图像增强综述

作者: 方阳

这篇文章是DIP的第二次作业，对图像增强技术进行综述，目录如下😁：

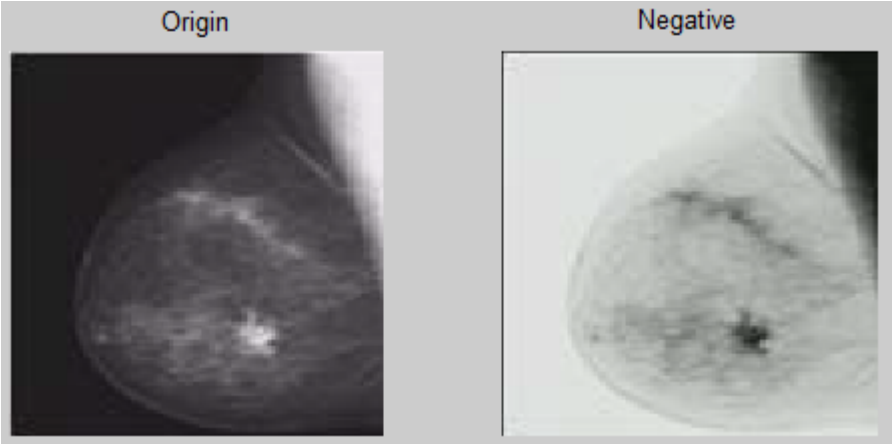
1. Point Operations

1.1 Image Negative

图像反转（Image Negative）在许多应用中都很有用，例如显示医学图像和用单色正片拍摄屏幕，其想法是将产生的负片用作投影片。

转换方程：$T: G(x,y) = L - F(x,y)$，其中L是最大强度值，灰度图像L为255。

实验结果：



source code:

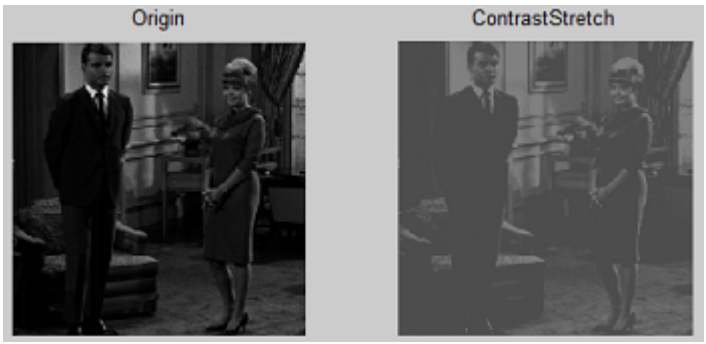
```
%% 图像反转
L = 255;
img_orig = imread('mammogram.png');
figure();
subplot(1, 2, 1);
imshow(img_orig);
title('Origin');
img_negative = L - img_orig;
subplot(1, 2, 2);
imshow(img_negative);
title('Negative');
```

1.2 Contrast Stretching

低对比度的图像可能是由于光照不足，图像传感器缺乏动态范围，甚至在图像采集过程中透镜孔径设置错误。对比度拉伸（Contrast Stretching）背后的想法是增加图像处理中灰度的动态范围。

转换公式：$G(x,y) = g_1 + (g_2 - g_1)/(f_2 - f_1)[F(x,y) - 1]$，其中这里$[f1, f2]$为灰度在新范围$[g1, g2]$上的映射，这里f1为图像的最小强度值，f2为图像的最大强度值。该函数增强了图像的对比度，显示了均匀的强度分布。

实验结果：



source code:

