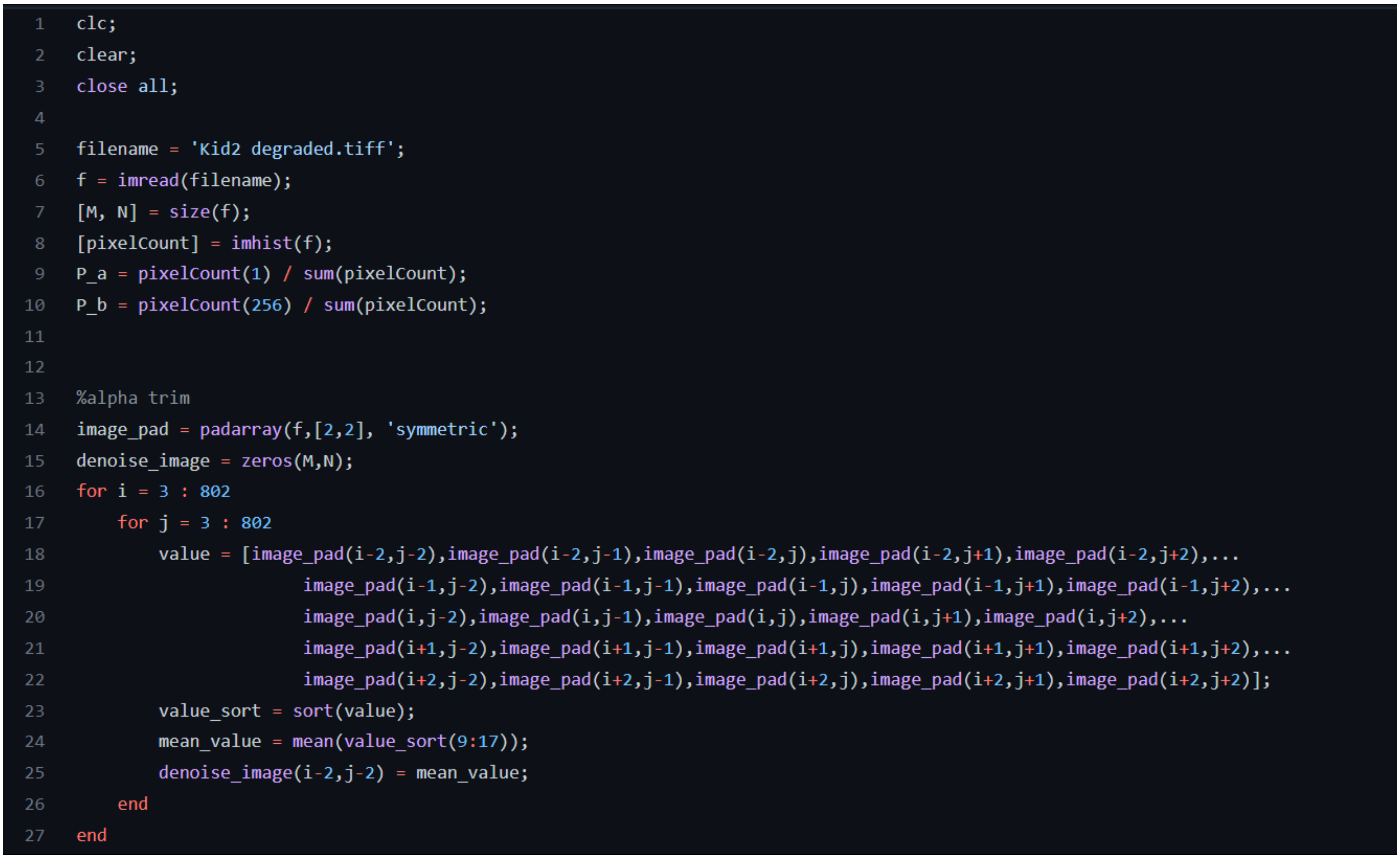
