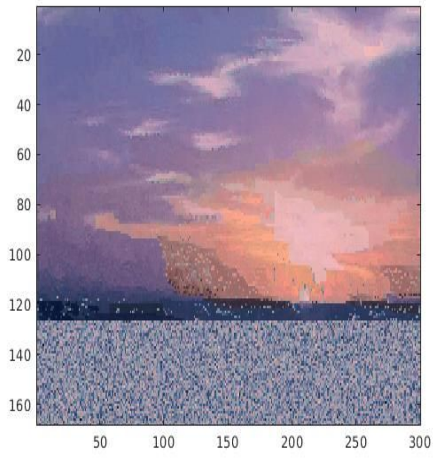
Random Search



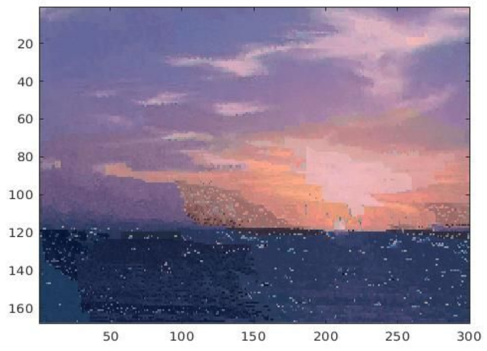
After 1/4rth iteration



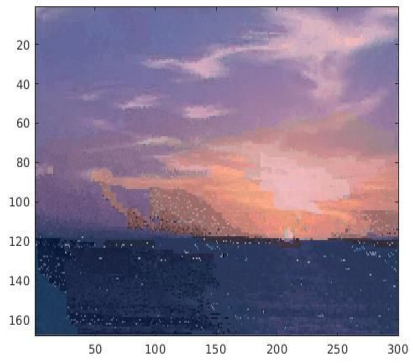
After 3/4rth iteration



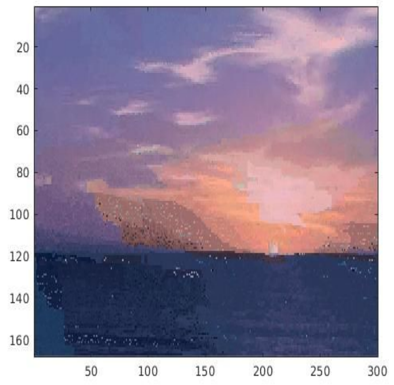
After 1st iteration



After 2nd iteration

