Digital Image Processing

Lab- Assignment

M.Sc. Computer Science, 3rd Sem, 2022-23

Registration No. : 2080002 of 2021-2022

Roll: 90/MCS NO.210015

Session: 2021-2023

Paper Code: MCS-DSE-31

Sample1.pgm

Sample2.pgm

Above two pictures have been used across the assignment.

Programming language used: MATLAB 2021a

CHAPTER-2

1. Apply linear and non-linear operations on sub image.

Let H is a general operator produces an output image g (x, y) from a given input image, f(x,y):

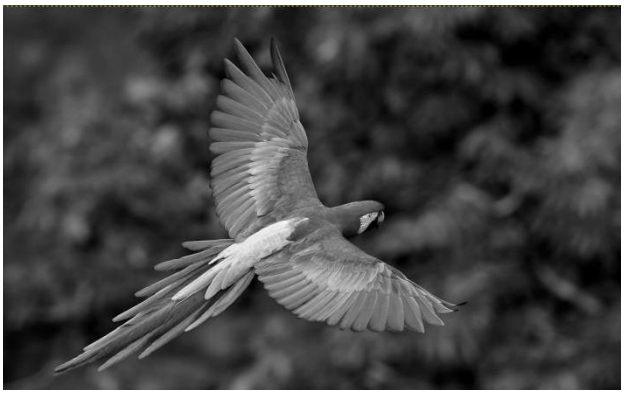
$H[f(x,y)]=g(x,y)$

$H[a.f1(x,y)+b.f2(x,y)]=a.H[f1(x,y)]+ b.H[f2(x,y)]$

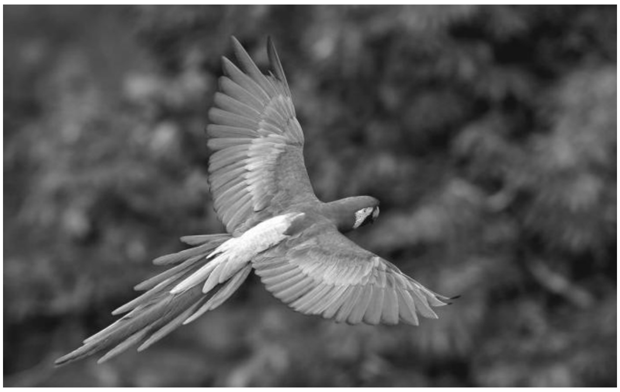
$=a.g1(x,y)+b.g2(x,y)$

e.g., Sum operator is a linear operator.

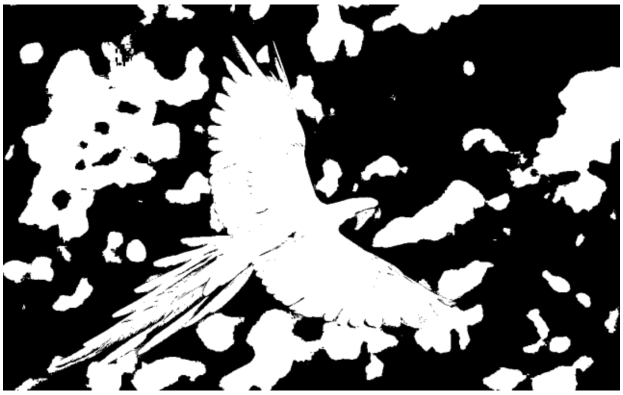
Max operator is a non-linear operator.



Original Image



Linear Operator Applied Image(Sum +20)



Non-linear Operator (Max)

Ans:

Source Code:

Linear Operator

```
x=imread("Sample.pgm");
m=size(x);

sum=zeros(m(1),m(2));

for i=1:m(1)
    for j=1:m(2)
        sum(i,j)=x(i,j)+20;
        if sum(i,j)>255
            sum(i,j)=255;
        end
    end
end
sum=uint8(sum);

imwrite(sum,"SumImage.pgm");
```

Non-linear Operator:

```
x=imread("Sample.pgm");
m=size(x);

max=zeros(m(1),m(2));
f1=zeros(m(1),m(2));
f2=zeros(m(1),m(2));
a=randi([0 2],1,1);
b=randi([0 2],1,1);

for i=1:m(1)
    for j=1:m(2)
        f1(i,j)=a*x(i,j);
    end
end
for i=1:m(1)
    for j=1:m(2)
        f2(i,j)=b*x(i,j);
    end
end
f1=uint8(f1);
f2=uint8(f2);
f1=scale(f1);
```

```
f2=scale(f2);

for i=1:m(1)
    for j=1:m(2)
        if f1(i,j)>f2(i,j)
            max(i,j)=f1(i,j);
        else
            max(i,j)=f2(i,j);
        end
    end
end
max=uint8(max);

imwrite(max,"MaxImage.pgm");
```

1. Apply Image Scaling: - The difference between two 8-bit images can range from a minimum of -255 to a maximum of 255 and the value of the sum of two such images can range from 0 to 510.

1) $gm=g-\min(g)\rightarrow$ creates an image where minimum value is 0.

2) $g_s=[g_m \max(g_m)/]$ ❼ creates a scaled image g_s where values are in the range [0, K], for 8-bit image, K = 255.

Answer:

```
x=importdata('sample.pgm');
m=size(x);

c=zeros(m(1),m(2));
x=double(x);
min=x(1,1);
max=x(1,1);
for i=1:m(1)
    for j=1:m(2)
        if x(i,j)<min
            min=x(i,j);
        end
        if x(i,j)>max
            max=x(i,j);
        end
    end
end

for i=1:m(1)
    for j=1:m(2)
        c(i,j)=255*((x(i,j)-min)/(max-min));
    end
end
c=uint8(c);

imwrite(c,"Scaled.pgm");
```

Original Image

Scaled Image

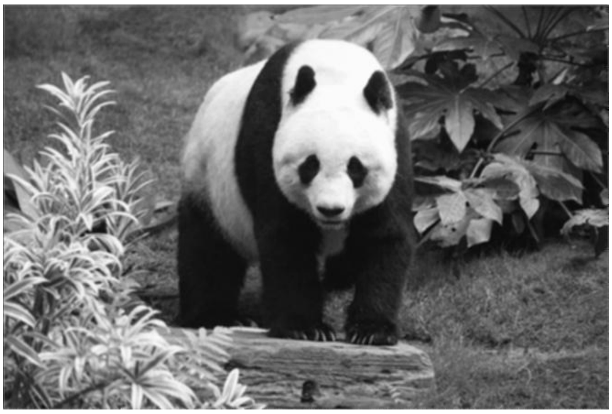
2. Neighbourhood Operation: -

$(x,y)= 1m.n\ \Sigma f(r,c)_{(r,c)\in Sxy}$

Answer:

```
image=imread("sample2.pgm");
m=size(image);
zero_padded_image=padarray(double(image),[1 1],0);
zero_padded_smooth_image=image;

for i=2:m(1)
    for j=2:m(2)
        sum=zero_padded_image(i-
1,j)+zero_padded_image(i+1,j)+zero_padded_image(i,j-
1)+zero_padded_image(i,j+1)+zero_padded_image(i-1,j-
1)+zero_padded_image(i+1,j+1)+zero_padded_image(i+1,j-1)+zero_padded_image(i-
1,j+1)+zero_padded_image(i,j);
        zero_padded_smooth_image(i-1,j-1)=sum/9;
    end
end
zero_padded_image=uint8(zero_padded_image);
zero_padded_smooth_image=uint8(zero_padded_smooth_image);
imwrite(zero_padded_smooth_image,"zero_padded_smooth_image.pgm");
```

Original Image



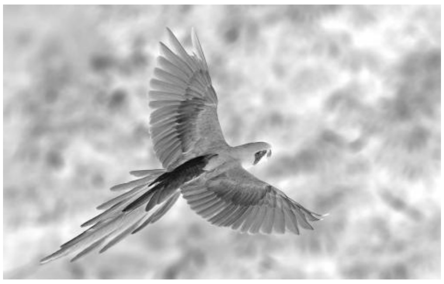
Neighbourhood Operation(Box Filter) Applied

3. Intensity transformation function to obtain negative of an image

Answer:

```
x=importdata('sample.pgm');

m=size(x);
neg=zeros(m(1),m(2));
size(neg);
for i=1:m(1)
    for j=1:m(2)
        neg(i,j)=255-x(i,j);
    end
end
```

Original Image



Negative Image

```
neg=uint8(neg);
imwrite(neg,"Negative.pgm");
```

4. Given a sub-image (binary), select two-pixel positions, check whether they are 4-adjacent, 8-adjacent, m-adjacent. Check connectivity between two pixels. Check the digital path between two pixels.