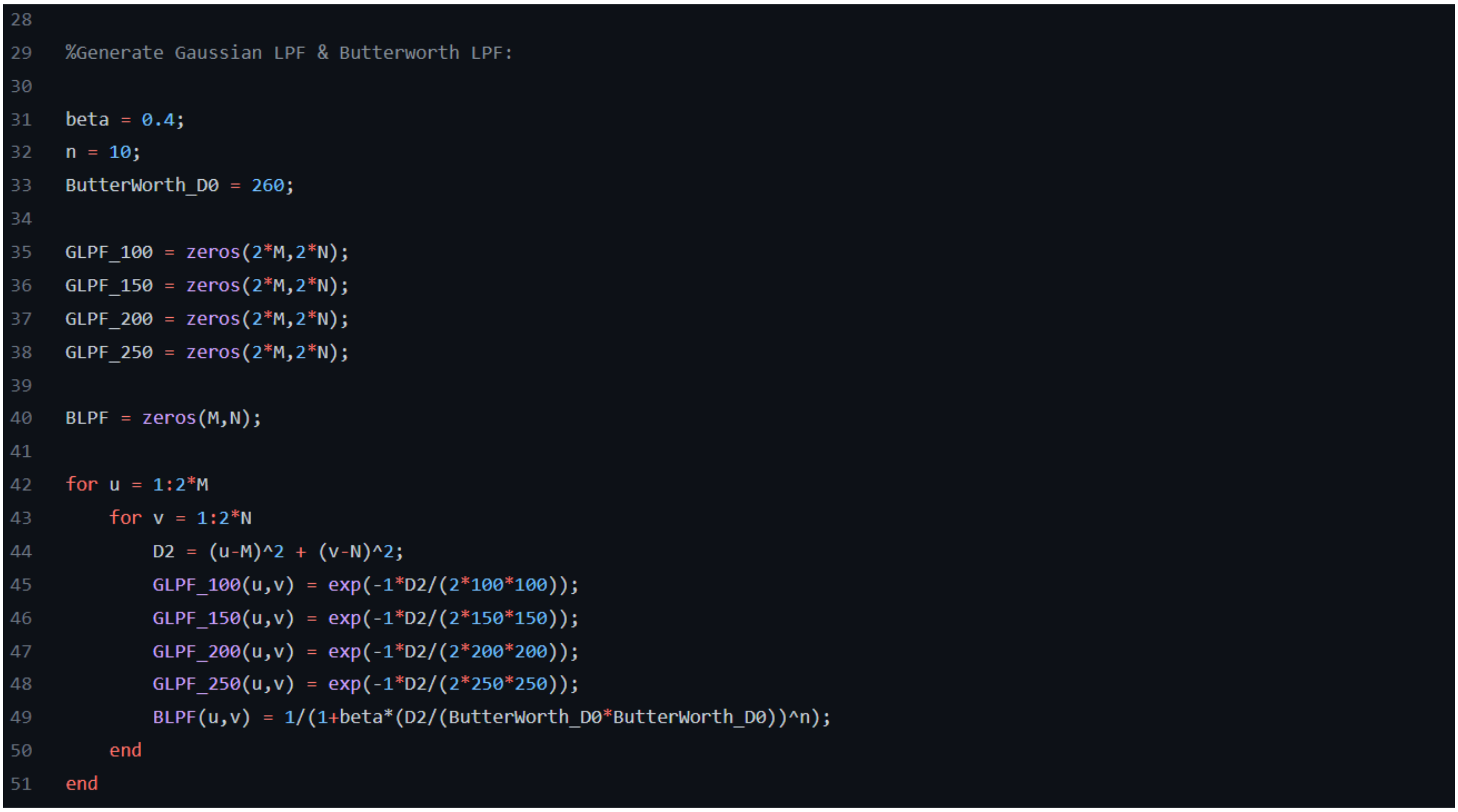
Project 2