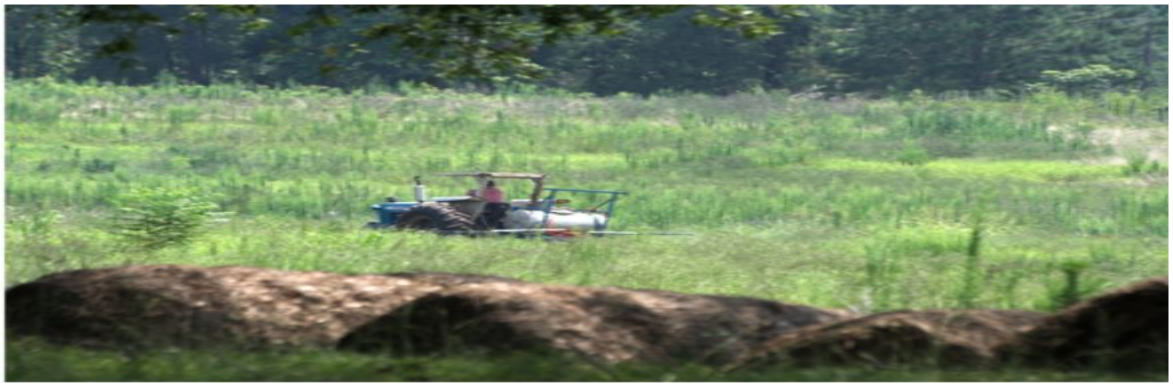


After 4rth iteration

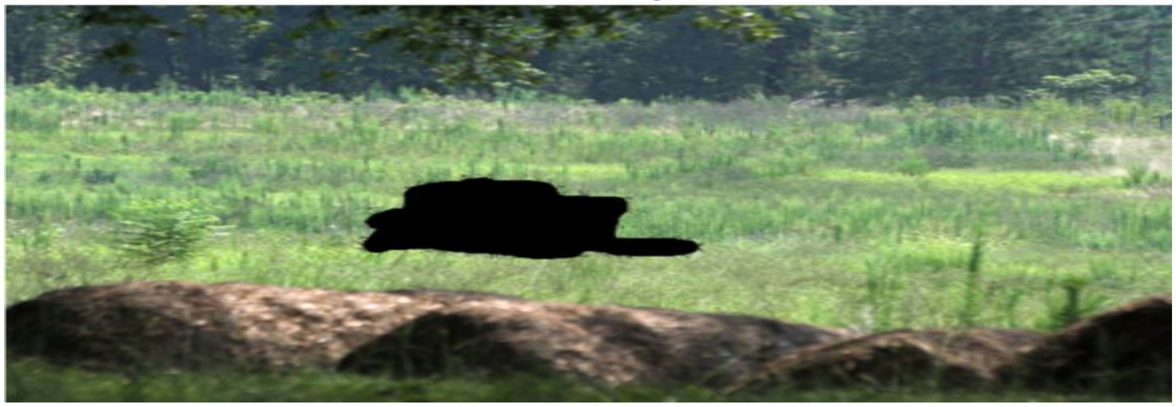


The above failure case is due to the inability to match with precision details and textures.

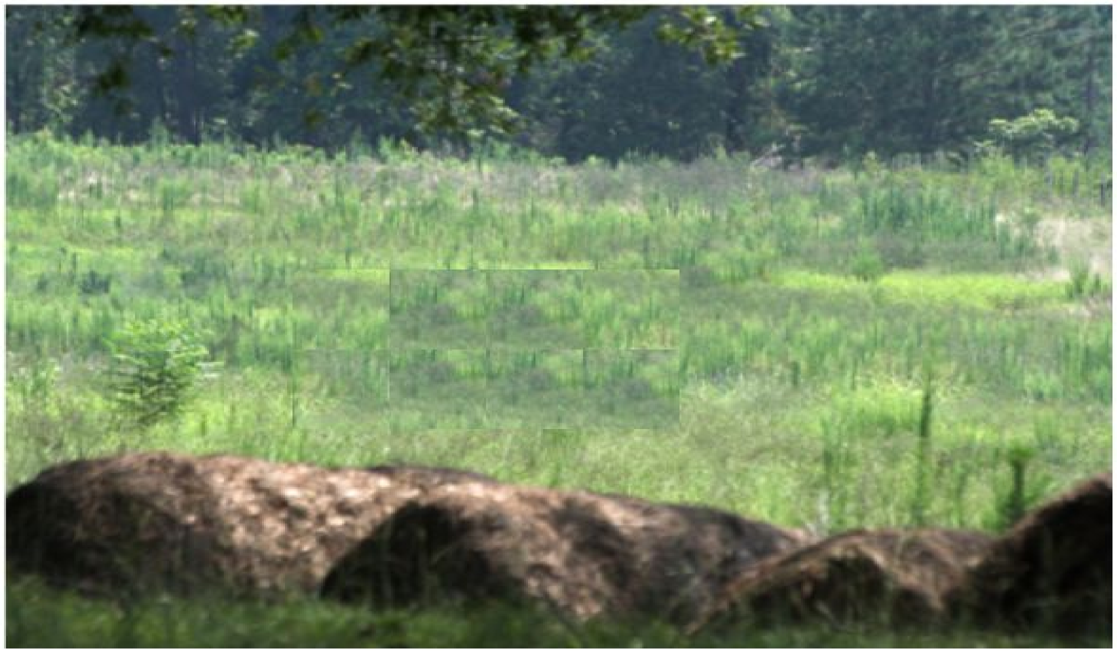
Hole Filling Using Patch Match (ANNF Algorithm)