Answer:

4-Adjaceny

```
x=randi([0 1],6,6);
m=size(x);
x=imbinarize(x);
x
p1x=2
p1y=3

p2x=2
p2y=4

if(check4Adjacency(p1x,p1y,p2x,p2y))&& x(p1x,p1y)==x(p2x,p2y)
    fprintf("\n\tYes! 4 Adjacent!\n");
else
    fprintf("\n\tNo! Not 4 Adjacent!\n");
end
function[flag]=check4Adjacency(p1x,p1y,p2x,p2y)
    flag=false;
    if ((p1x+1==p2x) || (p1x-1==p2x)) && p1y==p2y
        flag=true;
        return;
    end
    if((p1y+1==p2y) || (p1y-1==p2y)) && p1x==p2x
        flag=true;
    end
end
```

```
>> FourAdjacency
x =
  6×6 logical array
   1   1   1   0   1   0
   0   0   0   0   0   0
   0   1   1   0   0   0
   1   1   0   1   0   1
   1   1   1   1   1   1
   1   1   0   0   1   1
p1x =
     2
p1y =
     3
p2x =
     2
p2y =
     4
            Yes! 4 Adjacent!
```

8-Adjacency

```
x=randi([0 1],6,6);
m=size(x);
x=imbinarize(x);
x
p1x=2
p1y=3

p2x=3
p2y=4

if((check8Adjacency(p1x,p1y,p2x,p2y))&& x(p1x,p1y)==x(p2x,p2y))
    fprintf("\n\tYes! 8 Adjacent!\n");
else
    fprintf("\n\tNo! Not 8 Adjacent!\n");
end
function[flag]=check8Adjacency(p1x,p1y,p2x,p2y)
    flag=false;
    if(check4Adjacency(p1x,p1y,p2x,p2y))
        flag=true;
    else
        flag=checkDiagonalAdjacency(p1x,p1y,p2x,p2y);
    end
end
function[flag]=checkDiagonalAdjacency(p1x,p1y,p2x,p2y)
    flag=false;
    if (p1x-1==p2x && p1y-1==p2y) || (p1x-1==p2x && p1y+1==p2y)|| (p1x+1==p2x && p1y-1==p2y)||(p1x+1==p2x && p1y+1==p2y)
          flag=true;
    end
end
function[flag]=check4Adjacency(p1x,p1y,p2x,p2y)
    flag=false;
    if ((p1x+1==p2x) || (p1x-1==p2x)) && p1y==p2y
        flag=true;
        return;
    end
    if((p1y+1==p2y) || (p1y-1==p2y)) && p1x==p2x
        flag=true;
    end
end
```

```
>> EightAdjacency
x =
  6×6 logical array
   0   1   1   0   1   0
   1   0   1   0   0   1
   1   1   1   1   0   1
   0   0   1   0   0   1
   0   1   1   1   1   1
   0   0   0   0   0   0
p1x =
     2
p1y =
     3
p2x =
     3
p2y =
     4
            Yes! 8 Adjacent!
```

m-adjacency

```
x=randi([0 1],6,6);
x(1,4)=1;
x(2,5)=1;
x(2,4)=0;
x(1,5)=0;
m=size(x);
x=imbinarize(x);
x
p1x=1
p1y=4

p2x=2
p2y=5

if((checkMAdjacency(p1x,p1y,p2x,p2y,x)))
    fprintf("\n\tYes! M Adjacent!\n");
else
    fprintf("\n\tNo! Not M Adjacent!\n");
end
function[flag]=checkMAdjacency(p1x,p1y,p2x,p2y,x)
    flag=false;
    m=size(x);
    if(check4Adjacency(p1x,p1y,p2x,p2y)&& x(p1x,p1y)==x(p2x,p2y))
        flag=true;
        return;
    else
        if(checkDiagonalAdjacency(p1x,p1y,p2x,p2y)&& x(p1x,p1y)==x(p2x,p2y))
                p1Neighbours=getFourNeighbours(p1x,p1y)
                p2Neighbours=getFourNeighbours(p2x,p2y)
                for i=1:4
                    if ~isnan(p1Neighbours(i,1)) &&~isnan(p1Neighbours(i,2)) &&
x(p1Neighbours(i,1),p1Neighbours(i,2))==x(p1x,p1y)
                        for j=1:4
                            if ~isnan(p2Neighbours(j,1))&&~isnan(p2Neighbours(j,2))&&p1Neighbours(i,1)==p2Neighbours(j,1) &&
p1Neighbours(i,2)==p2Neighbours(j,2)
                                'ok'
                                flag=false;
                                return;
                            end
                        end
                    end
                end
                 flag=true;

        else
            flag=false;
        end
    end
end

function[neighbours]=getFourNeighbours(p1x,p1y)
    neighbours=zeros(4,2);
    neighbours(1,1)=p1x;
    neighbours(1,2)=p1y-1;
    neighbours(2,1)=p1x;
    neighbours(2,2)=p1y+1;
    neighbours(3,1)=p1x-1;
    neighbours(3,2)=p1y;
    neighbours(4,1)=p1x+1;
    neighbours(4,2)=p1y;
    for i=1:4
        for j=1:2
            if neighbours(i,j)<1
                neighbours(i,j)=NaN;
            end
        end
    end
%      neighbours=sort(neighbours);
end
function[flag]=checkDiagonalAdjacency(p1x,p1y,p2x,p2y)
    flag=false;
    if (p1x-1==p2x && p1y-1==p2y) || (p1x-1==p2x && p1y+1==p2y)|| (p1x+1==p2x && p1y-1==p2y)||(p1x+1==p2x && p1y+1==p2y)
           flag=true;
    end
end
function[flag]=check4Adjacency(p1x,p1y,p2x,p2y)
    flag=false;
    if ((p1x+1==p2x) || (p1x-1==p2x)) && p1y==p2y
        flag=true;
        return;
    end
    if((p1y+1==p2y) || (p1y-1==p2y)) && p1x==p2x
        flag=true;
    end
end
```

```
>> MAdjacency
x =
 6×6 logical array
  0  0  0  1  0  1
  0  0  0  0  1  0
  1  1  1  1  1  1
  0  1  0  0  0  1
  0  1  0  1  0  0
  0  0  0  0  1  0
p1x =
   1
p1y =
   4
p2x =
   2
p2y =
   5
p1Neighbours =
   1    3
   1    5
  NaN    4
   2    4
p2Neighbours =
   2    4
   2    6
   1    5
   3    5
          Yes! M Adjacent!
```

5. Given a binary image, find out the number of connected components.

6. Find Euclidean, D_4, D_8, D_m distances between two pixels with co-ordinates given

```
D4 Distance
x=imread("sample.pgm");
m=size(x);

x=imbinarize(x);
p1x=randi([1 m(1)],1,1);
p1y=randi([1 m(2)],1,1);

p2x=randi([1 m(1)],1,1);
p2y=randi([1 m(2)],1,1);

fprintf("\nPixel 1 : (x,y): (%d,%d) ",p1x,p1y);
fprintf("\nPixel 2 : (x,y): (%d,%d) ",p2x,p2y);

d4Distance=findD4Distance(p1x,p1y,p2x,p2y);

fprintf("\nD4 Distance between the pixels is: %f\n",d4Distance);

function[d4Distance]=findD4Distance(p1x,p1y,p2x,p2y)
    d4Distance=abs(p1x-p2x)+abs(p1y-p2y);
end
```

```
>> D4Distance


Pixel 1 : (x,y): (313,344)

Pixel 2 : (x,y): (356,573)

D4 Distance between the pixels is: 272.000000
```

```
D8 Distance
x=randi([0 1],8,8);
m=size(x);

x=imbinarize(x);
p1x=randi([1 m(1)],1,1);
p1y=randi([1 m(2)],1,1);

p2x=randi([1 m(1)],1,1);
p2y=randi([1 m(2)],1,1);

fprintf("\nPixel 1 : (x,y): (%d,%d) ",p1x,p1y);
fprintf("\nPixel 2 : (x,y): (%d,%d) ",p2x,p2y);

d8Distance=findD8Distance(p1x,p1y,p2x,p2y);

fprintf("\nD8 Distance between the pixels is: %f\n",d8Distance);

function[d8Distance]=findD8Distance(p1x,p1y,p2x,p2y)
    d8Distance=max(abs(p1x-p2x),abs(p1y-p2y));
end
```