ID：0811562   name：何祁恩

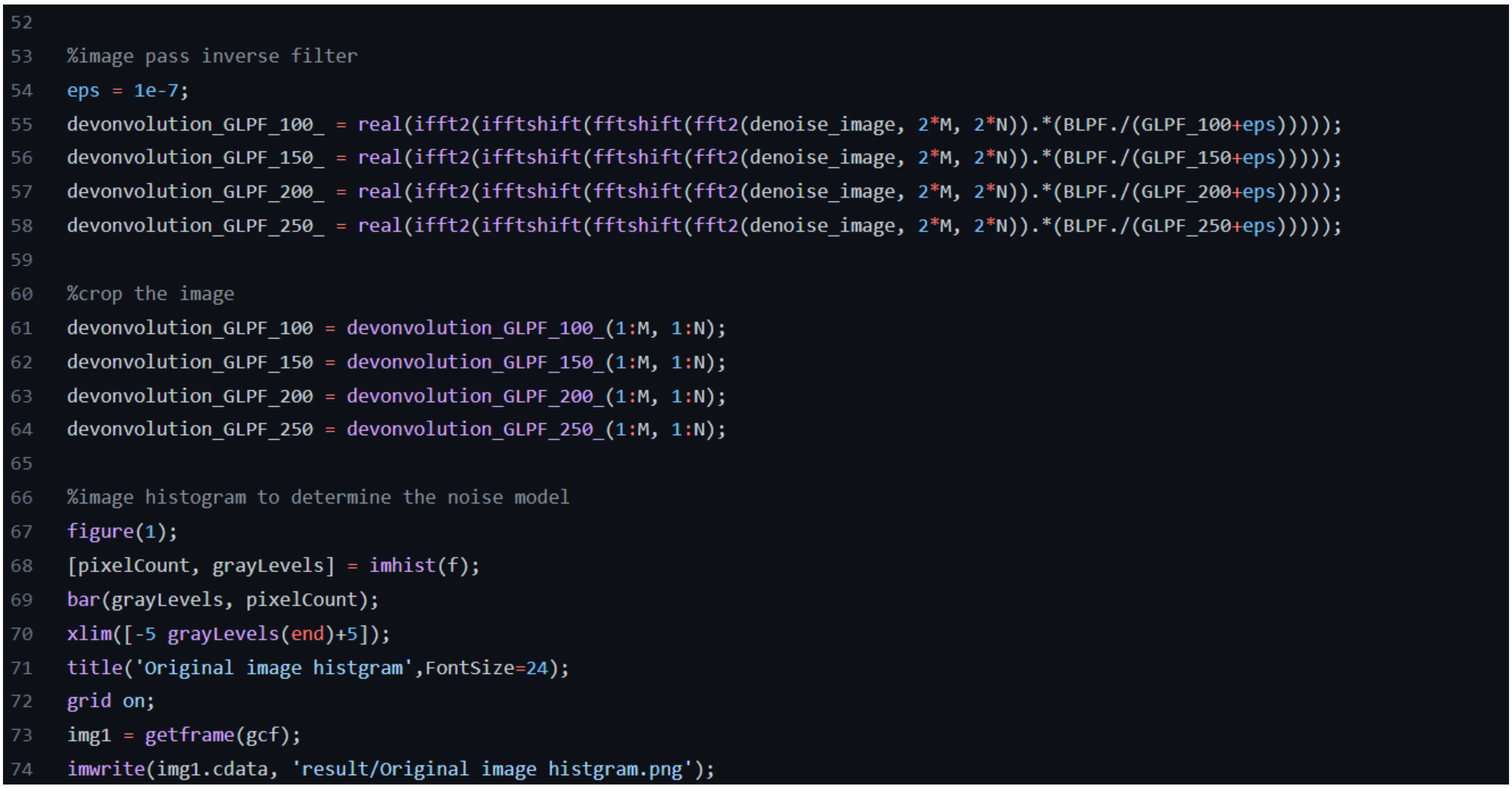
(a) Source codes:



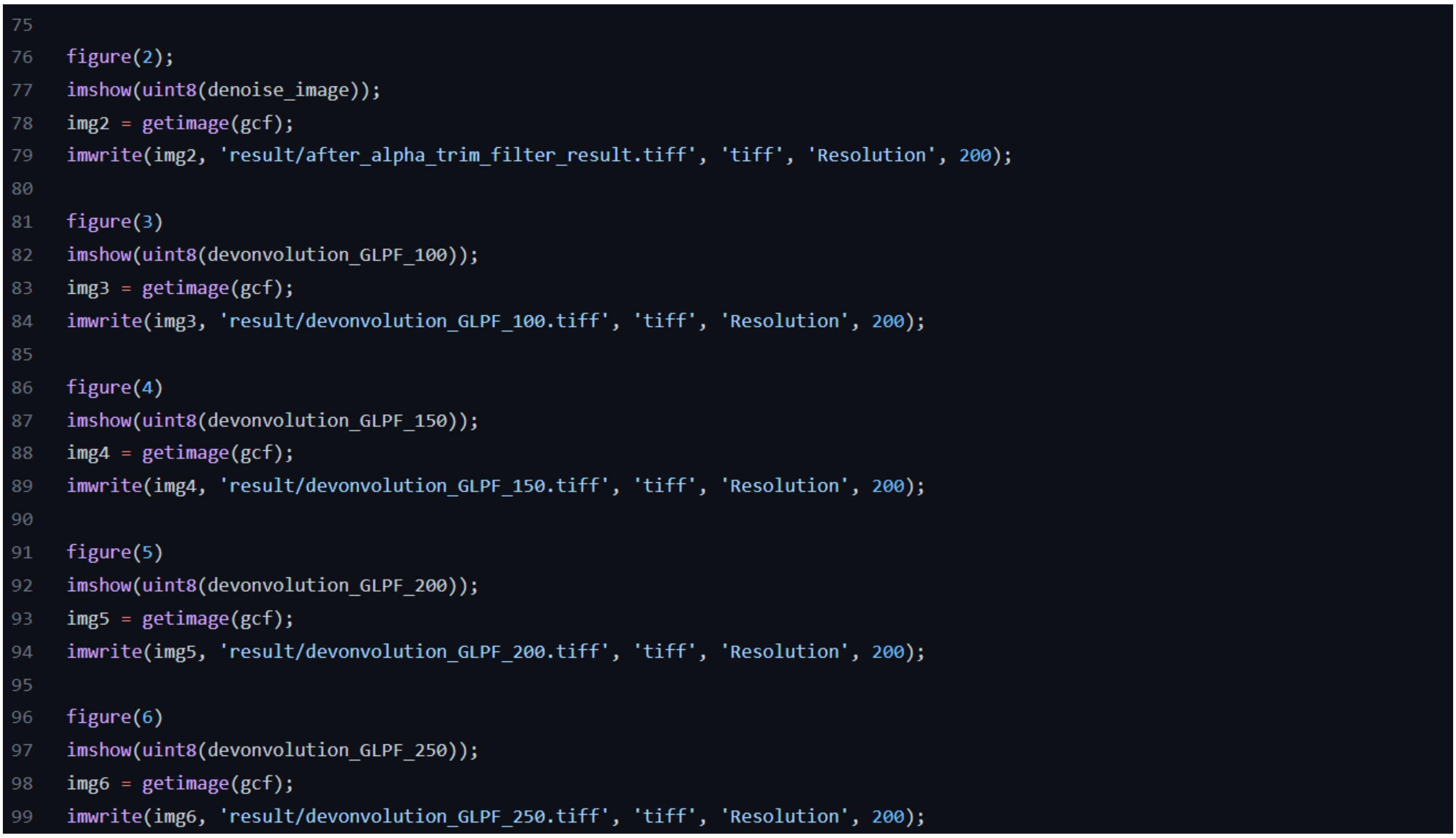
```
clc;
clear;
close all;

filename = 'Kid2 degraded.tiff';
f = imread(filename);
[M, N] = size(f);
[pixelCount] = imhist(f);
P_a = pixelCount(1) / sum(pixelCount);
P_b = pixelCount(256) / sum(pixelCount);


%alpha trim
image_pad = padarray(f,[2,2], 'symmetric');
denoise_image = zeros(M,N);
for i = 3 : 802
    for j = 3 : 802
        value = [image_pad(i-2,j-2),image_pad(i-2,j-1),image_pad(i-2,j),image_pad(i-2,j+1),image_pad(i-2,j+2),...
                      image_pad(i-1,j-2),image_pad(i-1,j-1),image_pad(i-1,j),image_pad(i-1,j+1),image_pad(i-1,j+2),...
                      image_pad(i,j-2),image_pad(i,j-1),image_pad(i,j),image_pad(i,j+1),image_pad(i,j+2),...
                      image_pad(i+1,j-2),image_pad(i+1,j-1),image_pad(i+1,j),image_pad(i+1,j+1),image_pad(i+1,j+2),...
                      image_pad(i+2,j-2),image_pad(i+2,j-1),image_pad(i+2,j),image_pad(i+2,j+1),image_pad(i+2,j+2)];
        value_sort = sort(value);
        mean_value = mean(value_sort(9:17));
        denoise_image(i-2,j-2) = mean_value;
    end
end
```



```

%Generate Gaussian LPF & Butterworth LPF:

beta = 0.4;
n = 10;
ButterWorth_D0 = 260;

GLPF_100 = zeros(2*M,2*N);
GLPF_150 = zeros(2*M,2*N);
GLPF_200 = zeros(2*M,2*N);
GLPF_250 = zeros(2*M,2*N);

BLPF = zeros(M,N);

for u = 1:2*M
    for v = 1:2*N
        D2 = (u-M)^2 + (v-N)^2;
        GLPF_100(u,v) = exp(-1*D2/(2*100*100));
        GLPF_150(u,v) = exp(-1*D2/(2*150*150));
        GLPF_200(u,v) = exp(-1*D2/(2*200*200));
        GLPF_250(u,v) = exp(-1*D2/(2*250*250));
        BLPF(u,v) = 1/(1+beta*(D2/(ButterWorth_D0*ButterWorth_D0))^n);
    end
end
```

