University ID : N17385760
Net ID : iv447

HW2

1 Image Smoothing :

1.1 k*k smoothing kernel with equal weights Implement a 2D scheme for 2D square filters of size k*k, and design a filter with equal weights (remember to normalize so that the weights sum up to 1.0). Use a k*k smoothing kernel (3*3, 5*5) to reduce noise and smooth input images. Apply a 3*3 and 5*5 smoothing to your favorite black and white images. Display images before and after filtering. You may also zoom a subregion to better show the filtering effect (as we have done in the slides).

Smoothing can be done to reduce noise in Image. There are different type of filtering that we can apply to image such as Median Filtering, Linear Filtering, Gaussian Filtering. Here we are going to use a linear filter which is an averaging filter.

Here, we are applying k*k smoothing kernel to image. We filter the entire image except for the boundary. Smoothing uses concept of convolution. Convolution is the most important function to implement because it's the basis of all the filtering operations we need to perform.

Code implemented in Matlab for smoothing :

```
function [smooth_img] = smoothing ( I, weight ,display)

r = size(I,1);
c = size(I,2);
smooth_img = zeros(r,c);

for i = ceil(size(weight,1)/2): (size(I,1) - ceil(size(weight,1)/2) + 1)
    for j = ceil(size(weight,2)/2): (size(I,2) - ceil(size(weight,1)/2) + 1)
        convol=0;
        %compute convolution for the neighbourhood associated to the kernel
        for a = 1:size(weight,1)
```

```
            for b=1:size(weight,2)
                convol = convol +
round((weight(a,b)*I(i-a+ceil(size(weight,1)/2),j-b+ceil(size(weight,2)/2))));
            end
        end
        smooth_img(i,j)=convol;
    end
end

if display
    subplot(2,2,1) , imshow(uint8(I));
    title("Original Image");
    subplot(2,2,2) , imshow(uint8(smooth_img));
    title("Smoothed Image");
end
```

The following function takes three parameters :

1. I : Input Image which to be smoothened.
2. weight : K*K kernel applied for filtering.
3. display : if display is passed as true. Subplots of original and smoothened image will be created. Else, only the smoothed image will be returned.

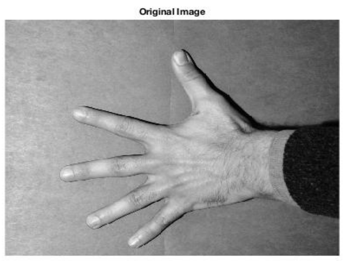
We are iterating over the image and using the averaging filter for convolution.

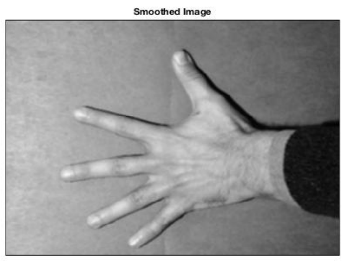
From Matlab command terminal follow these steps to plot the images :

1. For 3*3 kernel :
   ```
   >> I = imread('../hand-bw.jpg');
   >> Gray_image = rgb2gray(I);
   >> w = [1 1 1; 1 1 1; 1 1 1]/9;
   >> smooth_image = smoothing(Gray_image, w,  true);
   ```

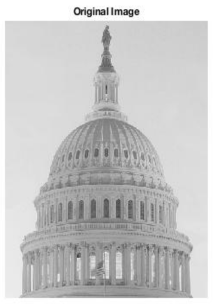
Output :

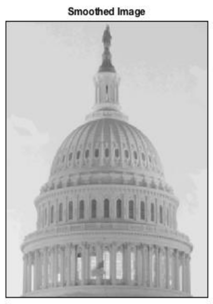
a. For hand-bw.jpg :





b. For capitol.jpg Image :