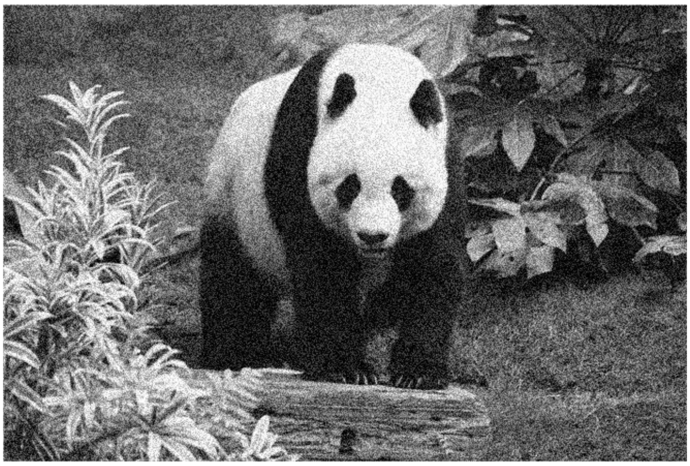
```
>> D8Distance


Pixel 1 : (x,y): (8,7)

Pixel 2 : (x,y): (1,3)

D8 Distance between the pixels is: 7.000000
```

7. Generate noise matrix which is uncorrelated and average value is zero. Generate at least ten noisy images.

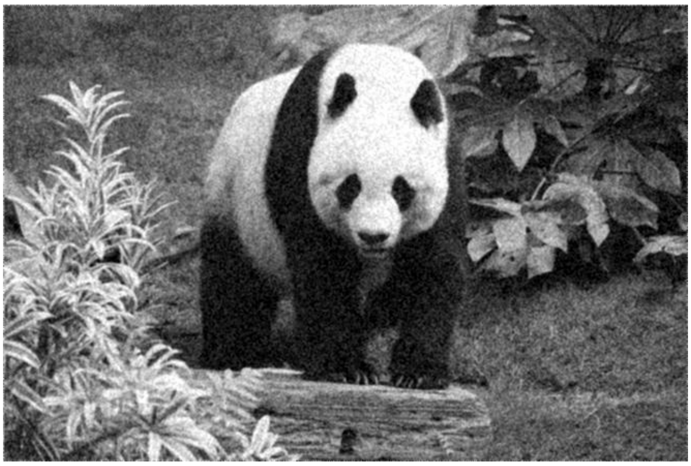
$$g(x,y)=f(x,y)+\eta(x,y)$$

Apply averaging to resolve noise present in the image.

```
x=imread("sample2.pgm");
% x=imresize(x,[500 500]);
m=size(x)
imshow(x)

x=double(x);
 for n=1:10
    filename = strcat('noise-', num2str(n), '.pgm');
    noisy_image=x+generateNoiseMatrix(m);
    imwrite(uint8(noisy_image),filename);
 end
    smooth_image=generateSmoothMatrix(noisy_image,m);
    smooth_image=uint8(smooth_image);
    filename = strcat('smooth-', num2str(n), '.pgm');
    imwrite(smooth_image,filename);

 function[smooth_image]=generateSmoothMatrix(noisy_image,m)
    noisy_image=double(noisy_image);
     smooth_image = noisy_image;
    for i=2:m(1)-1
        for j=2:m(2)-1
            sum=noisy_image(i-1,j)+noisy_image(i+1,j)+noisy_image(i,j-
1)+noisy_image(i,j+1)+noisy_image(i-1,j-1)+noisy_image(i+1,j+1)+noisy_image(i+1,j-
1)+noisy_image(i-1,j+1)+noisy_image(i,j);
            smooth_image(i,j)=sum/9;
        end
    end
 end


function[noise_matrix]=generateNoiseMatrix(m)
    noise_matrix=randi([-50 50],m(1),m(2));
    noise_matrix=noise_matrix-round(mean(noise_matrix(:)));
end
```



Noisy Image 1



Smooth Image 1

8. Perform Logical Operations i.e., OR, AND, XOR between two images.
   Answer:
   OR

```
x1=imread("sample.pgm");
m=size(x1);

x2=imread("sample2.pgm");
n=size(x2);
x1=imbinarize(x1);
x2=imbinarize(x2);
if m(1)>n(1)
    m1=n(1);
else
    m1=m(1);
end
if m(2)>n(2)
    n1=n(2);
else
    n1=m(2);
end

orImg=zeros(m1,n1);


for i=1:m1
    for j=1:n1
        orImg(i,j)=or(x1(i,j),x2(i,j));

    end
end

for i=1:m1
    for j=1:n1
        if(orImg(i,j)==1)
            orImg(i,j)=255;
        end
    end
end
% orImg=uint8(orImg);
imshow(orImg)

imwrite(orImg,"orImage.pgm");
```

Or Image

AND

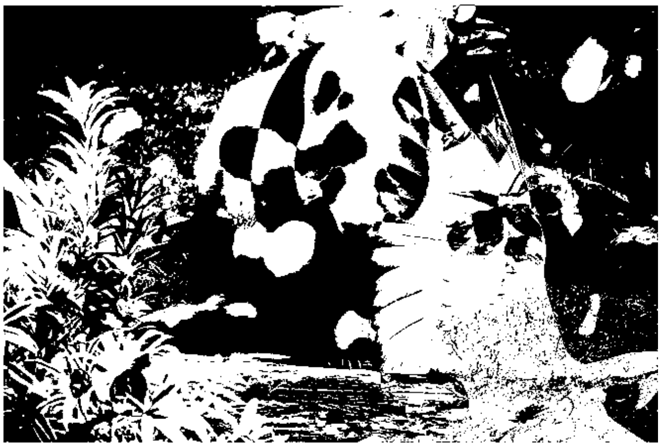
```
x1=imread("sample.pgm");
m=size(x1);

x2=imread("sample2.pgm");
n=size(x2);
x1=imbinarize(x1);
x2=imbinarize(x2);
if m(1)>n(1)
    m1=n(1);
else
    m1=m(1);
end
if m(2)>n(2)
    n1=n(2);
else
    n1=m(2);
end

andImg=zeros(m1,n1);
```

And Image

```
for i=1:m1
    for j=1:n1
        andImg(i,j)=and(x1(i,j),x2(i,j));

    end
end

for i=1:m1
    for j=1:n1
        if(andImg(i,j)==1)
            andImg(i,j)=255;
        end
    end
end
% orImg=uint8(orImg);
imshow(andImg)

imwrite(andImg,"andImage.pgm");
```

XOR

```
x1=imread("sample.pgm");
m=size(x1);

x2=imread("sample2.pgm");
n=size(x2);
x1=imbinarize(x1);
x2=imbinarize(x2);
if m(1)>n(1)
    m1=n(1);
else
    m1=m(1);
end
if m(2)>n(2)
    n1=n(2);
else
    n1=m(2);
end

xorImg=zeros(m1,n1);


for i=1:m1
    for j=1:n1
        xorImg(i,j)=xor(x1(i,j),x2(i,j));

    end
end

for i=1:m1
    for j=1:n1
        if(xorImg(i,j)==1)
            xorImg(i,j)=255;
        end
    end
end
% orImg=uint8(orImg);
imshow(xorImg)

imwrite(xorImg,"xorImage.pgm");
```



XOR Image

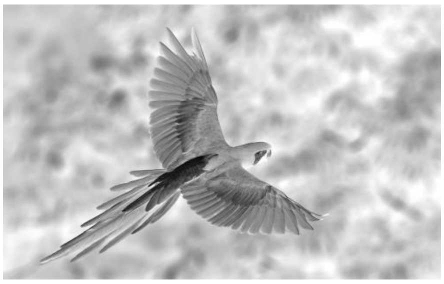
CHAPTER-3

1. Compute negative image: $S = L - 1 - r$

Answer:

```
x=importdata('sample.pgm');

m=size(x);
neg=zeros(m(1),m(2));
size(neg);
for i=1:m(1)
    for j=1:m(2)
        neg(i,j)=255-x(i,j)
    end
end
```

Original Image



Negative Image

2. Compute log transformation: $S = c.log(1 + r)$, where c is a positive constant.

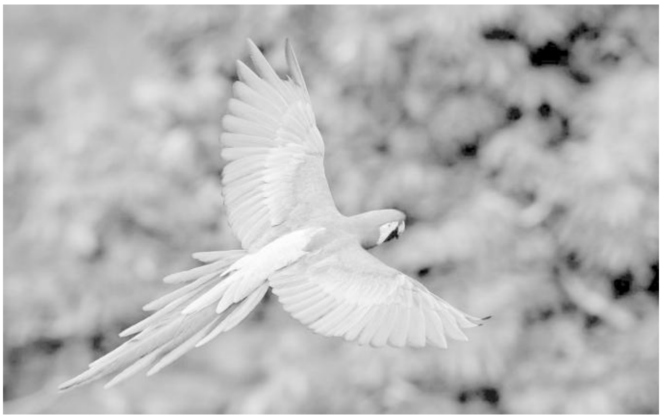
→ maps narrow range of low intensity values of the input into a wide range of output levels.