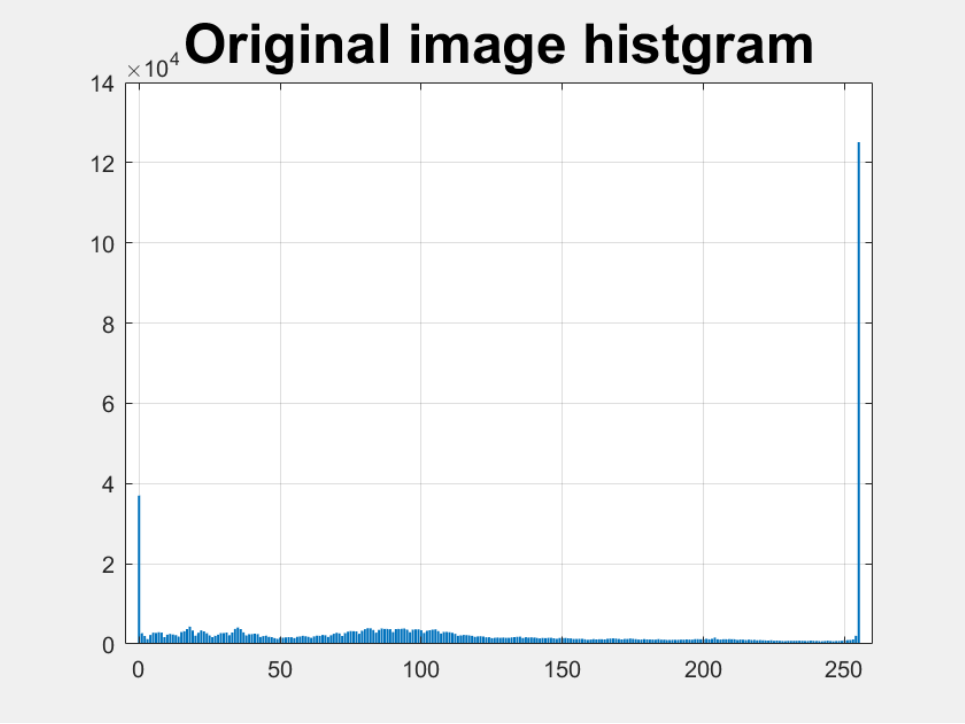


```

%image pass inverse filter
eps = 1e-7;
devonvolution_GLPF_100_ = real(ifft2(ifftshift(fftshift(fft2(denoise_image, 2*M, 2*N)).*(BLPF./(GLPF_100+eps)))));
devonvolution_GLPF_150_ = real(ifft2(ifftshift(fftshift(fft2(denoise_image, 2*M, 2*N)).*(BLPF./(GLPF_150+eps)))));
devonvolution_GLPF_200_ = real(ifft2(ifftshift(fftshift(fft2(denoise_image, 2*M, 2*N)).*(BLPF./(GLPF_200+eps)))));
devonvolution_GLPF_250_ = real(ifft2(ifftshift(fftshift(fft2(denoise_image, 2*M, 2*N)).*(BLPF./(GLPF_250+eps)))));

%crop the image
devonvolution_GLPF_100 = devonvolution_GLPF_100_(1:M, 1:N);
devonvolution_GLPF_150 = devonvolution_GLPF_150_(1:M, 1:N);
devonvolution_GLPF_200 = devonvolution_GLPF_200_(1:M, 1:N);
devonvolution_GLPF_250 = devonvolution_GLPF_250_(1:M, 1:N);

%image histogram to determine the noise model
figure(1);
[pixelCount, grayLevels] = imhist(f);
bar(grayLevels, pixelCount);
xlim([-5 grayLevels(end)+5]);
title('Original image histgram',FontSize=24);
grid on;
img1 = getframe(gcf);
imwrite(img1.cdata, 'result/Original image histgram.png');
```



```

figure(2);
imshow(uint8(denoise_image));
img2 = getimage(gcf);
imwrite(img2, 'result/after_alpha_trim_filter_result.tiff', 'tiff', 'Resolution', 200);

figure(3)
imshow(uint8(devonvolution_GLPF_100));
img3 = getimage(gcf);
imwrite(img3, 'result/devonvolution_GLPF_100.tiff', 'tiff', 'Resolution', 200);

figure(4)
imshow(uint8(devonvolution_GLPF_150));
img4 = getimage(gcf);
imwrite(img4, 'result/devonvolution_GLPF_150.tiff', 'tiff', 'Resolution', 200);

figure(5)
imshow(uint8(devonvolution_GLPF_200));
img5 = getimage(gcf);
imwrite(img5, 'result/devonvolution_GLPF_200.tiff', 'tiff', 'Resolution', 200);

figure(6)
imshow(uint8(devonvolution_GLPF_250));
img6 = getimage(gcf);
imwrite(img6, 'result/devonvolution_GLPF_250.tiff', 'tiff', 'Resolution', 200);
```

