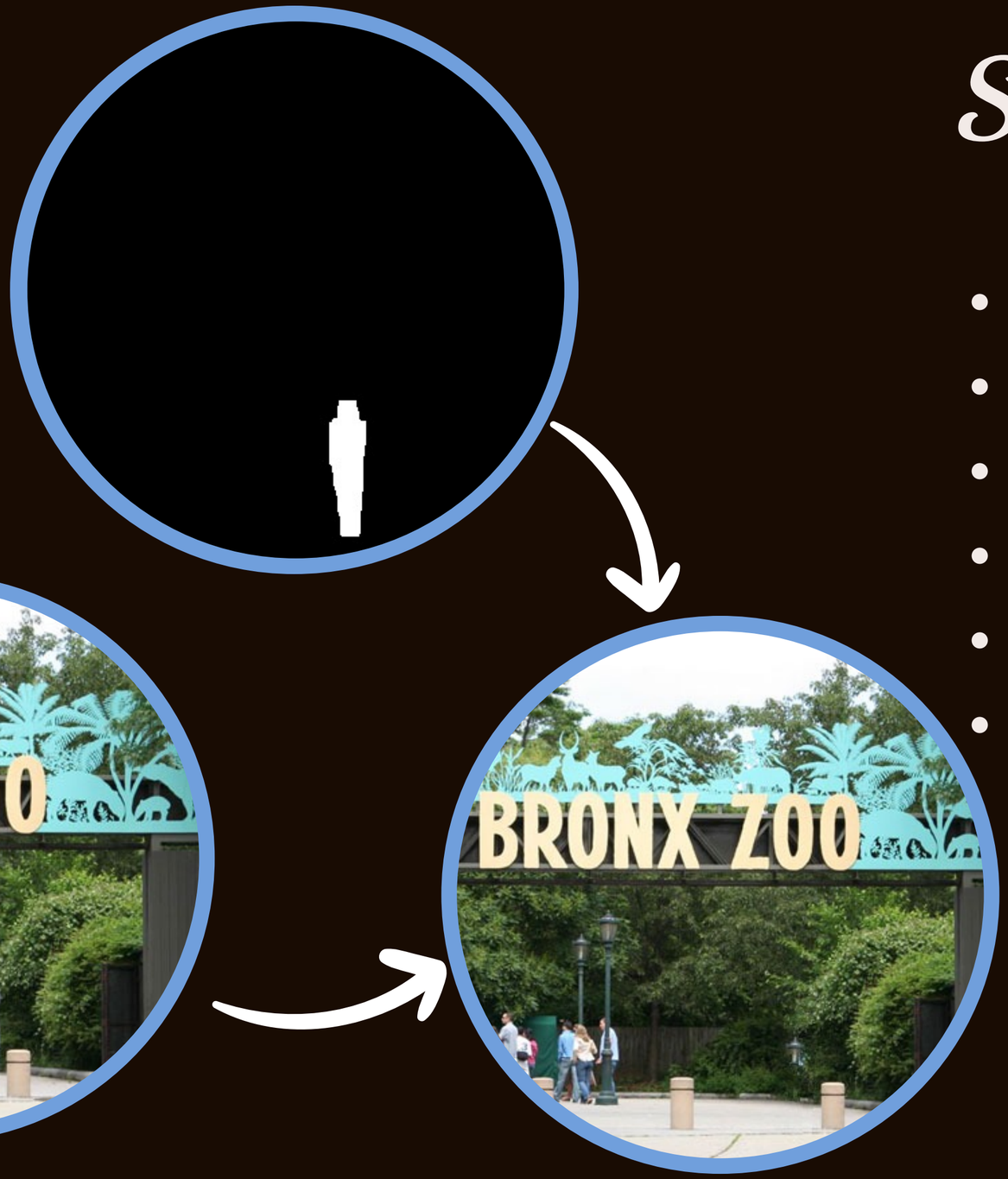
IMAGE INPAINTING

~ DSP MINI PROJECT

Akshara EE21B005
Mahalakshmi EE21B029



Synopsis

01

- What is image inpainting?
- Objective
- Code Overview
- Algorithm steps
- Advantages
- Conclusion

What is image inpainting ?