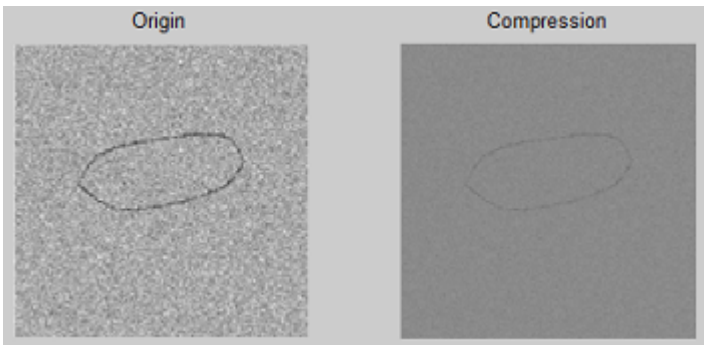
```
%% 对比度拉伸
imgContrastStretch=imadjust(imgOrig, [0.1 0.4], [0.3 0.5 ])
figure();
subplot(1, 2, 1);
imshow(imgOrig);
title('Origin');
subplot(1, 2, 2);
imshow(imgContrastStretch);
title('ContrastStretch');
```

1.3 Compression of dynamic range

有时，处理后的图像的动态范围远远超过显示设备的能力，在这种情况下，只有图像最亮的部分在显示屏上可见. 则需要对图像进行动态范围压缩（Compression of dynamic range）。

转换公式：$s = clog(1 + |r|)$， c是度量常数，是对数函数执行所需的压缩。r表示当前像素的灰度，s为转换后该像素的灰度。例如将以下图像的$[0.255]$压缩到$[0, 150]$，其中$c = \frac{150}{log(1+255)}$.

实验结果：



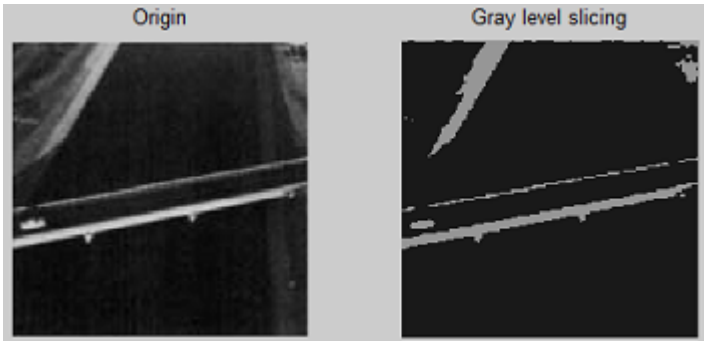
source code:

```
%% 动态范围压缩
img_orig1 = rgb2gray(imread('circle.png'));
c = 150 / log(1 + 255);
img_size = size(img_orig1);
for i = 1:img_size(1)
    for j = 1:img_size(2)
        s(i, j) = c * log(double(1 + img_orig1(i, j)));
    end
end
figure();
subplot(1, 2, 1);
imshow(img_orig1);
title('Origin');
subplot(1, 2, 2);
imshow(uint8(s));
title('Compression');
```

1.4 Grey level slicing

灰度切片(Grey level slicing)相当于带通滤波的空间域。灰度切片函数既可以强调一组灰度值而减少其他所有灰度值，也可以强调一组灰度值而不考虑其他灰度值。例如对图像中灰度值为[100, 180]的区域进行强调，对其他区域进行抑制。

实验结果：

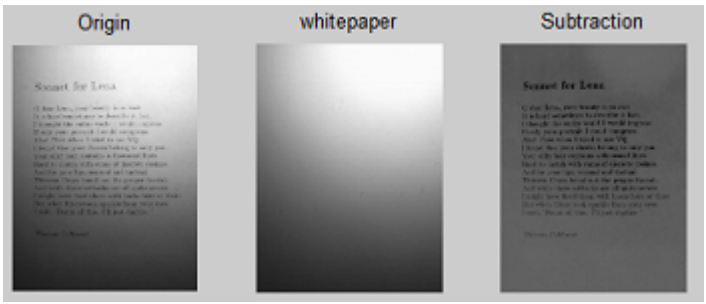


source code:

```
%% 灰度切片
img_orig2 = rgb2gray(imread('GLS.png'));
figure();
subplot(1, 2, 1);
imshow(img_orig2);
title('Origin');
img_size = size(img_orig2);
for i = 1:img_size(1)
    for j = 1:img_size(2)
        if img_orig2(i, j) > 100 || img_orig2(i, j) > 180
            img_orig2(i, j) = 150;
        else
            img_orig2(i, j) = 25;
        end
    end
end
subplot(1, 2, 2);
imshow(img_orig2);
title('Gray level slicing');
```