→ we use this type of transformation to expand the values of dark pixels in an Image, while compressing the higher-level values.

Answer:

```
x=imread('sample.pgm');
m=size(x);
x=double(x);
min=x(1,1);
max=x(1,1);
for i=1:m(1)
    for j=1:m(2)
        if x(i,j)<min
            min=x(i,j);
        end
        if x(i,j)>max
            max=x(i,j);
        end
    end
end

s=zeros(m(1),m(2));
for i=1:m(1)
    for j=1:m(2)
        s(i,j)=(255/(log(1+max))*log((1+x(i,j)));
    end
end
s=uint8(s);

imwrite(s,"Log.pgm");
```

Original Image



Log transformed image

3. Power-Law (Gamma) transformation:

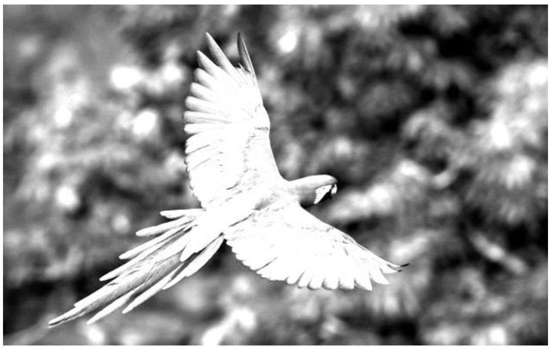
$S=c.r_{\gamma}$, where c and γ are positive constant.

If c = γ = 1 ❼it is identity transformation

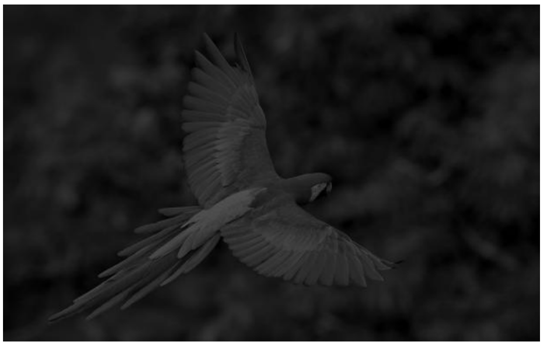
Answer:

```
x=imread('sample.pgm');
m=size(x);
x=double(x);


p=zeros(m(1),m(2));
for i=1:m(1)
    for j=1:m(2)
        p(i,j)=1*(x(i,j)^1.2);
    end
end

p=uint8(p);
imwrite(p,"Gamma.pgm");
```

Original Image



Gamma Corrected with γ=1.2

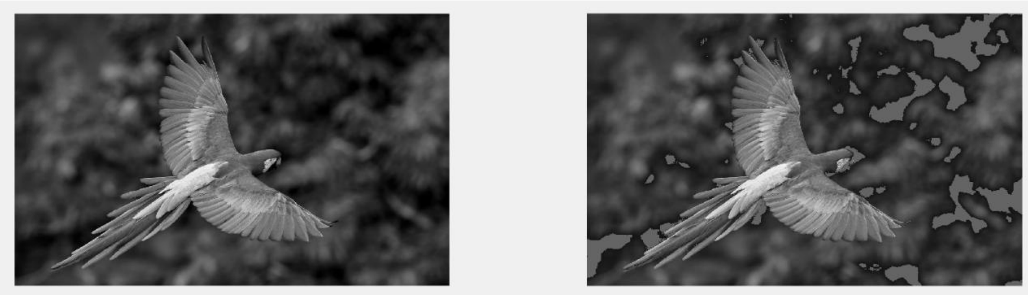


Gamma Corrected with γ=0.8

4. Piecewise Linear transformation:

Contrast stretching, Intensity level slicing

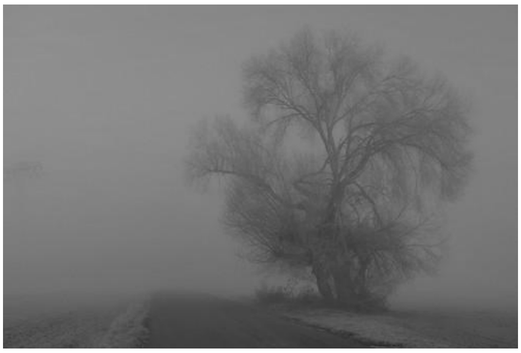
```
Answer:
x=imread("sample.pgm");
m=size(x);
sum=0;
for i=1:m(1)
    for j=1:m(2)
        sum=sum+x(i,j);
    end
end
average=sum/(m(1)*m(2));
y=x;
for i=1:m(1)
    for j=1:m(2)
        if x(i,j)>=(average-20) && x(i,j)<=average+20
            y(i,j)=average+100;
        end
    end
end

subplot(1,2,1),imshow(x);
subplot(1,2,2),imshow(y);
```

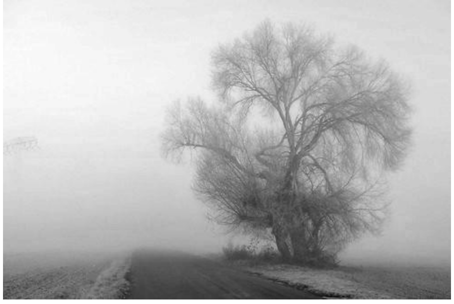


Intensity Level Slicing

Contrast Stretching

```
x=importdata('sample4.pgm');
m=size(x);

c=zeros(m(1),m(2));
x=double(x);
min=x(1,1);
max=x(1,1);
for i=1:m(1)
    for j=1:m(2)
        if x(i,j)<min
            min=x(i,j);
        end
        if x(i,j)>max
            max=x(i,j);
        end
    end
end

for i=1:m(1)
    for j=1:m(2)
        c(i,j)=255*((x(i,j)-min)/(max-min));
    end
end
c=uint8(c);

imwrite(c,"Scaled.pgm");
```


Original Image


Contrast Stretched Image

5. Bit-plane slicing: Here an 8-bit image may be considered as being composed of eight one-bit planes.

128 × bit plane 8 + 64 × bit plane 7 +…+… = grey scaled image

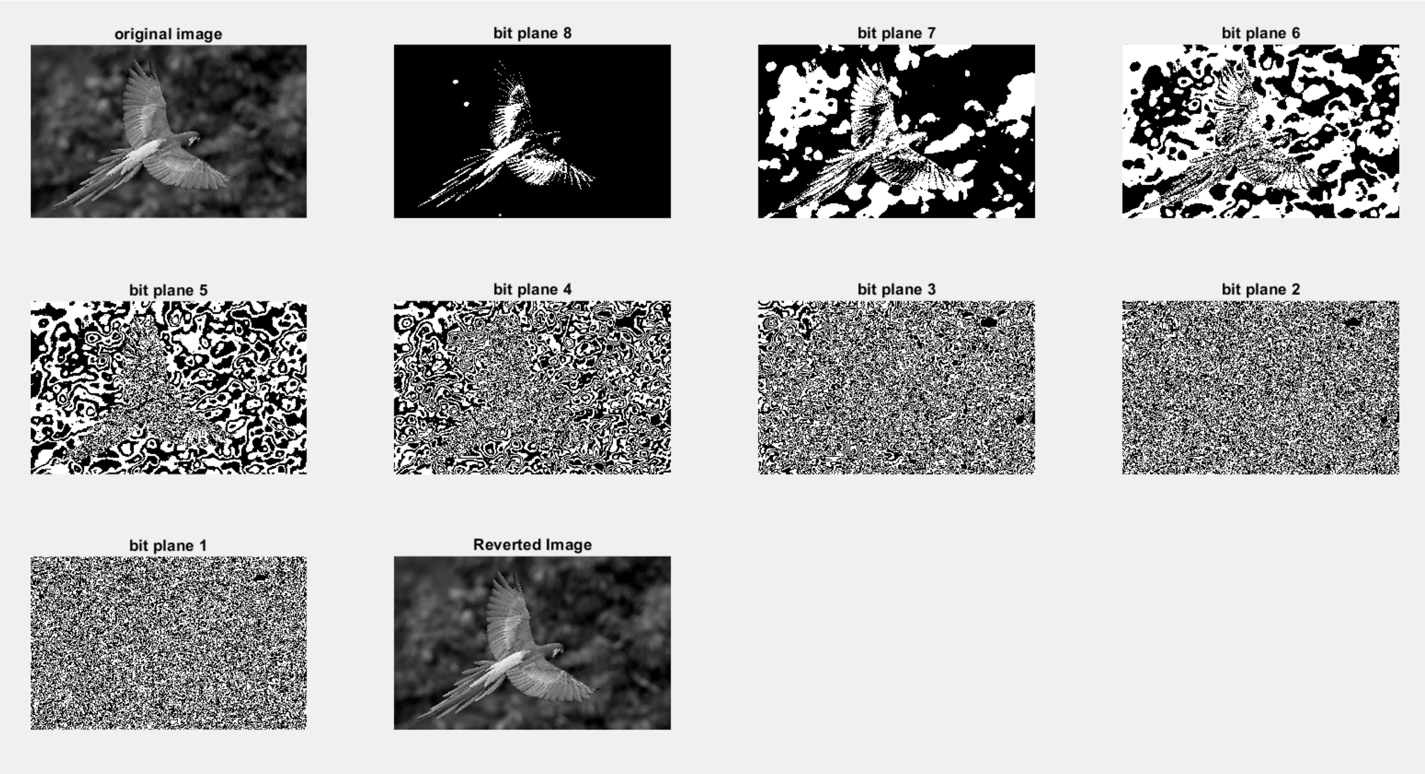
How to obtain 8th bit plane: - $T = 127$