Image inpainting is a technique used in image processing to fill in missing or damaged parts of an image. This technique is used for various purposes like object removal, image restoration, etc.,
There are various inpainting algorithms such as:

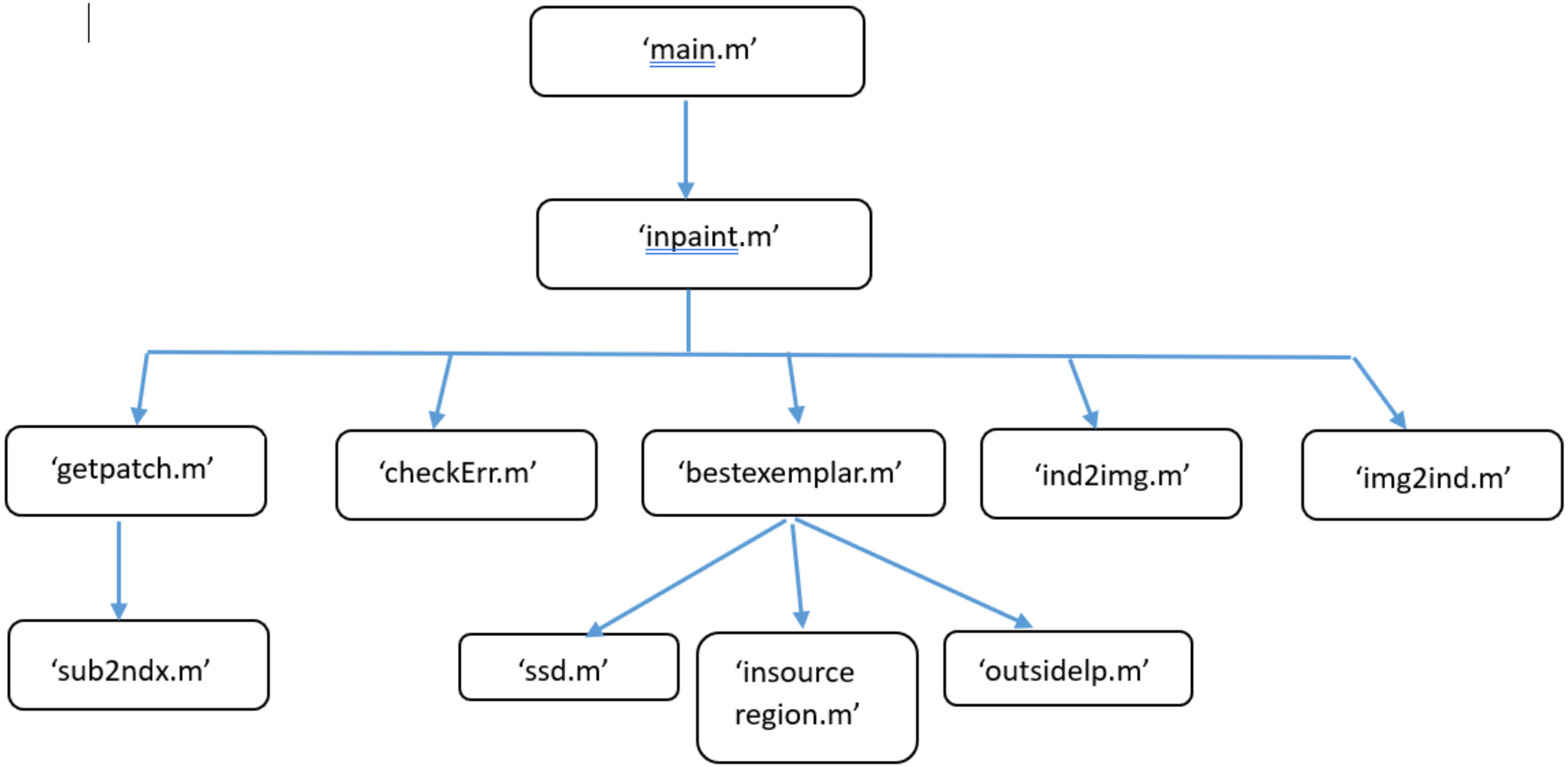
- exemplar-based inpainting
- diffusion-based inpainting
- patch-based inpainting

These algorithms use information from the surrounding areas of the missing or damaged regions to generate an inpainted result.

Objective

The primary objective of the project is to describe an exemplar-based approach for image inpainting. The aim is to introduce a technique that looks at the nearby parts of an image (exemplars) and uses that information to smoothly fill in areas that need to be taken out or fixed.

Key functions:



1. Main Code

```
clear all;
close all;

% imagefilename =input('Select image you want to inpaint: ');
% maskfilename  = input('Select mask : ');
% psz = input('Select patch size(odd scalar) : ');

% imagefilename = 'images/air1.png';
% maskfilename  = 'images/air2.png';
% psz = 9;
%
% inpaintedImg = inpaint(imagefilename,maskfilename,psz);
%
% imshow(uint8(inpaintedImg));
% imwrite(uint8(inpaintedImg),'images/air3.png')
%

imagefilename = 'images/walk.png';
maskfilename  = 'images/walk2.png';
psz = 9;


inpaintedImg = inpaint(imagefilename,maskfilename,psz);
imshow(uint8(inpaintedImg));
```

Overview

Input Parameters:
imagefilename: Path to the input image.
maskfilename: Path to the binary mask indicating regions to be inpainted.
psz: Patch size (odd scalar).