1.5 Image Subtraction

图像相减是从另一个图像中减去一个像素或整个图像的数字数值的过程。这主要是出于两个原因——调平图像的不均匀部分和检测两幅图像之间的变化。一个常用的方法是从场景中减去背景光照的变化，以便更容易地分析其中的前景对象.例如在捕获过程中光照很差的文本，以便在整个图像中有很强的光照梯度，如果我们希望将前景文本从背景页面中分离出来，也许我们不能调整光照，但是我们可以在场景中放入不同的东西。例如，显微镜成像通常就是这样。所以我们用一张白纸替换文本，在不改变任何东西的情况下，我们捕捉到一个新的图像。我们可以从原始图像中减去光场图像，试图消除背景强度的变化。 实验结果如下：



source code:

```
%% 图像相减
img_orig3 = rgb2gray(imread('son1.png'));
img_add = uint16(img_orig3) + 100;
whitepaper = rgb2gray(imread('son2.png'));
img_sub = uint8(img_add - uint16(whitepaper));
```

```
figure();
subplot(1, 3, 1);
imshow(img_orig3);
title('Origin');
subplot(1, 3, 2);
imshow(whitepaper);
title('whitepaper');
subplot(1, 3, 3);
imshow(img_sub);
title('Subtraction');
```

1.6 Image Averaging

图像平均是通过找到K个图像的平均值来获得的。应用于图像去噪。 转换公式：$\overline{g}(x,y)=\frac{1}{K}\sum_{i=1}^{K}g_i(x,y)$，噪声图像定义为$g(x,y)=f(x,y)+\eta(x,y)$， 假设噪声与零均值不相关。（下图只显示了一张噪声图片） 实验结果:

source code:

```
%% 图像平均
img_orig4 = rgb2gray(imread('cat.jpg'));
img_noise1 = imnoise(img_orig4, 'gaussian', 0, 0.01);
img_noise2 = imnoise(img_orig4, 'gaussian', 0, 0.01);
img_noise3 = imnoise(img_orig4, 'gaussian', 0, 0.01);
img_noise4 = imnoise(img_orig4, 'gaussian', 0, 0.01);
img_noise5 = imnoise(img_orig4, 'gaussian', 0, 0.01);
img_average = imlincomb(0.2,img_noise1, 0.2,img_noise2, 0.2,img_noise3, 0.2,img_noise4,
0.2,img_noise5);
figure();
subplot(1, 3, 1);
imshow(img_orig4);
title('Origin');
subplot(1, 3, 2);
imshow(img_noise1);
title('img_noise1');
subplot(1, 3, 3);
imshow(img_average);
title('img_average');
```

1.7 Histogram

1.7.1 Histogram Equalization

直方图均衡化是一种常用的图像增强技术。假设我们有一个主要是黑色的图像。然后它的直方图会向灰度的下端倾斜，所有的图像细节都被压缩到直方图的暗端。如果能将暗端的灰度“拉伸”，生成一个更均匀分布的直方图，那么图像就会清晰得多。

具体做法：1. 求出原图的直方图分布 2.计算原图直方图的累计概率分布 3. 映射，公式可以表示为$f(D_A)=\frac{L}{A_0}\sum_{u=0}^{D_A}H_A(u)$，$A$为原图，$H$为直方图，$L$为灰度级，$A_0$为像素点个数。