$g(x,y) = 0$, if $f(x,y) \le T$

= 1, if $f(x,y) > T$

Answer:

```
x=imread('sample.pgm');
m=size(x);
x=double(x);

b8=zeros(m(1),m(2));

for i=1:m(1)
    for j=1:m(2)
        if mod(floor(double(x(i,j))/(2^7)),2)==0
            b8(i,j)=0;
        else
             b8(i,j)=128;
        end
    end
end
% b8=imbinarize(b8);

b7=zeros(m(1),m(2));
for i=1:m(1)
    for j=1:m(2)
        if mod(floor(double(x(i,j))/(2^6)),2)==0
            b7(i,j)=0;
        else
             b7(i,j)=64;
        end
    end
end
% b7=imbinarize(b7);

b6=zeros(m(1),m(2));
for i=1:m(1)
    for j=1:m(2)
        if mod(floor(double(x(i,j))/(2^5)),2)==0
            b6(i,j)=0;
        else
             b6(i,j)=32;
        end
    end
end
% b6=imbinarize(b6);

b5=zeros(m(1),m(2));
for i=1:m(1)
    for j=1:m(2)
        if mod(floor(double(x(i,j))/(2^4)),2)==0
            b5(i,j)=0;
        else
             b5(i,j)=16;
        end
    end
end
% b5=imbinarize(b5);

b4=zeros(m(1),m(2));
for i=1:m(1)
    for j=1:m(2)
        if mod(floor(double(x(i,j))/(2^3)),2)==0
            b4(i,j)=0;
        else
             b4(i,j)=8;
        end
    end
end
% b4=imbinarize(b4);

b3=zeros(m(1),m(2));
for i=1:m(1)
    for j=1:m(2)
        if mod(floor(double(x(i,j))/(2^2)),2)==0
            b3(i,j)=0;
        else
             b3(i,j)=4;
        end
    end
end
% b3=imbinarize(b3);

b2=zeros(m(1),m(2));
for i=1:m(1)
    for j=1:m(2)
        if mod(floor(double(x(i,j))/(2^1)),2)==0
            b2(i,j)=0;
        else
             b2(i,j)=2;
        end
    end
end

% b2=imbinarize(b2);
b1=zeros(m(1),m(2));
for i=1:m(1)
    for j=1:m(2)
        if mod(floor(double(x(i,j))),2)==0
            b1(i,j)=0;
        else
             b1(i,j)=1;
        end
    end
end
% b1=imbinarize(b1);



subplot(3,4,1),imshow(x),title('original image');
subplot(3,4,2),imshow(b8),title('bit plane 8');
subplot(3,4,3),imshow(b7),title('bit plane 7');
subplot(3,4,4),imshow(b6),title('bit plane 6');
subplot(3,4,5),imshow(b5),title('bit plane 5');
subplot(3,4,6),imshow(b4),title('bit plane 4');
subplot(3,4,7),imshow(b3),title('bit plane 3');
subplot(3,4,8),imshow(b2),title('bit plane 2');
subplot(3,4,9),imshow(b1),title('bit plane 1');

revert=zeros(m(1),m(2));
for i=1:m(1)
    for j=1:m(2)
        revert(i,j)=b8(i,j)+b7(i,j)+b6(i,j)+b5(i,j)+b4(i,j)+b3(i,j)+b2(i,j)+b1(i,j);
    end
end

o=revert;
revert=uint8(o);

subplot(3,4,10),imshow(revert),title('Reverted Image');
```



6. To find histogram: -

r_k, for k = 0, 1, 2, …, L – 1, denotes the intensities of an L-level digital image, f(x,y). The unnormalized histogram of f is defined as

$h(rk)= nk$, for k = 0, 1, 2, …, L – 1.

The normalized histogram of f is defined as

$p(r_k)= h(r_k)/M.N= n_k/M.N$

```
Answer:
x=imread("Sample.pgm");
m=size(x);

h=zeros(1,256);
for i=1:m(1)
    for j=1:m(2)
        h(1,x(i,j)+1)=h(1,x(i,j)+1)+1;
    end
end
fprintf("\nUnnormalized Histogram:\n");
h
normHist=zeros(1,256);
for j=1:256
    normHist(1,j)=h(1,j)/(m(1)*m(2));
end
fprintf("\nNormalized Histogram:\n");
normHist
```

7. Histogram equalization or linearization transformation function: -

$sk=T(rk)= \Sigma pr(rj)$, k = 0, 1, 2, …, L – 1

$pr(rk)= nkM.N$

Show normalized transformation function, equalized histogram

Answer:

```
x=imread("Sample.pgm");
m=size(x);

h=zeros(1,256);
for i=1:m(1)
    for j=1:m(2)
        h(1,x(i,j)+1)=h(1,x(i,j)+1)+1;
    end
end

normHist=zeros(1,256);
for j=1:256
    normHist(1,j)=h(1,j)/(m(1)*m(2));
end
cdf=zeros(1,256);
for i=1:256
    if i==1
        cdf(1,i)=normHist(1,i)*255;
    else
        cdf(1,i)=cdf(1,i-1)+normHist(1,i)*255;
    end
end
equalisedHist=round(cdf);

mapping=cat(1,[0:255],equalisedHist)
```

8. Histogram matching: -

$sk=T(rk)=(L-1)\Sigma pr(rj)kj=0$

$G(zq)=(L-1)\Sigma pz(zq)= sk$

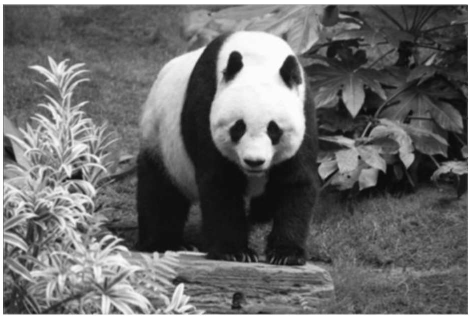
$zq= G-1(sk)$

9. Apply average filtering with 1) Box filter, 2) Gaussian filter and compare. Use Zero Padding, mirror padding, replicate padding