This code snippet initializes variables and inputs for image inpainting. It uses a predefined image ("walk.png"), mask ("walk2.png"), and patch size (9). The `inpaint` function is called with these parameters, producing an inpainted image, which is then displayed. The code serves as an example of the inpainting process on a specific image with fixed settings.

2. Inpaint

```
function [inpaintedImg] = inpaint(imagefilename,maskfilename,psz)

    Img = imread(imagefilename);
    mask = double(imread(maskfilename)/255);
    nout=4;

    % Check if mask and patch size are valid
    checkErr(mask,psz);


    fillRegion = mask;

    Img = double(Img);
    auxImg = Img;
    indexes = img2ind(auxImg);
    sz = [size(auxImg,1) size(auxImg,2)];
    sourceRegion = ~fillRegion;

    % Initialize isophote values
    [Ix(:,:,3), Iy(:,:,3)] = gradient(auxImg(:,:,3));
    [Ix(:,:,2), Iy(:,:,2)] = gradient(auxImg(:,:,2));
    [Ix(:,:,1), Iy(:,:,1)] = gradient(auxImg(:,:,1));
    Ix = sum(Ix,3)/(3*255); Iy = sum(Iy,3)/(3*255);
    temp = Ix; Ix = -Iy; Iy = temp;  % Rotate gradient 90 degrees

    % Initialize confidence and data terms
    C = double(sourceRegion);
    D = repmat(-.1,sz);
    iter = 1;

% %     Seed 'rand' for reproducible results (good for testing)
%     rand('state',0);

    % Loop until entire fill region has been covered
    var = 1;

    while any(fillRegion(:))    %while each element is nonzero(the regione to be
                                %inpainted is filled at all)

      iteration = var
      var = var + 1;

      %Find contour(making convolution with laplacian filter we find edge) & norm
      fillRegionD = double(fillRegion);
      [Nx,Ny] = gradient(double(~fillRegion));
      N = [Nx(contour(:)) Ny(contour(:))];
      N = normr(N);
      N(~isfinite(N))=0; % handle NaN and Inf

      % Compute confidences along the fill front
      for k = contour'
        Hp = getPatch(sz,k,psz);
        q = Hp(~(fillRegion(Hp)));
        C(k) = sum(C(q))/numel(Hp);
      end

      % Compute data term
      D(contour) = abs(Ix(contour).*N(:,1)+Iy(contour).*N(:,2)) + 0.001;%

      % Compute patch priorities = confidence term * data term
      priorities = C(contour).* D(contour);

      % Find patch with maximum priority, Hp
      [~,ndx] = max(priorities(:));
      p = contour(ndx(1));
      [Hp,rows,cols] = getPatch(sz,p,psz);
      toFill = fillRegion(Hp);

      imshow(uint8(auxImg));figure
        inpaintedMovie(iter) = im2frame(uint8(auxImg));

      % Find exemplar that minimizes error, Hq
      Hq = bestexemplar(auxImg,auxImg(rows,cols,:),toFill',sourceRegion,rows,cols

      % Update fill region
      toFill = logical(toFill);
      fillRegion(Hp(toFill)) = false;

      % Propagate confidence & isophote values
      C(Hp(toFill))  = C(p);
      Ix(Hp(toFill)) = Ix(Hq(toFill));
      Iy(Hp(toFill)) = Iy(Hq(toFill));

      % Copy image data from Hq to Hp
      indexes(Hp(toFill)) = indexes(Hq(toFill));
      auxImg(rows,cols,:) = ind2img(indexes(rows,cols),Img);
```

The inpainting process is iterative and involves filling the target region by selecting patches with the highest priority. This priority is determined by considering both confidence and data terms. The confidence term reflects the reliability of existing information, while the data term assesses the similarity between gradients.
During each iteration, the function updates isophote values, confidence values, and image data. This iterative process continues until the entire target region is covered, resulting in a final inpainted image.

```
function [Hp,rows,cols] = getPatch(sz,p,psz)
    % Returns the indices for a 9x9 patch centered at pixel p.
    % [x,y] = ind2sub(sz,p);  % 2*w+1 == the patch size

    w=(psz-1)/2; p=p-1; y=floor(p/sz(1))+1; p=rem(p,sz(1)); x=floor(p)+1;
    rows = max(x-w,1):min(x+w,sz(1));
    cols = (max(y-w,1):min(y+w,sz(2)))';
    Hp = sub2ndx(rows,cols,sz(1));
return;
```

3. getPatch

The `getPatch` function takes the image size (`sz`), pixel index (`p`), and patch size (`psz`) as inputs. It calculates the row and column indices (`rows` and `cols`) to form a square patch of size `psz` centered around the specified pixel. Using half of the patch size (`w`) prevents indices from exceeding image boundaries. The output, `Hp`, consists of linear indices representing the pixels within the computed patch.