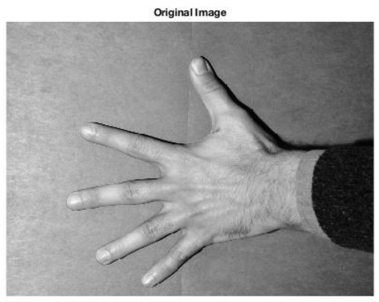


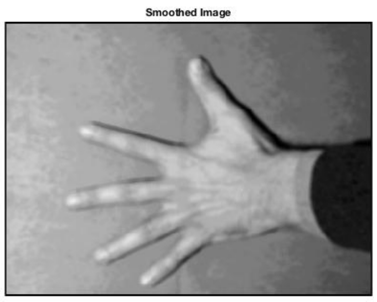
2. For 5*5 Kernel :

```
>> I = imread('../hand-bw.jpg');
>> Gray_image = rgb2gray(I);
>> w = [1 1 1 1 1; 1 1 1 1 1; 1 1 1 1 1; 1 1 1 1 1; 1 1 1 1 1]/25;
>> smooth_image = smoothing(Gray_image, w,  true);
```

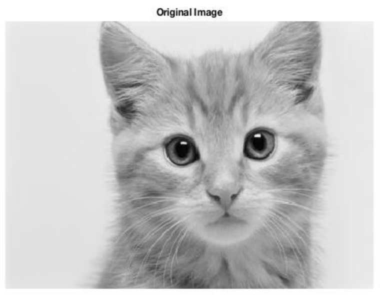
Output :

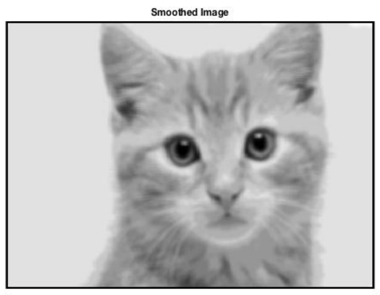
a. For hand-bw.jpg :





b. For test.jpg :





2 Edge Detection

2.1 Local Edge Filtering :

Steps for local Edge Filtering :

1. First Image is smoothened. We can perform the smoothing using our previous smoothing function.
2. We then calculate x derivative of smoothened image. We can use the following kernel for calculation of x derivative : dx = ∂ x I(x, y)

   [-1 0 1] or [-1 0 1]/2

3. After calculation of x derivative, we calculate y derivative of image. We can use following kernel for calculation of y derivative : dy = ∂ y I(x, y)

   [-1 ; 0 ; 1] or [-1 ; 0 ; 1] / 2

4. Calculation of edge map : For calculating edge map we that the square root of summation of squares of x derivative and y derivative. The edge map represents the norm of the gradient so it's a combination of horizontal and vertical edges.

5. Calculation of orientation map : The gradient orientation is calculated as α = tan−1 (dy/dx). It's in degrees in the range [−180, 180]. There is a lot of noise and then that there are areas where the color is the same. So it means that in these areas the gradient is the same. So the angle map gives us the area where the gradient vector has the same orientation.

Implementation of Edge Detection in Matlab :

```
function [] = edgeDetection( InputImage, weight1, weight2)
r = size(InputImage ,1 );
c = size(InputImage ,2);
x_derivative = zeros(r,c);
y_derivative = zeros(r,c);
edge_map = zeros(r,c);
orientation = zeros(r,c);
weight = [1 1 1 ; 1 1 1; 1 1 1]/9;
smoothed_image = smoothing(InputImage,weight,false);
for i = 1 : size(InputImage,1)
   for j = ceil(size(weight1,2)/2) : (size(InputImage,2) - ceil(size(weight1,2)/2) + 1)
      convol=0;
      for a = 1:size(weight1,2)
         w = weight1(a);
         im = smoothed_image(i,j-a+ceil(size(weight1,2)/2));
         x = w * im;
         convol = convol + x;
      end
      x_derivative(i,j)= abs(convol);
   end
end

for i = ceil(size(weight2,1)/2) : (size(InputImage,1) - ceil(size(weight2,1)/2) + 1)
   for j = 1 : size(InputImage,2)
      convol_vertical=0;
      for b = 1:size(weight2,1)
         w = weight2(b);
```

```
            im = smoothed_image(i-b+ceil(size(weight2,1)/2),j);
            y = w * im;
            convol_vertical = convol_vertical + y;
        end
        y_derivative(i,j)=abs(convol_vertical);
    end
end

for i = ceil(size(weight1,2)/2): (size(InputImage,1) - ceil(size(weight1,2)/2) + 1)
    for j = ceil(size(weight1,2)/2): (size(InputImage,2) - ceil(size(weight2,1)/2) + 1)
        edge_map(i,j)=sqrt(x_derivative(i,j)^2+y_derivative(i,j)^2);
        orientation(i,j)=atan(y_derivative(i,j)/x_derivative(i,j))*(360/pi);
    end
end
subplot(2,2,1) ,imshow(uint8(x_derivative),gray);
title("dx");
subplot(2,2,2) , imshow(uint8(y_derivative),gray);
title("dy");
subplot(2,2,3) , imshow(uint8(edge_map),gray);
title("edge Map");
subplot(2,2,4) , imshow(uint8(orientation),gray);
title("Orientation Angle");
```