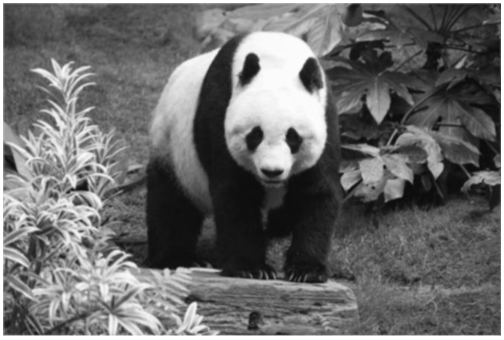
Answer:

```
image=imread("sample2.pgm");
m=size(image);
zero_padded_image=padarray(double(image),[1 1],0);
zero_padded_smooth_image=image;

for i=2:m(1)
    for j=2:m(2)
        sum=zero_padded_image(i-1,j)+zero_padded_image(i+1,j)+zero_padded_image(i,j-
1)+zero_padded_image(i,j+1)+zero_padded_image(i-1,j-1)+zero_padded_image(i+1,j+1)+zero_padded_image(i+1,j-
1)+zero_padded_image(i-1,j+1)+zero_padded_image(i,j);
        zero_padded_smooth_image(i-1,j-1)=sum/9;
    end
end
zero_padded_image=uint8(zero_padded_image);
zero_padded_smooth_image=uint8(zero_padded_smooth_image);
imwrite(zero_padded_smooth_image,"zero_padded_smooth_image.pgm");

replicate_padded_image=padarray(double(image),[1 1],'replicate');
replicate_padded_smooth_image=image;

for i=2:m(1)
    for j=2:m(2)
        sum=replicate_padded_image(i-1,j)+replicate_padded_image(i+1,j)+replicate_padded_image(i,j-
1)+replicate_padded_image(i,j+1)+replicate_padded_image(i-1,j-
1)+replicate_padded_image(i+1,j+1)+replicate_padded_image(i+1,j-1)+replicate_padded_image(i-
1,j+1)+replicate_padded_image(i,j);
        replicate_padded_smooth_image(i-1,j-1)=sum/9;
    end
end
replicate_padded_smooth_image=uint8(replicate_padded_smooth_image);
imwrite(replicate_padded_smooth_image,"replicate_padded_smooth_image.pgm");


mirror_padded_image=padarray(double(image),[1 1],'symmetric');
mirror_padded_smooth_image=image;

for i=2:m(1)
    for j=2:m(2)
        sum=mirror_padded_image(i-1,j)+mirror_padded_image(i+1,j)+mirror_padded_image(i,j-
1)+mirror_padded_image(i,j+1)+mirror_padded_image(i-1,j-
1)+mirror_padded_image(i+1,j+1)+mirror_padded_image(i+1,j-1)+mirror_padded_image(i-
1,j+1)+mirror_padded_image(i,j);
        mirror_padded_smooth_image(i-1,j-1)=sum/9;
    end
end
mirror_padded_smooth_image=uint8(mirror_padded_smooth_image);
imwrite(mirror_padded_smooth_image,"mirror_padded_smooth_image.pgm");
```

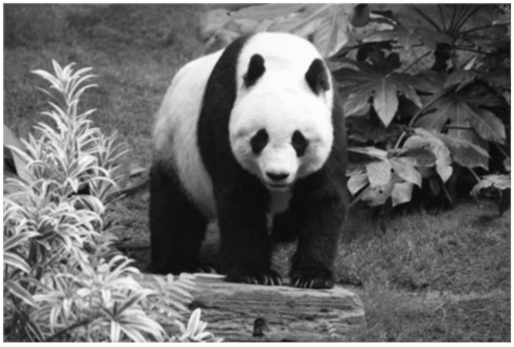
Original Image



Zero padded smooth image



Replica Padded Smooth Image



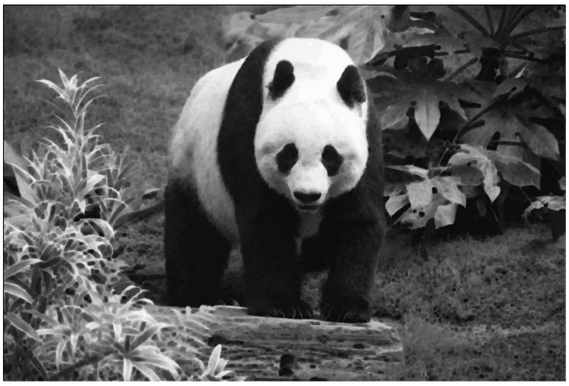
Mirror Padded Smooth Image

10. Apply median, min, max filter using above mention paddings

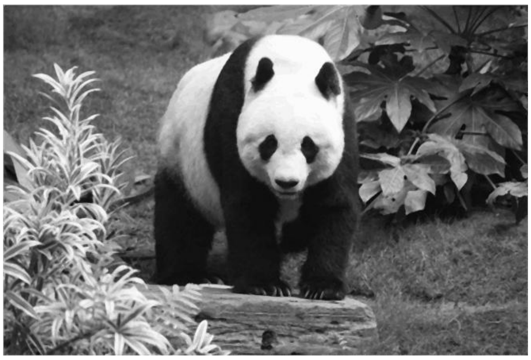
Answer:

```
image=imread("sample2.pgm");
m=size(image);
zero_padded_image=padarray(double(image),[1 1],0);
zero_padded_min_image=image;

for i=2:m(1)
    for j=2:m(2)
        arr=[zero_padded_image(i-1,j) zero_padded_image(i+1,j) zero_padded_image(i,j-1)
zero_padded_image(i,j+1) zero_padded_image(i-1,j-1) zero_padded_image(i+1,j+1)
zero_padded_image(i+1,j-1) zero_padded_image(i-1,j+1) zero_padded_image(i,j)];
        zero_padded_min_image(i-1,j-1)=min(arr);
    end
end
zero_padded_image=uint8(zero_padded_image);
zero_padded_min_image=uint8(zero_padded_min_image);
imwrite(zero_padded_min_image,"zero_padded_min_image.pgm");

replicate_padded_image=padarray(double(image),[1 1],'replicate');
replicate_padded_max_image=image;

for i=2:m(1)
    for j=2:m(2)
        arr=[replicate_padded_image(i-1,j) replicate_padded_image(i+1,j)
replicate_padded_image(i,j-1) replicate_padded_image(i,j+1) replicate_padded_image(i-
1,j-1) replicate_padded_image(i+1,j+1) replicate_padded_image(i+1,j-1)
replicate_padded_image(i-1,j+1) replicate_padded_image(i,j)];
        replicate_padded_max_image(i-1,j-1)=max(arr);

    end
end
replicate_padded_max_image=uint8(replicate_padded_max_image);
imwrite(replicate_padded_max_image,"replicate_padded_max_image.pgm");


mirror_padded_image=padarray(double(image),[1 1],'symmetric');
mirror_padded_median_image=image;

for i=2:m(1)
    for j=2:m(2)
        arr=[mirror_padded_image(i-1,j) mirror_padded_image(i+1,j)
mirror_padded_image(i,j-1) mirror_padded_image(i,j+1) mirror_padded_image(i-1,j-1)
mirror_padded_image(i+1,j+1) mirror_padded_image(i+1,j-1) mirror_padded_image(i-1,j+1)
mirror_padded_image(i,j)];
        mirror_padded_median_image(i-1,j-1)=median(arr);
    end
end
mirror_padded_median_image=uint8(mirror_padded_median_image);
imwrite(mirror_padded_median_image,"mirror_padded_median_image.pgm");
```