(b) Results of noise model and model parameters:

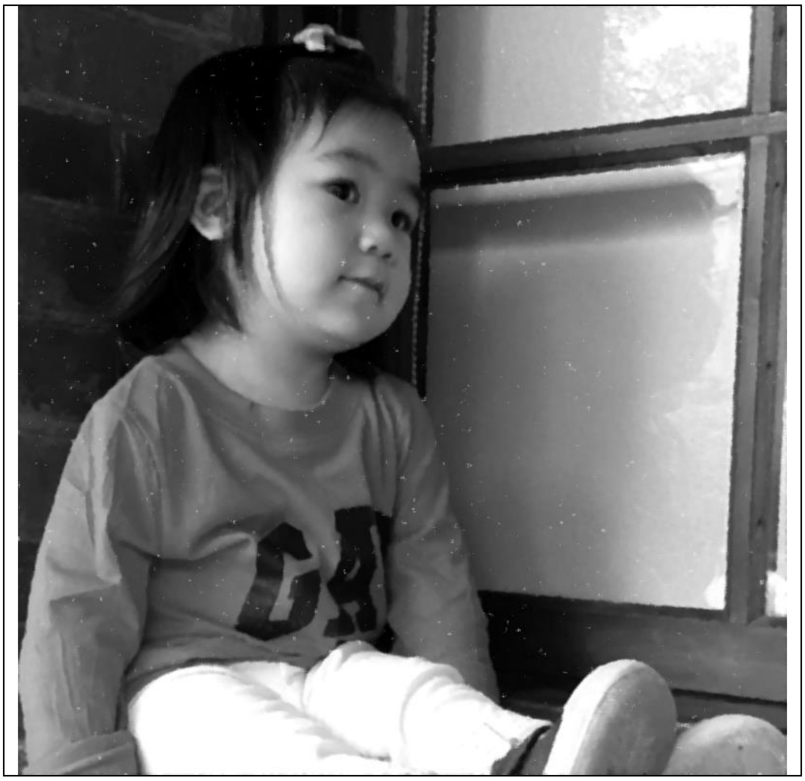


From the original image histogram, we can see that the pixel values of 0 and 255 have the highest frequency, therefore, the noise model is pepper and salt.

The pepper and salt noise model parameters are Pa(pepper probability) and Pb(salt probability).
To obtain Pa and Pb, I use the number of black or white pixels divided by all pixels. However, there must be some pixels that are originally black and white, therefore, the probability is roughly estimated.

P_a	0.0578
P_b	0.1955

(c) De-noised image by alpha-trimmed mean filter:



We can see that almost all the pepper and salt noise was removed.