4. Sub2ndx

`sub2ndx` converts row and column indices (`rows` and `cols`) to linear indices in Matlab. It uses matrix operations to transform coordinate-style indices into a single vector of linear indices, considering the total number of rows in the original matrix (`nTotalRows`).

```
function N = sub2ndx(rows,cols,nTotalRows)
    %Converts the (rows,cols) subscript-style indices
    %indices

    X = rows(ones(length(cols),1),:);
    Y = cols(:,ones(1,length(rows)));
    N = X+(Y-1)*nTotalRows;
return;
```

5. best Exemplar


```
function Hq = bestexemplar(img,Ip,toFill,sourceRegion,rows,cols)


    m=size(Ip,1);
    mm=size(img,1);
    n=size(Ip,2);
    nn=size(img,2);
    patchErr=0.0;
%   err=0.0;
    bestErr=1000000000.0;
    best = zeros(1,4);%1x4 vector that will contain the coordinates of the most similar patch Iq

    %foreach patch
    N=nn-n+1;%300-9+1
    M=mm-m+1;%300-9+1

    Ip = rgb2lab(Ip);
    img = rgb2lab(img);

    %sliding window over the image
    for j = 1:M %1:282 ROWS
        J=j+m-1;
        for i = 1:N %1:282 COLS
            I = i+n-1;

            %check that sliding window is inside source region and outside Ip
            if(inSourceRegion(j,J,i,I,sourceRegion) && outsideIp(rows,cols,j,J,i,I))

                patchErr = ssd(Ip,img(j:J,i:I,:),toFill);

                if (patchErr <= bestErr) %Update
                    bestErr = patchErr;
                    best(1,1) = j; best(1,2) = J;
                    best(1,3) = i; best(1,4) = I;

                end

            end
        end

    end

    Hq = sub2ndx(best(1):best(2),(best(3):best(4))',mm);

return;
```

This function looks for the best match in an image (img) for a given target patch (Ip). It scans through the entire image, comparing color information in the Lab color space. The goal is to find a patch that closely matches the target in both color and location while excluding already filled regions (toFill). The result, a set of indices (Hq), represents the best-matching area in the image. This process is part of fixing damaged or missing parts in an image.

6. checkErr

```
function checkErr(mask,psz)
    %Check some error: if mask is a matrix and if it is valid and
    %patch size is odd

    if ~ismatrix(mask); error('Invalid mask'); end
    if sum(sum(mask~=0 & mask~=1))>0; error('Invalid mask'); end
    if mod(psz,2)==0; error('Patch size psz must be odd.'); end
```

The function ensures that the input mask is a valid binary matrix with elements either 0 or 1 and that the patch size is an odd scalar. If any of these conditions is not met, an error is raised to alert the user about the issue.

7. ssd

The `ssd` function calculates the Sum of Squared Differences (SSD) between two RGB image patches (`Ip` and `patch`). It converts the images to double precision and computes squared differences for each color channel. Contributions from pixels in the fill region are excluded. The function returns the mean of these channel-wise SSD values.

```
function [ssdout] = ssd(Ip,patch,toFill)
    Ip=double(Ip);
    patch=double(patch);
    ssdout = 0;

    if(size(Ip,3)==3)
        C = zeros(1,3);

        for k=1:3
                X = (patch(:,:,k)-Ip(:,:,k)).^2;
                X(logical(toFill))=0;
                ssd = sum(X(:));
                C(1,k) = ssd;
        end
        ssdout = mean(C);
    end
end
```

```
function [isinsourceregion] = inSourceRegion(j,J,i,I,sourceRegion)
    %Return 1 true if patch is in sourceRegion
    %return 0 if patch is outside or touch targetregion

    isinsourceregion = 1;
    for k = j:J
        for z = i:I
            if(sourceRegion(k,z)==0)
                    isinsourceregion = 0;
            end
        end
    end
end
```

8. insource region

The `inSourceRegion` function checks if a specified patch, defined by boundaries (j, J, i, I), is entirely within the source region (`sourceRegion`). It returns 1 if the patch is wholly within the source region and 0 if any part extends outside or touches the target region. In essence, it assesses whether the patch is fully sourced from available information or extends into the area needing inpainting.