This function takes three arguments :

1. InputImage : It's the gray Image for edge detection.
2. weight1 : Kernel for calculating x derivative.
3. weight2 : kernel for calculating y derivative.

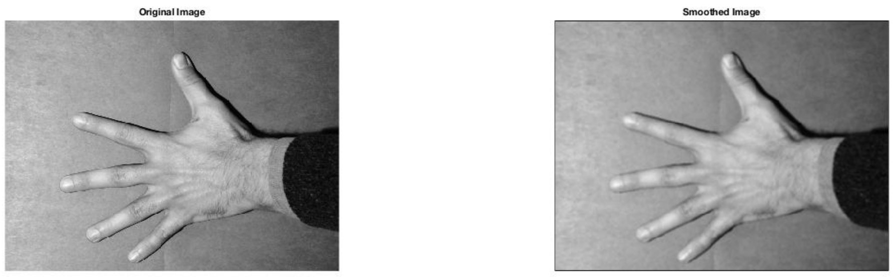
From Matlab command terminal follow these steps to plot the results :

```
>> I = imread('../hand-bw.jpg');
>> i = rgb2gray(I);
>> weight1 = [-1 0 1];
>> weight2 = [-1; 0; 1];
>> edgeDetection(i,weight1,weight2);
```

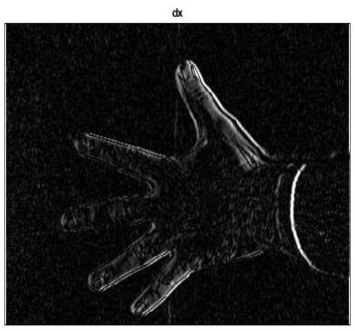
Output :

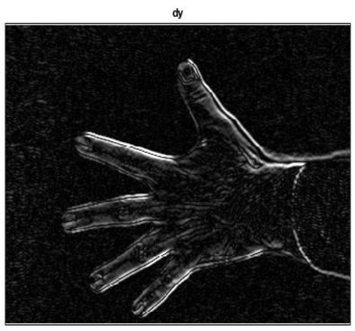
1. For hand-bw.jpg :

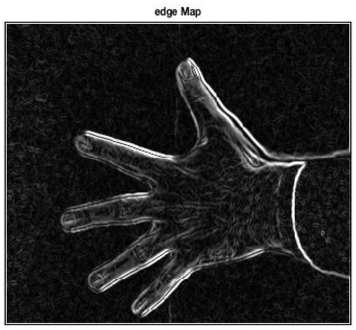
Original and Smoothe Image with 3*3 averaging filter :

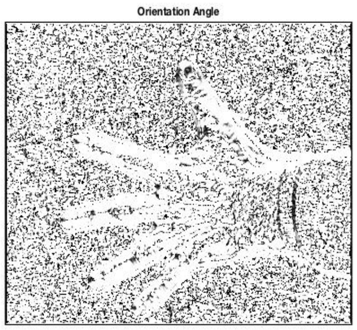


dx, dy, Edge Map and Orientation Map For the hand-bw.jpg Image :









2 . For capitol.jpg :

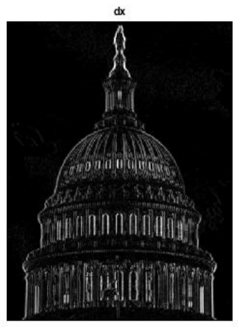
Type following commands from command terminal :

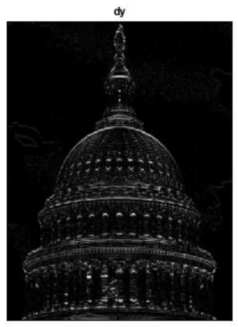
```
>> I = imread('../capitol.jpg');
>> weight1 = [-1 0 1];
>> weight2 = [-1; 0; 1];
>> edgeDetection(I,weight1,weight2);
```