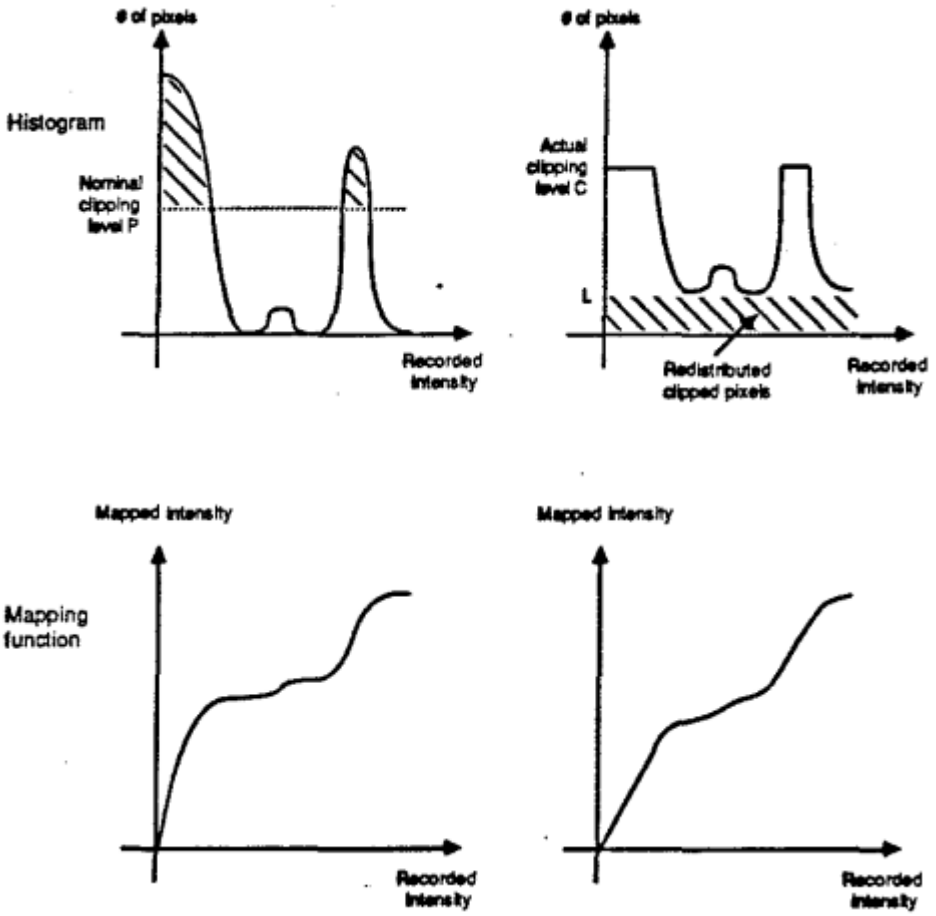
1.7.2 adaptive histogram equalization

直方图均衡化中，是直接对全局图像进行均衡化，是Global Histogram Equalization，而没有考虑到局部图像区域(Local Region)，自适应直方图均衡化（AHE）就是在均衡化的过程中只利用局部区域窗口内的直方图分布来构建映射函数首先，最简单并且直接的想法，对A图像每个像素点进行遍历，用像素点周围W * W的窗口进行计算直方图变换的CDF（累计概率分布），然后对该像素点进行映射。[9]

1.7.3 Contrast Limited Adaptive Hitogram Equalization（CLAHE）

CLAHE相对于AHE，提出了两个改进的地方。

- 第一，提出一种限制直方图分布的方法。考虑图像A的直方图，设定一个阈值，假定直方图某个灰度级超过了阈值，就对 之进行裁剪，然后将超出阈值的部分平均分配到各个灰度级，这个过程可以用下图[10]来进行解释。图中左上图是原来的直方图分布，可以看出有两处峰值，其对应的CDF为左下图，可以看出有两段梯度比较大，变化剧烈。对于之前频率超过了阈值的灰度级，那么就把这些超过阈值的部分裁剪掉平均分配到各个灰度级上，如右上图，那么这会使得CDF变得较为平缓，如右下图。通常阈值的设定可以直接设定灰度级出现频数，也可以设定为占总像素比例，后者更容易使用。由于右下图所示的CDF不会有太大的剧烈变化，所以可以避免过度增强噪声点。[9]