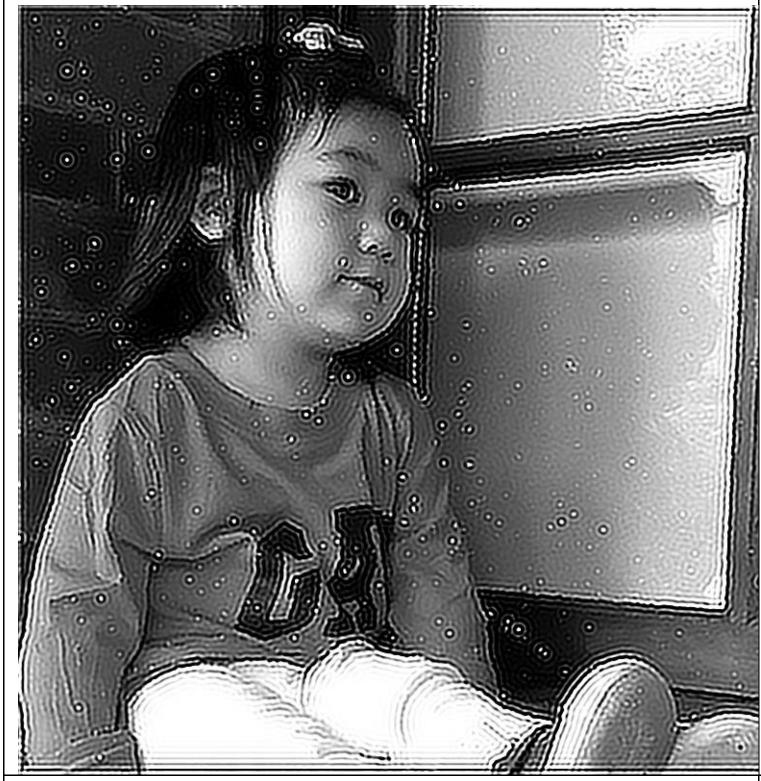
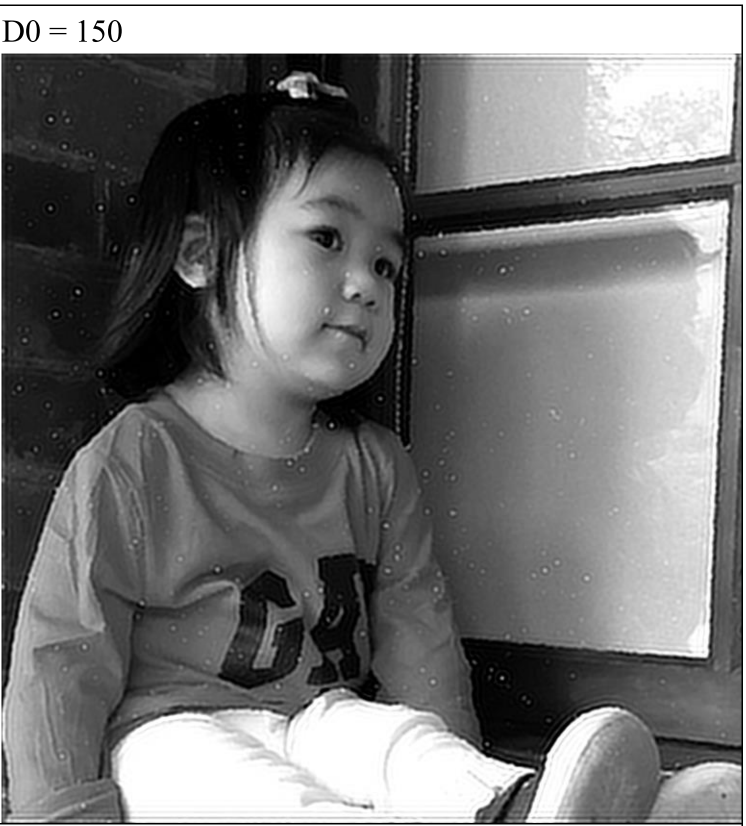
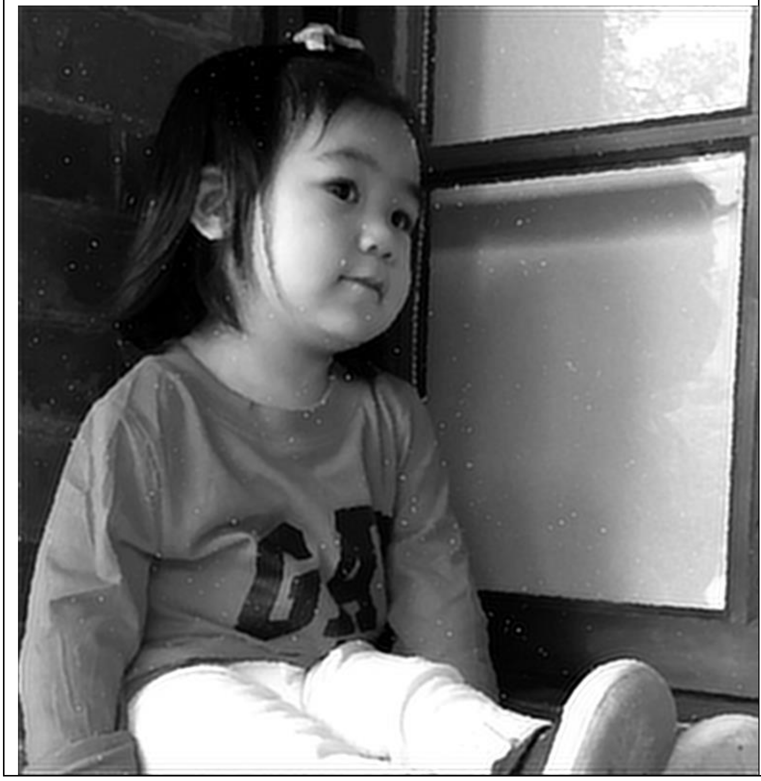
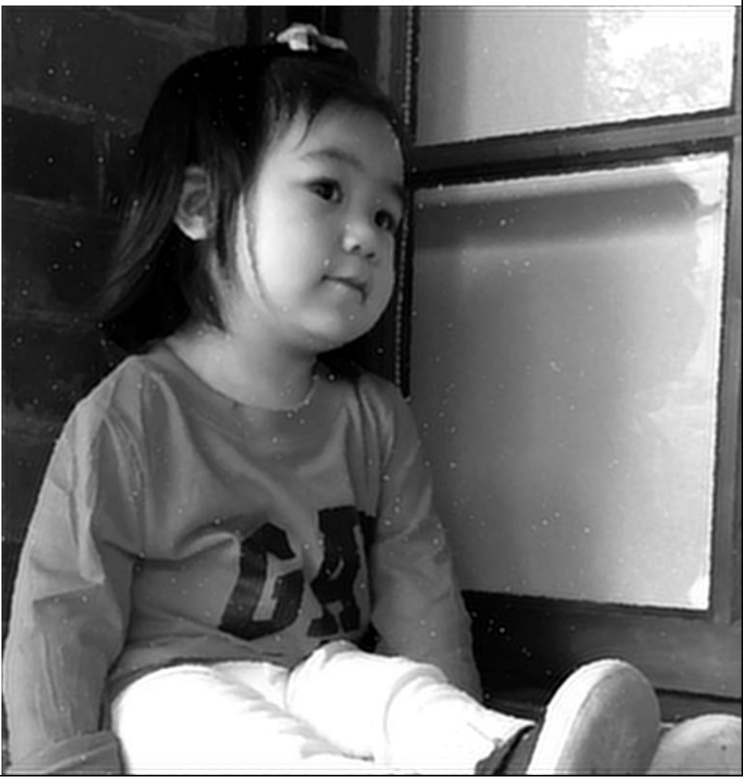
Since we use the 5x5 mask for the alpha-trimmed filter, we may encounter boundary issues on the image boundary. Therefore, I pad two rows on the top and bottom of the original image and pad two cols on the left and right of the original image. The padding method is mirror padding instead of zero padding so that the output image will not become darker.

The implementation detail of the alpha-trimmed filter:

Pad ➜ Find 25 neighbors ➜ Sort ➜ Get rid of 16 pixels ➜ Calculate the mean value of the rest pixels

(d) Output image、parameters:

Butterworth parameter: n = 10, beta = 0.4, cut frequency for Butterworth: 260.

D0 = 100 	D0 = 150 
D0 = 200 	D0 = 250 

In my opinion, I think the best result might be D0 = 250.