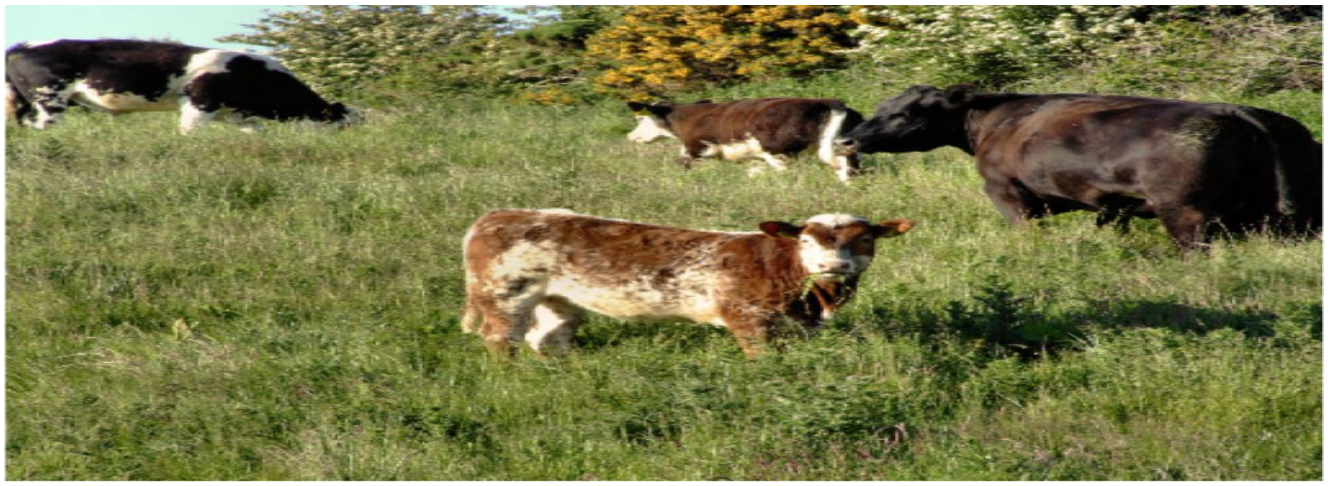
Input image

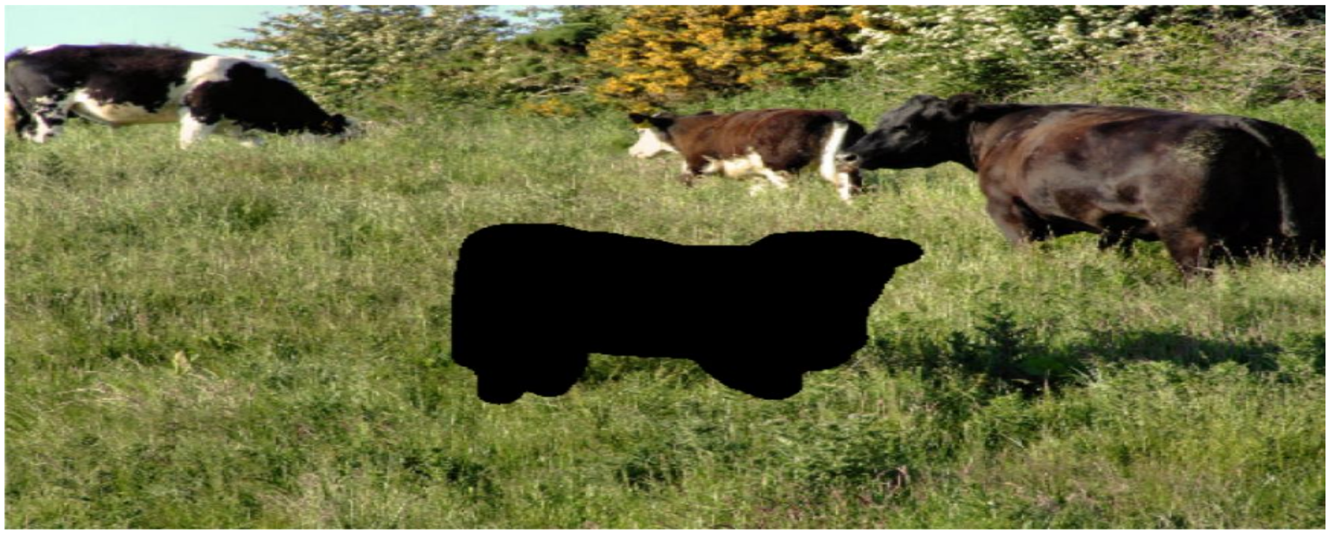


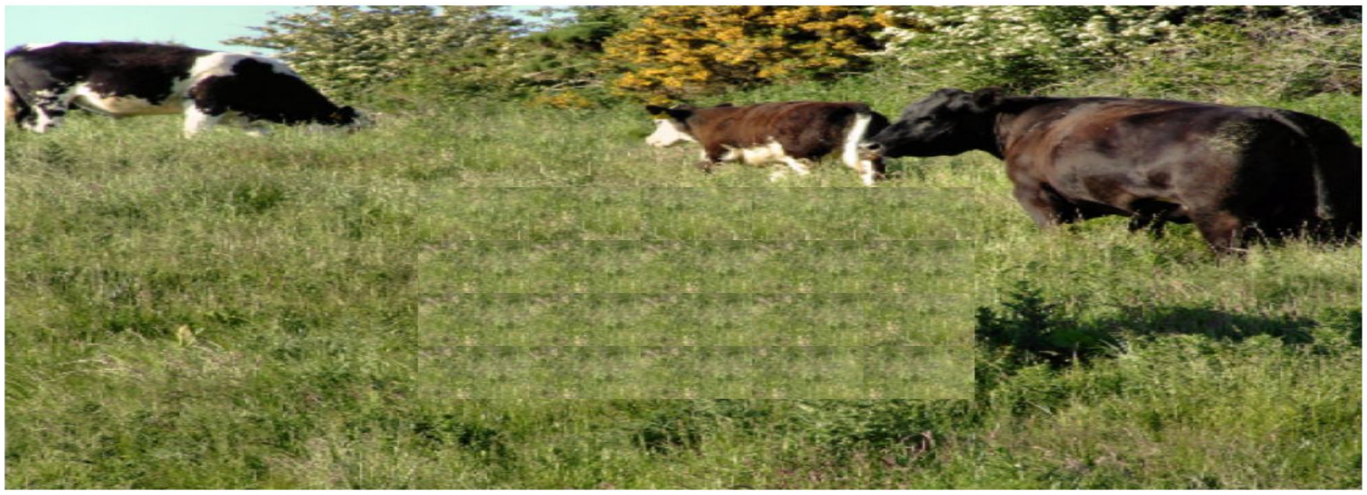