- 第二，提出了一种插值的方法，加速直方图均衡化。首先，将图像分块，每块计算一个直方图CDF，其次，对于图像的每一个像素点，找到其邻近的四个窗口，分别计算四个窗口直方图CDF对该点像素点的映射值，记作$f_{ul}(D), f_{ur}(D), f_{dl}(D), f_{dr}(D)$， 然后进行双线性插值得到最终该像素点的映射值，双线性插值(BiLinear)公式为$f(D) = (1-\Delta y)\left((1-\Delta x)f_{ul}(D) + \Delta x f_{bl}(D)\right) + \Delta y\left((1-\Delta x)f_{ur}(D) + \Delta x f_{br}(D)\right)$

  其中$\Delta x, \Delta y$是像素点相对于左上角窗口中心，即左上角黑色像素点的距离与窗口大小的比值。[9]

实验结果(从左往右依次是原图，HE，AHE，CLAHE）：

source code(hw_1):

```
![a](assets/a.png)//HE,AHE,CLAHE
import numpy as np
import argparse
import cv2

ap = argparse.ArgumentParser()
ap.add_argument('--image', required=True)
args = vars(ap.parse_args())

image = cv2.imread(args["image"])
image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

eh_image = cv2.equalizeHist(image)

ahe = cv2.createCLAHE(clipLimit=0.0, tileGridSize=(8,8))
ahe_image = ahe.apply(image)

clahe = cv2.createCLAHE(clipLimit=10.0, tileGridSize=(8,8))
clahe_image = clahe.apply(image)

cv2.imshow("Histogram Equalization", np.hstack([image, eh_image,
                                                ahe_image, clahe_image]))
cv2.waitKey(0)
```

2. Mask Operations

掩模算子通过在图像上滑动掩模，将掩模值与落在它们下面的像素值相乘并获得总和，来执行掩模与图像的卷积。

可以表示为$g(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t) f(x+s, y+t)$，总和用作图像上掩模中心位置的值，$w(s,t)$即为掩模算子。

2.1 Smoothing operations

图像平滑是指用于突出图像的宽大区域、低频成分、主干部分或抑制图像噪声和干扰高频成分的图像处理方法，目的是使图像亮度平缓渐变，减小突变梯度，改善图像质量。

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \times \frac{1}{9}$$

就是一个用来平滑图像的掩模算子。

实验结果：

source code:

```
img_orig5 = imread('a.png');
H1 = fspecial('average', 3);
img_smooth1 = imfilter(img_orig5, H1);
H2 = fspecial('average', 7);
img_smooth2 = imfilter(img_orig5, H2);
H3 = fspecial('average', 11);
img_smooth3 = imfilter(img_orig5, H3 );
figure();
subplot(2, 2, 1);
imshow(img_orig5);
title('Origin');
subplot(2, 2, 2);
imshow(img_smooth1);
title('kernel=3');
subplot(2, 2, 3);
imshow(img_smooth2);
title('kernel=7');
subplot(2, 2, 4);
imshow(img_smooth3);
title('kernel=11');
```

2.2 Median Filtering

中值滤波是基于排序统计理论的一种能有效抑制噪声的非线性信号处理技术，中值滤波的基本原理是把数字图像或数字序列中一点的值用该点的一个邻域中各点值的中值代替，让周围的像素值接近的真实值，从而消除孤立的噪声点。