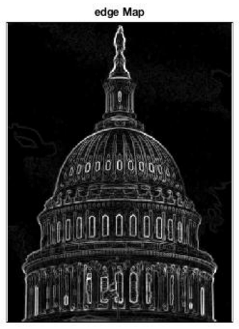
Output:

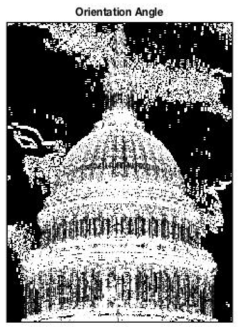
Original and Smoothe Image with 3*3 averaging filter :

dx, dy, Edge Map and Orientation Map For the hand-bw.jpg Image :









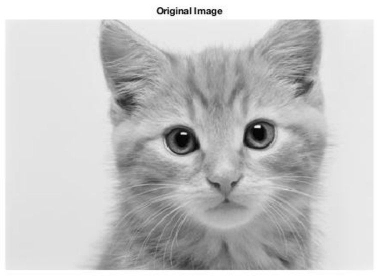
3. For test.jpg Image :

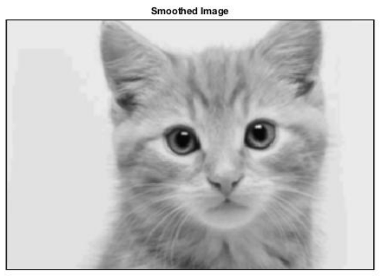
Type following commands from command terminal :

```
>> I = imread('../test.jpg');
>> i = rgb2gray(I);
>> weight1 = [-1 0 1]/2;
>> weight2 = [-1; 0; 1]/2;
>> edgeDetection(i,weight1,weight2);
```

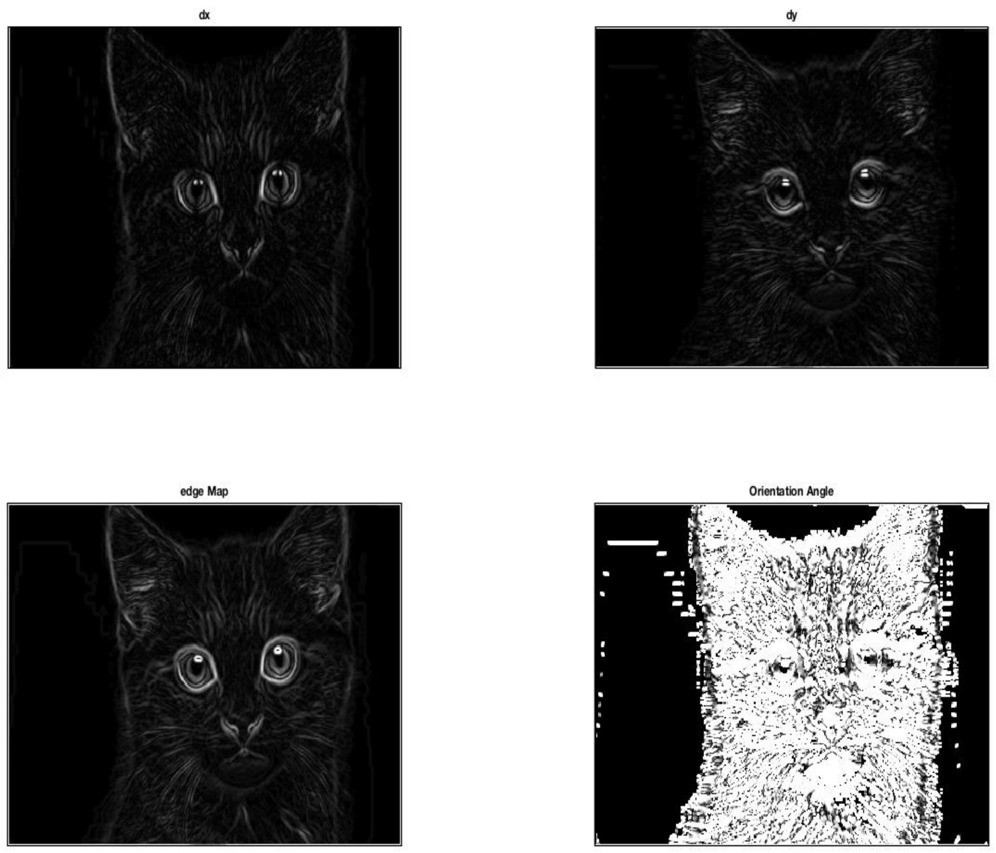
Output:

Original and Smoothe Image with 3*3 averaging filter :





dx, dy, Edge Map and Orientation Map For the hand-bw.jpg Image :

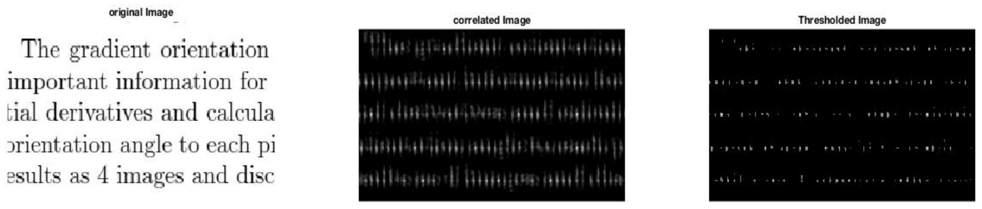


3 Template Matching

In this section, we are going to use the correlation between the image and the mask we are going to apply. First thing to do is to select an image and then select a sub image as template. We are going to use normalized correlation to find the correlated image and then apply thresholding to get the peaks. We can further apply laplacian filter [0 -1 0 ; -1 4 -1 ; 0 -1 0] to get the peaks, if the thresholded result is not satisfactory.

1. Text Correlation :
   Here i have taken a image text.jpg and trying to locate alphabet a in it.

Results :


original Image
correlated Image
Thresholded Image
The gradient orientation
important information for
tial derivatives and calcula
orientation angle to each pi
esults as 4 images and disc

So as we can see in the results, that it is harder to get exactly the position of ”a” in the text correlated image, because objects are smaller, closer to each other and not separable easily. Thresholding improves the result.

Template Matching for bigger objects :

Implementation of Template Matching in Matlab :

1. Template Matching Code :

```
function [] = TemplateMatching(InputImage, template)

%calculate norm correlation
correlate_Image = correlation ( InputImage, template);

%convert double image to gray levels
```

```
maxv1 = max(correlate_Image(:));
corr_image = uint8((double(correlate_Image) ./ maxv1) .* 255);

%Thresholded Image
corr_threshold = 0.02;

threshold_peakdetection = threshold_image ( correlate_Image,corr_threshold);

%Laplacian Peak Detection
kernel = [1 1 1; 1 1 1; 1 1 1]/9;
smooth_image = smoothing(corr_image,kernel,false);
%Threshold on laplacian
threshold = 10;
peak_detected = laplacian(smooth_image,threshold);

%convert double image to gray levels
maxv2 = max(peak_detected(:));
peak_image = uint8((double(peak_detected) ./ maxv2) .* 255);

subplot(2,2,1) ,imshow(InputImage);
title("original Image");
subplot(2,2,2) , imshow(corr_image);
title("correlated Image");
subplot(2,2,3) , imshow(threshold_peakdetection);
title("Thresholded Image");
subplot(2,2,4) , imshow(peak_image);
title("Peak Detected Image");
```

2. Correlation code :

```
function [correlated_image] = correlation ( InputImage, template)
[r1,c1]=size(InputImage);
[r2,c2]=size(template);
image22 = template-mean(mean(template));
correlated_image = [];

for i=1:(r1-r2+1)
   for j=1:(c1-c2+1)
      Nimage=InputImage(i:i+r2-1,j:j+c2-1);
```

```
        Nimage=Nimage-mean(mean(Nimage));  % mean of image part under mask
        corr = sum(sum(Nimage.* image22));
        norm = sqrt(sum(sum(Nimage.^2)))*(sum(sum(image22.^2)));
        correlated_image(i,j)=corr/norm;
    end
end
```

3. Threshold Code :

```
function [threshold_image] = threshold_image ( InputImage,threshold)
OutputImage = [];

for i=1:size( InputImage,1)
    for j=1:size( InputImage,2)
        value= InputImage(i,j);
        if value <= threshold
            OutputImage(i,j)=0;
        else
            OutputImage(i,j)=value;
        end
    end
end

%thresholded image
max_thresh = max(OutputImage(:));
threshold_image = uint8((double(OutputImage) ./ max_thresh) .* 255);
```