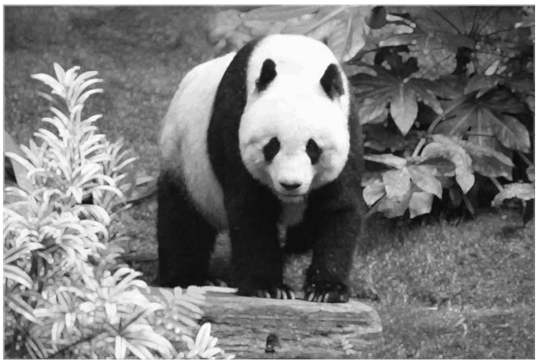
Original Image`



Zero Padded Min Image



Mirror-padded median image

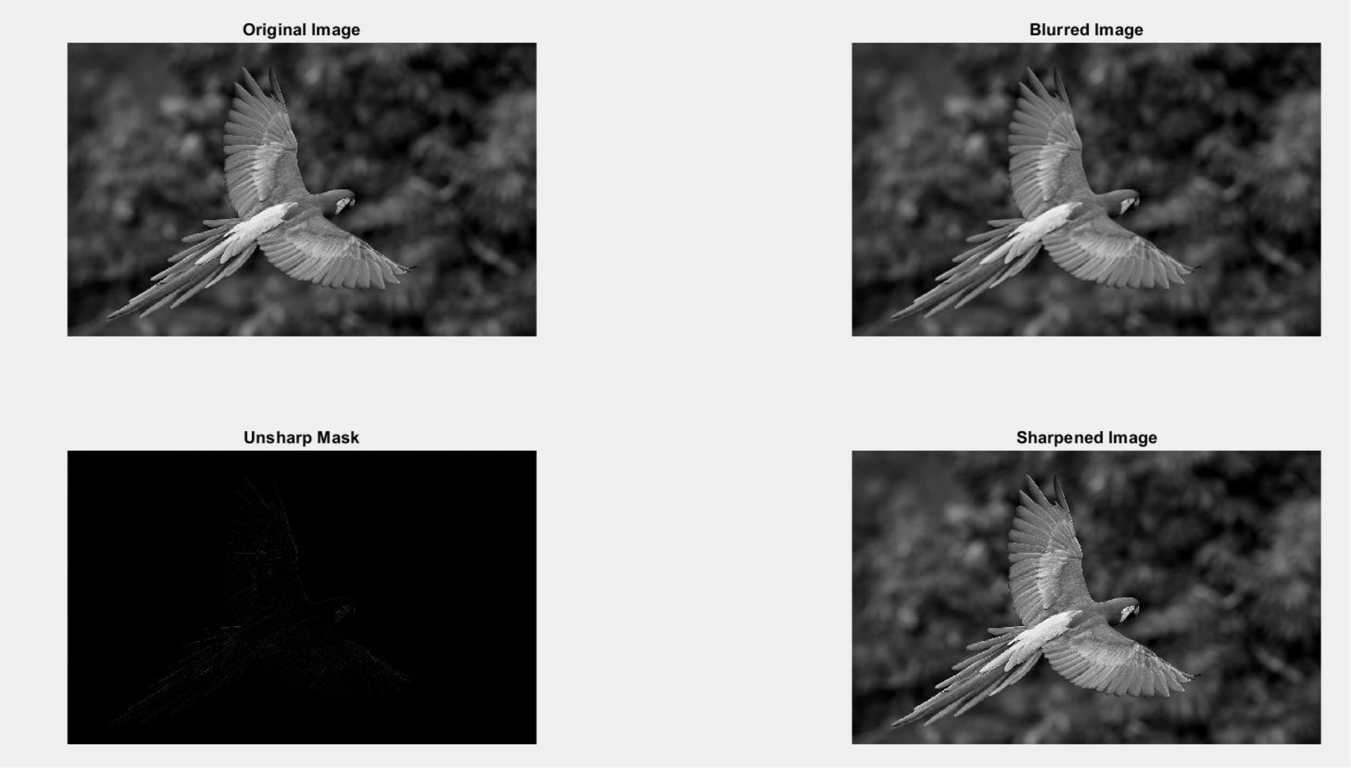


Replica Padded max image

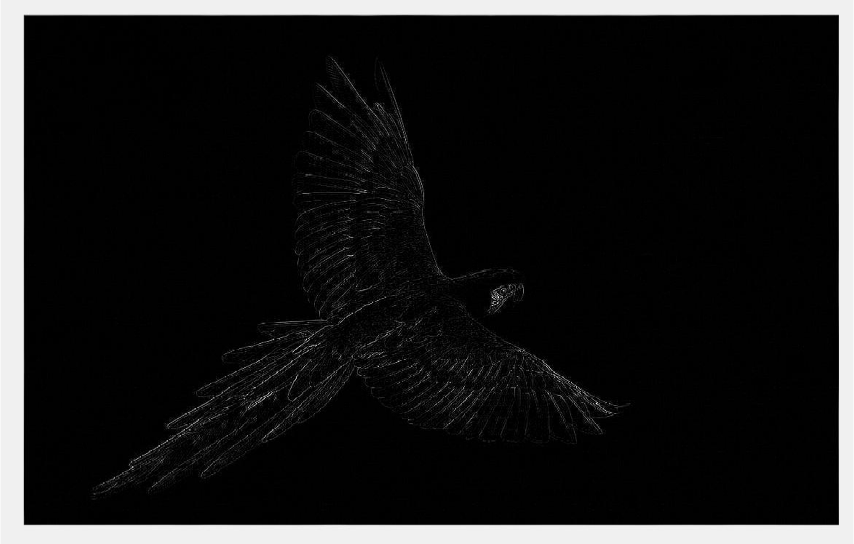
11. Apply Laplacian, unsharp masking, high boost filtering for image sharpening

Answer:

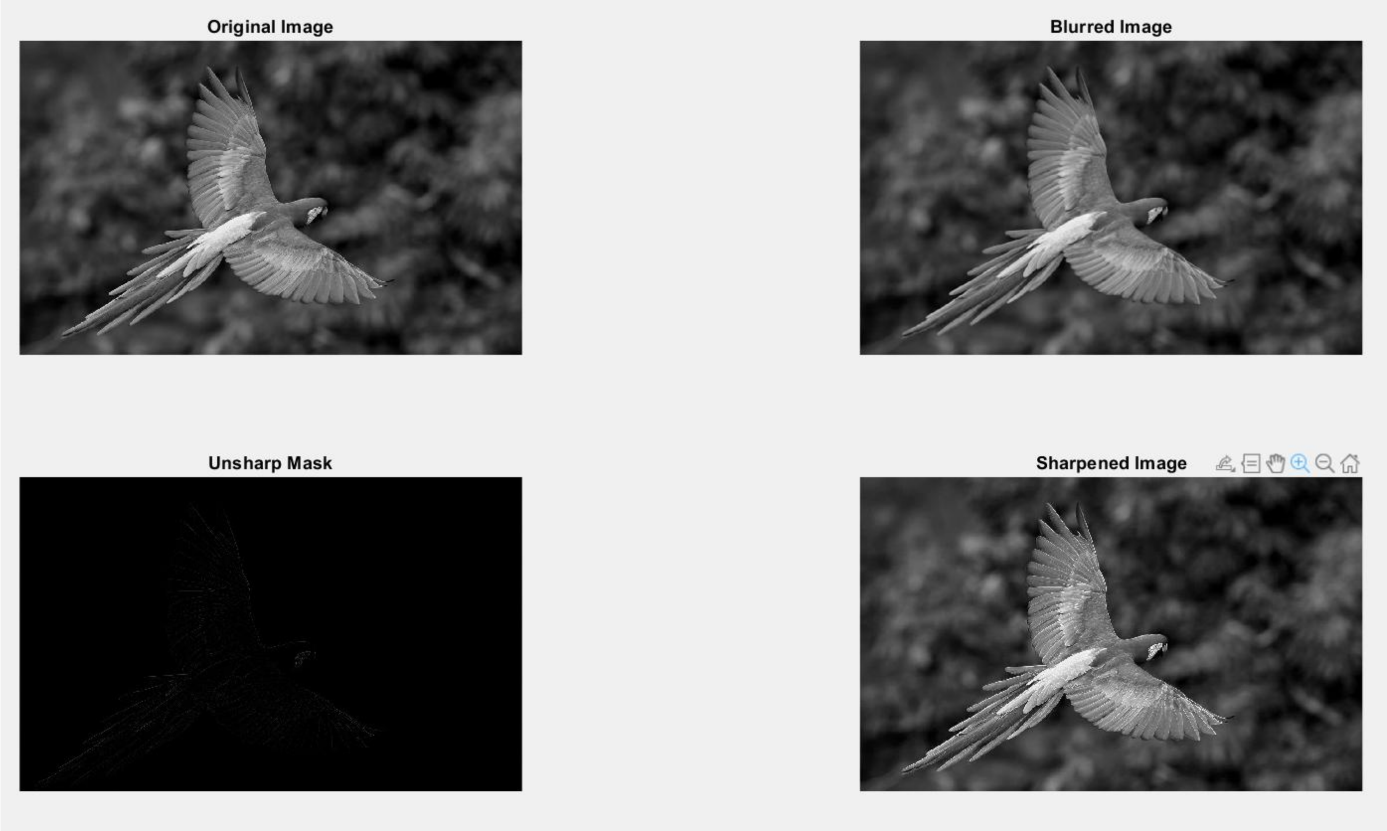
```
image=imread("sample.pgm");
m=size(image);
zero_padded_image=padarray(double(image),[1 1],0);
zero_padded_smooth_image=image;

for i=2:m(1)
    for j=2:m(2)
        sum=zero_padded_image(i-1,j)+zero_padded_image(i+1,j)+zero_padded_image(i,j-
1)+zero_padded_image(i,j+1)+zero_padded_image(i-1,j-
1)+zero_padded_image(i+1,j+1)+zero_padded_image(i+1,j-1)+zero_padded_image(i-
1,j+1)+zero_padded_image(i,j);
        zero_padded_smooth_image(i-1,j-1)=sum/9;
    end
end

mask=image-zero_padded_smooth_image;


sharpenedImage=image+mask;
sharpenedImage=uint8(sharpenedImage);

subplot(2,2,1),imshow(image),title('Original Image');
subplot(2,2,2),imshow(zero_padded_smooth_image),title('Blurred Image');
subplot(2,2,3),imshow(mask),title('Unsharp Mask');
subplot(2,2,4),imshow(sharpenedImage),title('Sharpened Image');
```



```
x=imread("sample.pgm");
m=size(x);

x=double(x);

y=x;

for i=2:m(1)-1
    for j=2:m(2)-1
        y(i,j)=x(i-1,j)+x(i,j-1)+x(i,j+1)+x(i+1,j)-4*x(i,j);
    end
end

y=uint8(y);
imshow(y)
imwrite(y,"Laplacian.pgm");
```