Mask Image

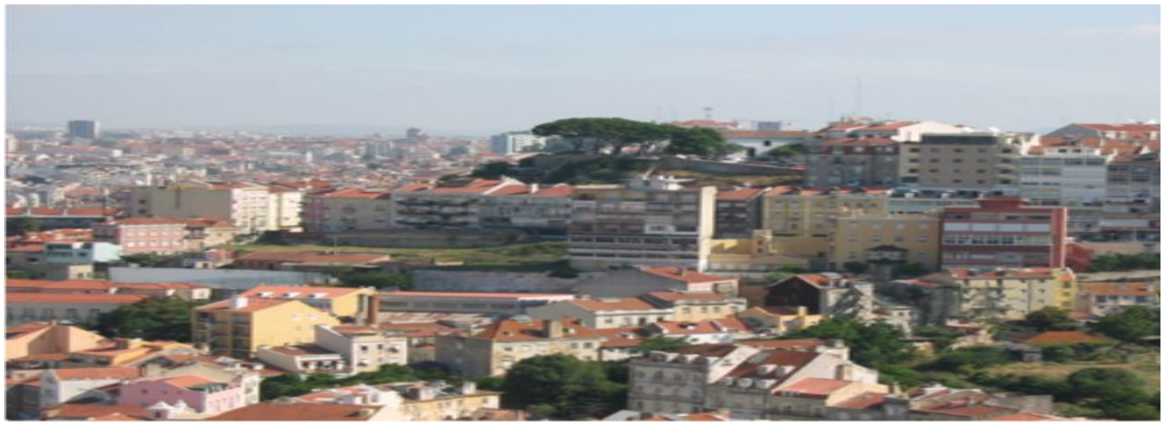


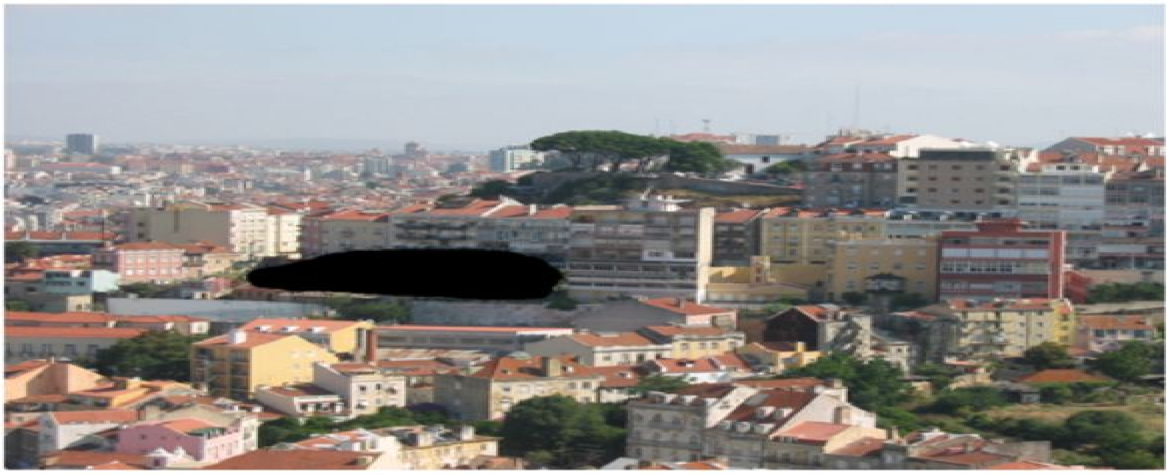
Hole filled image

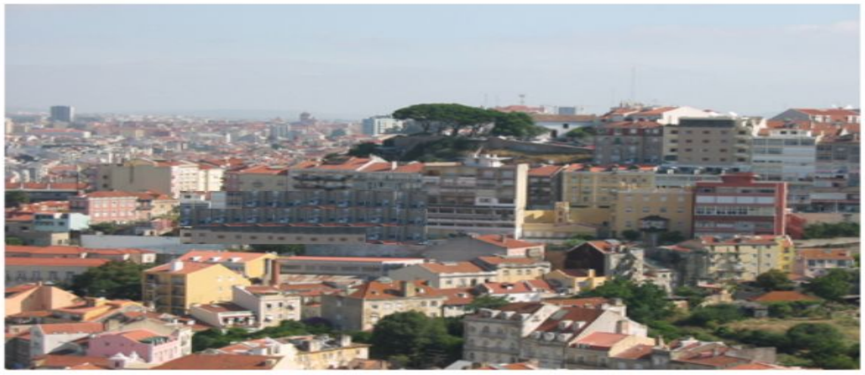