4. Laplacian Code :

```
function [OutputImage] = laplacian( InputImage,threshold)


laplacian_kernel = [0 -1 0; -1 4 -1; 0 -1 0];
OutputImage = [];
%can use smoothing function for obtaining peaks
laplacian_image = smoothing(InputImage,laplacian_kernel,false);

for i=1:size( laplacian_image,1)
    for j=1:size( laplacian_image,2)
```

```
        value= laplacian_image(i,j);
        if value <= threshold
           OutputImage(i,j)=value;
        else
           OutputImage(i,j)=0;
        end
    end
end
```

TemplateMatching function takes two inputs :

1. InputImage : Input image in which we have to find the matching Template
2. template : Template (sub image within image) to be found.

TemplateMatching first calls correlation function on image and template to find the normalized correlation. correlation takes two arguments InputImage and template.

In Correlation Function we perform following steps :

1. We Calculate the mean intensity of template region and deduct the mean from each template pixel. The result is a zero-mean template within the template with positive and negative pixel values.

2. Use the template as a mask m*n, and filter the image with this mask to obtain a correlation image. For each move of the template to filter the image, we calculate the mean intensity within the respective region of the image and then subtract this mean from each pixel value while calculating the correlation. After obtaining the correlated image we apply threshold on the correlated image. Exact value of threshold can be found by plotting the histogram. Here, the threshold for the image is 0.02. We call the threshold_image function from TemplateMatching function to get the thresholded Image.

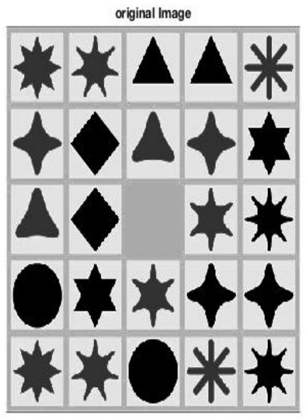
For further better peak detection we can appply laplacian filter [0 -1 0 ; -1 4 -1 ; 0 -1 0] to the correlated image. First perform the smoothing of the correlated image using averaging linear filter of [1 1 1; 1 1 1 ; 1 1 1]/9 mask. Then to the smoothened image we can apply laplacian filter. Laplacian is applied for a peak detection to the correlation image to enhance peaks of best correlating regions.

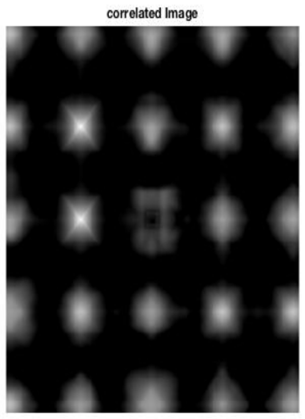
We can apply the above Template Matching functionality to shape-bw.jpg image. Here, we are trying to locate diamond image within shape-bw image.

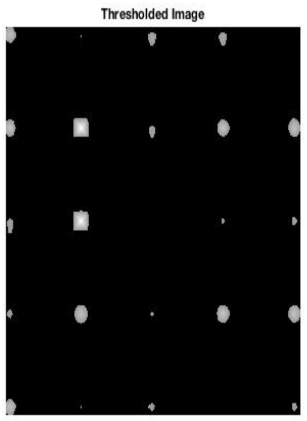
From Matlab Command Terminal follow these steps to execute the above Template Matching code :

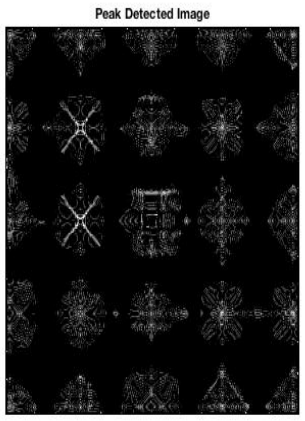
```
>> I = imread('../shapes-bw.jpg');
>> a = rgb2gray(I);
>> i = im2double(a);
>> T = imread('../temp.jpg');
>> b = rgb2gray(T);
>> t = im2double(b);
>> TemplateMatching(i,t);
```

Output :


original Image


correlated Image


Thresholded Image


Peak Detected Image