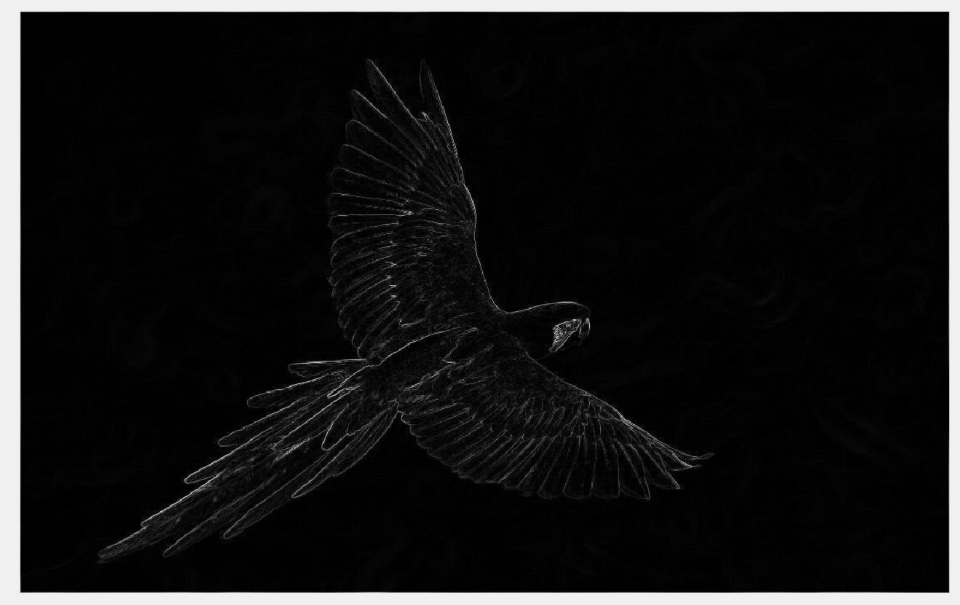
```
image=imread("sample.pgm");
m=size(image);
zero_padded_image=padarray(double(image),[1 1],0);
zero_padded_smooth_image=image;

for i=2:m(1)
    for j=2:m(2)
        sum=zero_padded_image(i-
1,j)+zero_padded_image(i+1,j)+zero_padded_image(i,j-
1)+zero_padded_image(i,j+1)+zero_padded_image(i-1,j-
1)+zero_padded_image(i+1,j+1)+zero_padded_image(i+1,j-1)+zero_padded_image(i-
1,j+1)+zero_padded_image(i,j);
        zero_padded_smooth_image(i-1,j-1)=sum/9;
    end
end

mask=image-zero_padded_smooth_image;


sharpenedImage=image+2*mask;
sharpenedImage=uint8(sharpenedImage);

subplot(2,2,1),imshow(image),title('Original Image');
subplot(2,2,2),imshow(zero_padded_smooth_image),title('Blurred Image');
subplot(2,2,3),imshow(mask),title('Unsharp Mask');
subplot(2,2,4),imshow(sharpenedImage),title('Sharpened Image');
```



12. Apply gradient operators for image sharpening using Roberts cross-gradient operator and Sobel operator.

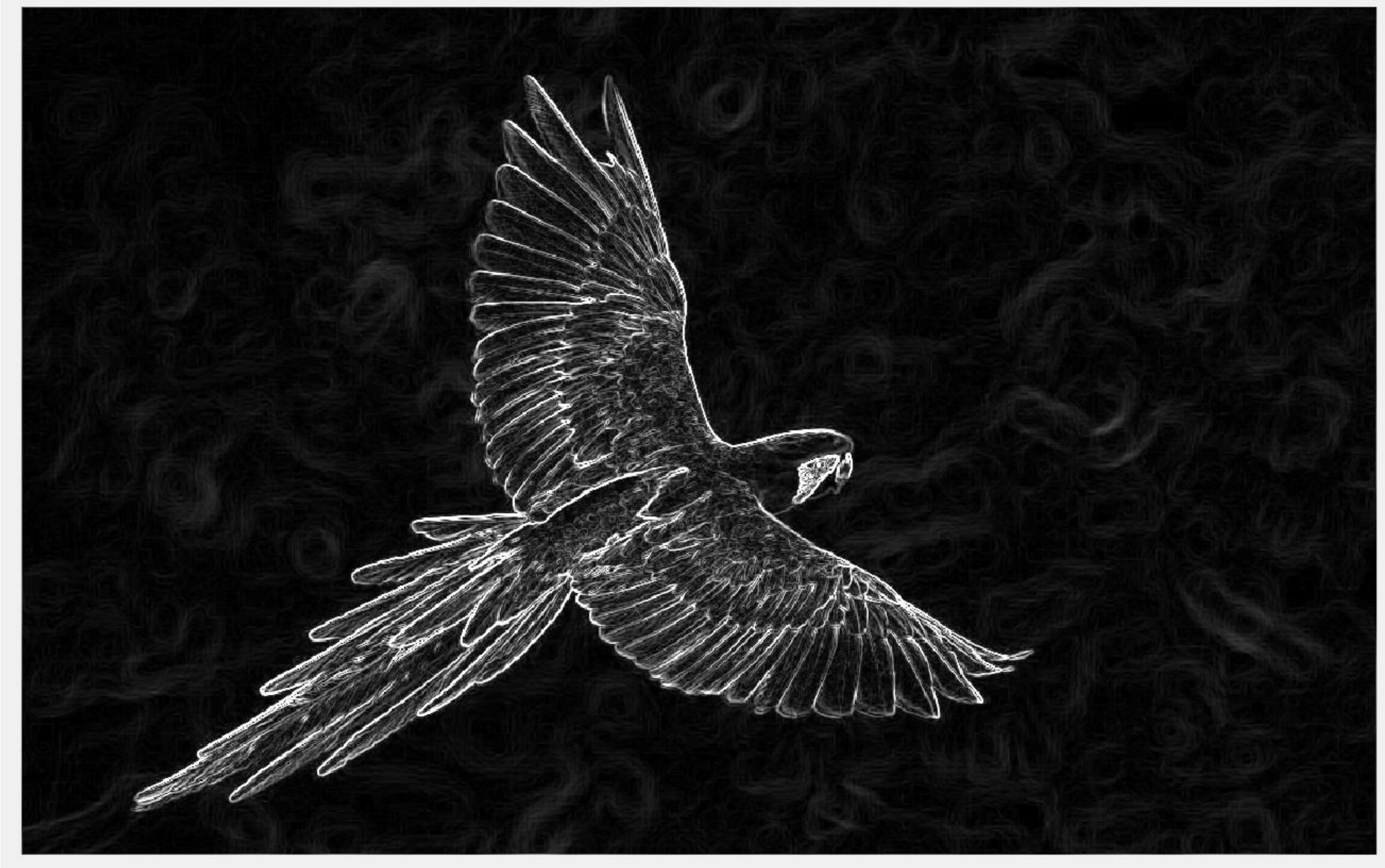
Answer:

```
x=imread("sample.pgm");
m=size(x);

x=double(x);

y=x;

for i=2:m(1)-1
    for j=2:m(2)-1
        robertx=(-1)*x(i-1,j-1)+x(i,j);
        roberty=(-1)*x(i-1,j)+x(i,j-1);
        y(i,j)=((robertx)^2+(roberty)^2)^0.5 ;
    end
end

y=uint8(y);
imshow(y)
imwrite(y,"robert.pgm");
```



Robert

```
x=imread("sample.pgm");
m=size(x);

x=double(x);

y=x;

for i=2:m(1)-1
    for j=2:m(2)-1
        sobelx=((-1)*x(i-1,j-1)+(-2)*x(i-1,j)+(-1)*x(i-1,j+1))+(x(i+1,j-
1)+2*x(i+1,j)+x(i+1,j+1));
        sobely=((-1)*x(i-1,j-1)+(-2)*x(i,j-1)+(-1)*x(i+1,j-1))+(x(i-
1,j+1)+2*x(i,j+1)+x(i+1,j+1));
        y(i,j)=((sobelx)^2+(sobely)^2)^0.5 ;
    end
end

y=uint8(y);
imshow(y)
imwrite(y,"Sobel.pgm");
```



Sobel