Input Image

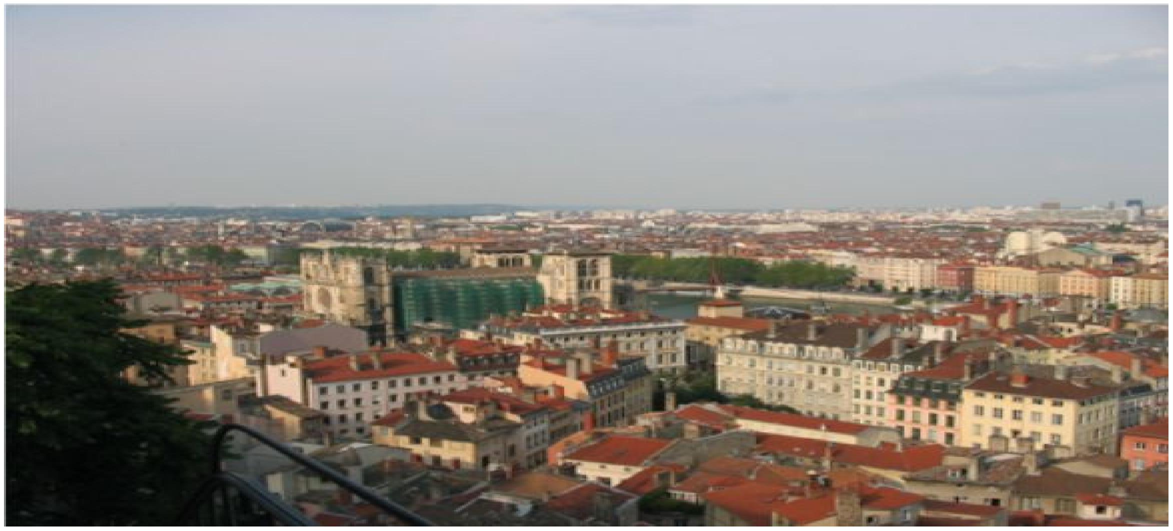

Mask Image

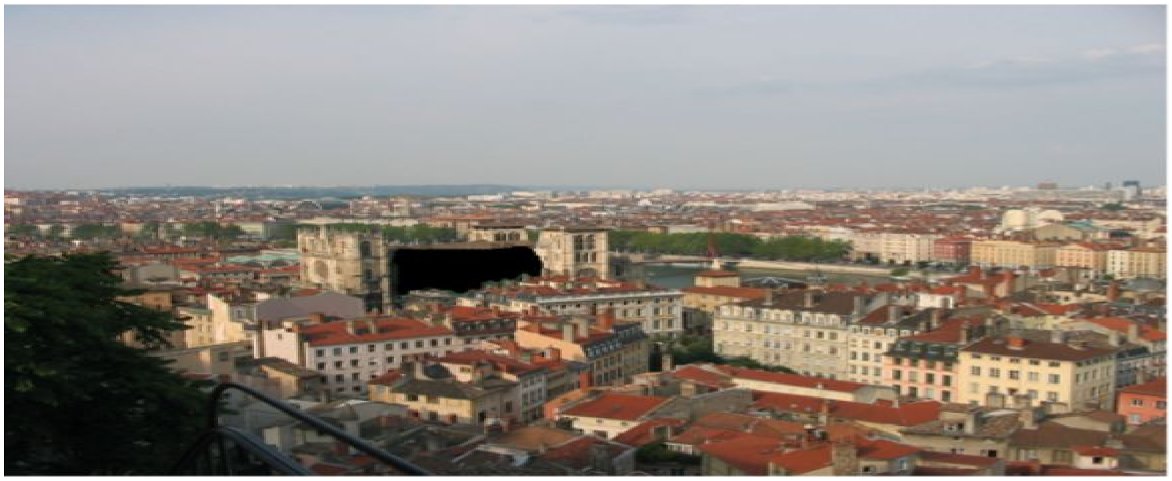

Hole filled image