9. outsideIp

`outsideIp` checks if a sliding window (j, J, i, I) is entirely outside a specified region (`Ip`). It returns 1 for complete outside and 0 for any overlap. This ensures the sliding window considered during inpainting doesn't overlap with the region to be inpainted.

```
function [isoutsideIp] = outsideIp(rows,cols,j,J,i,I)
    %Return 1 true if the sliding window is outside Ip
    %return 0 false if the sliding window is inside Ip and so it has not
    %verified(is inside = touch or contain)

    isoutsideIp = 1;
    if((ismember(j,rows) || ismember(J,rows)) && ...
            ((ismember(i,cols) || ismember(I,cols))))
        isoutsideIp = 0;
    end
end
```

```
function img2 = ind2img(ind,img)
    %Converts an indexed image into an RGB image, using 'img' as a colormap

    for i=3:-1:1, temp=img(:,:,i); img2(:,:,i)=temp(ind); end

return
```

10. ind2Img

`ind2img` transforms an indexed image (`ind`) into an RGB image using another image (`img`) as a colormap. The resulting RGB image (`img2`) assigns colors to pixels based on the colormap values corresponding to the indices in the original indexed image.

11. img2Ind

The img2ind function converts a three-dimensional RGB image (img) into an indexed image, using the image itself as the colormap. It determines the image size (s) and reshapes linear indices from 1 to the total pixel count. The function returns a two-dimensional matrix, where each element represents a pixel in the original image.

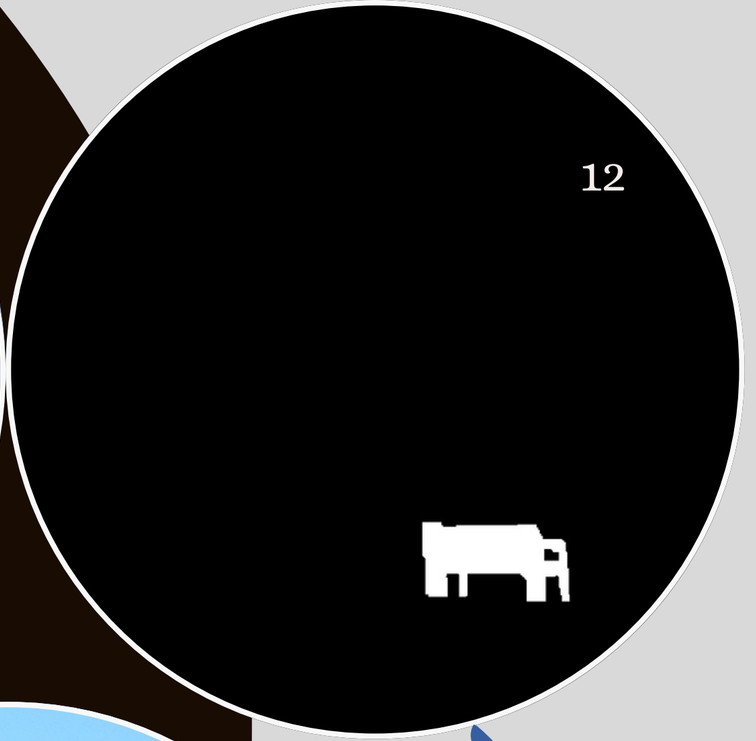
```
function ind = img2ind(img)
    % Converts an RGB image into a indexed image, using th
    % the colormap.

    s=size(img);
    ind=reshape(1:s(1)*s(2),s(1),s(2));

return;
```

Algorithm steps

- Initialization
- Preprocessing
- Gradient Computation
- Confidence and Data Initialization
- Iterative Inpainting
- Output



ADVANTAGES

Crucial for addressing concerns related to misinformation and fake content.

Allows for the removal of sensitive or undesirable content from an image

It allows photographers to enhance their images by removing distracting elements or imperfections,

In forensic analysis, it can be used to reconstruct obscured or tampered regions of an imaget

Can be applied to remove unwanted objects or artifacts from video frames,

It aids in the reconstruction of medical images with missing data.

Conclusion

Image inpainting is a powerful technique with numerous applications across various domains.
Its ability to fill in missing or damaged regions in images, while maintaining visual consistency, contributes to tasks ranging from artistic editing to scientific analysis.
we can say that exemplar-based inpainting stands out as an efficient and reliable tool for enhancing and repairing digital images.

THANK YOU