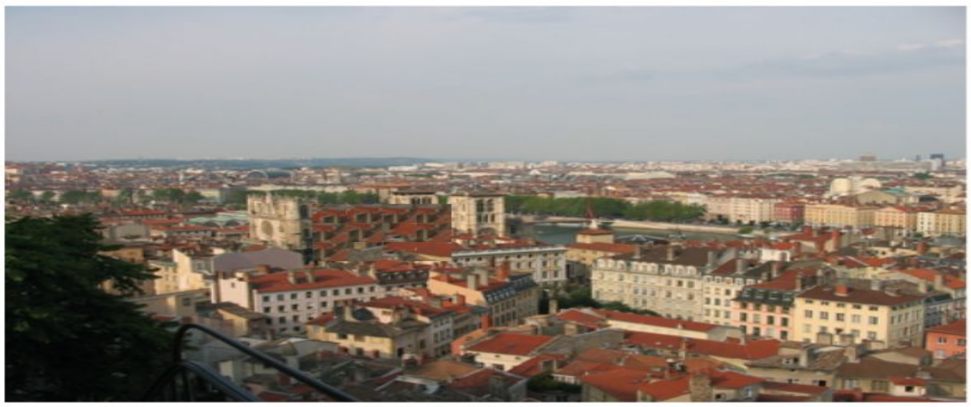

Input Image


Mask Image


Hole Filled Image


Input Image


Mask Image



Output image

Discussions:

- To see the outputs run textcode.m and testcode2.m giving appropriate inputs.
- We observe that clarity of the reconstructed image is dependent upon the content of the input image.
- The quality of the image also depends on window size .(To overcome this use patch voting)
- While reconstructing the image the patches with high amount of texture and detail are not matched properly sometimes.
- NNF can be implemented in various tools like image denoising,image retargeting,image filling,image reshuffling
- **Image retargeting(DEformation Constraints):**
    - Many recent retargeting methods allow the user to mark semantically important regions to be used with other automatically detected cues
    - One important cue that has been overlooked by previous methods are the lines and objects with straight edges that are so common in images of both man-made scenes (indoor photos, buildings, roads) and in natural scenes (tree trunks, horizon lines).
    - Keeping such lines straight is important to produce plausible outputs.
    - However, marking a line as an important region in existing techniques usually forces the line to appear in its entirety in the output image, but does not guarantee that the line will not bend or break .
    - Moreover often we do not care if the line gets shorter or longer in the output, as long as it remains straight
    - To accomplish this, we extend the BDS formulation from a free optimization to a constrained optimization problem, by constraining the domain of possible nearest neighbor locations in the output for certain subsets of input points
    - $\arg\min d_{BDS} \quad \text{s.t.} \quad \mathcal{M}_k(p_i, NN(p_i), q_j, NN(q_j)) = 0$ where p_i=Pixels in the constrained region q_i=PIxels corresponding to p_i in the NNF.
    - where we have K models Mk(k∈1...K) to satisfy. The meaning of Mk() = 0 is as follows: in the case of lines, satisfying a model means that the dot product of the line 3-vector (l, in homogeneous coordinates) and all the points in the output image should be zero, i.e.,NN(pi)Tl=0. In the case of regions, we limit ourselves here to 2D homography transformations (H) and therefore satisfying the model means that the distance of all projected points and the corresponding NN points in the output

image should be zero, i.e., Hkpi−NN(pi) =0.

- **Image Reshuffling(Hard Constraints):**
  - The model constraints described in the previous section usually succeed in preserving lines and regions.
  - However, in difficult cases, with large scale factors or when there are contradictions between the various constraints they cannot all be well satisfied automatically.
  - In other cases, such as in image reshuffling the user may want to strictly define the location of a region in the output as a hard constraint on the optimization.
  - This can be easily done in our framework, by fixing the NN fields of the relevant region points to the desired offsets according to these hard constraints.
  - After each EM iteration we simply correct the offsets to the output position, so that the other regions around the objects gradually rearrange themselves to align with these constrained regions.
  - For an object that moves substantially from its original output location, we give three intuitive options to the user to determine the initialization of the contents of the hole before the optimization starts: swap, in which the system simply swaps the pixels between the old and new locations; interpolate, in which the system smoothly interpolates the hole from the boundaries and clone , in which the system simply clones the object in its original location.
  - For small translations these all generate similar outputs, but for large objects and large motions these options lead to entirely different results
- **Image Filling (Search Space Constraints):**
  - Image completion of large missing regions is a challenging task.
  - we can adopt to same user interaction approach,allowing the user to draw curves across the missing region.
  - The curves can have different labels (represented in our interface using different colors) to indicate propagation of different structures.
  - This is accomplished by limiting the search space for labeled pixels inside the hole to the regions outside the hole with the same label.(Paradoxically, adding these extra constraints accelerates the convergence properties by limiting the search

space.)

Github Link : https://github.com/Shritishma/DIP

Task assignment:

1)Sriya Deepika(20161186) :

- Propagation part in the algorithm(Propagation.m)
- Construction of NNF from image A and image B(PatchMatch3.m)
- Finding NNF for the hole filling(PatchMatch4.m)

2) Shritishma Reddy(20161165):-

- Random Search part in the algorithm(RandomSearch.m)
- Proving convergence by reconstructing A from B (textcode.m)
- Hole filling of the image using NNF(textcode2.m)