实验结果：

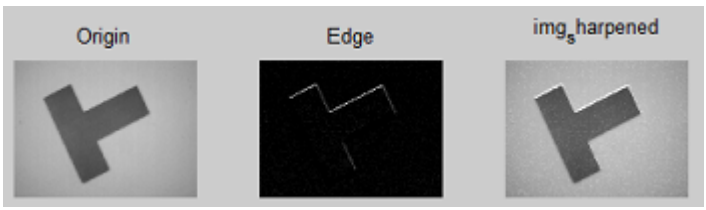
source code:

```
%% 中值滤波
img_orig6 = rgb2gray(imread('lena.png'));
img_noise6=imnoise(img_orig6, 'salt & pepper', 0.02);
img_recover = medfilt2(img_noise6);
figure();
subplot(1, 3, 1);
imshow(img_orig6);
title('Origin');
subplot(1, 3, 2);
imshow(img_noise6);
title('img_salt & pepper');
subplot(1, 3, 3);
imshow(img_recover);
title('img_recover');
```

2.3 sharpening operations

图像锐化*(image sharpening)*是补偿图像的轮廓，增强图像的边缘及灰度跳变的部分，使图像变得清晰，分为空间域处理和频域处理两类。图像锐化是为了突出图像上地物的边缘、轮廓，或某些线性目标要素的特征。这种滤波方法提高了地物边缘与周围像元之间的反差，因此也被称为边缘增强。实验用的sobel算子对图像进行锐化。

实验结果：



source code:

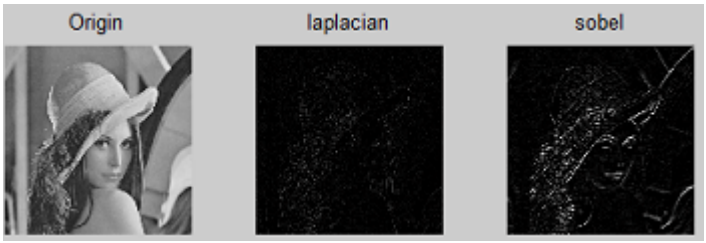
```
%% 锐化
img_orig7 = imread('t.png');
H5 = fspecial('sobel');
edge = imfilter(img_orig7, H5);
sharpened = img_orig7 + edge;
figure();
subplot(1, 3, 1);
imshow(img_orig7);
title('Origin');
subplot(1, 3, 2);
imshow(edge);
title('Edge');
subplot(1, 3, 3);
imshow(sharpened);
title('img_sharpened');
```

2.4 Derivative operations

常见的偏导算子有sobel算子和laplacian算子，sobel算子是一阶偏导算子，laplacian算子是二阶偏导算子。

sobel算子有两个方向$Gx = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix}$，$Gy = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix}$，laplacian常见的算子有四邻域 $\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$，八领域 $\begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix}$.

实验结果：




```
%% 偏导
img_orig7 = rgb2gray(imread('lena.png'));
H6 = fspecial('laplacian');
H7 = fspecial('sobel');
laplacian = imfilter(img_orig7, H6);
sobel = imfilter(img_orig7, H7);
figure();
subplot(1, 3, 1);
imshow(img_orig7);
title('Origin');
subplot(1, 3, 2);
imshow(laplacian);
title('laplacian');
subplot(1, 3, 3);
imshow(sobel);
title('sobel');
```

3. Transform operations

这一节是将图片从空间域转到频域，在频域进行操作，公式表示为$G(u,v) = F(u,v)H(u,v)$，$F(u,v)$是原图经过快速傅里叶变换转换（Fast Fourier Transform, 简称fft）到频域的频谱，$H(u,v)$是在频域执行的操作，$G(u,v)$是在频域处理后的频谱结果，最后$G(u,v)$可以通过快速傅里叶反变换(ifft)得到滤波的图像。

3.1 Low pass filtering

流程：1） 原始正常的图像，加噪处理，得到img_noise； 2） img_noise图像进行傅里叶变换，得到频谱； 3） 对得到的频谱进行理想低通滤波，低于截止频率d_0的通过，高于的抑制； 4） 对滤波后的频谱进行反傅里叶变换，得到滤波后图像。[2]

实验结果：

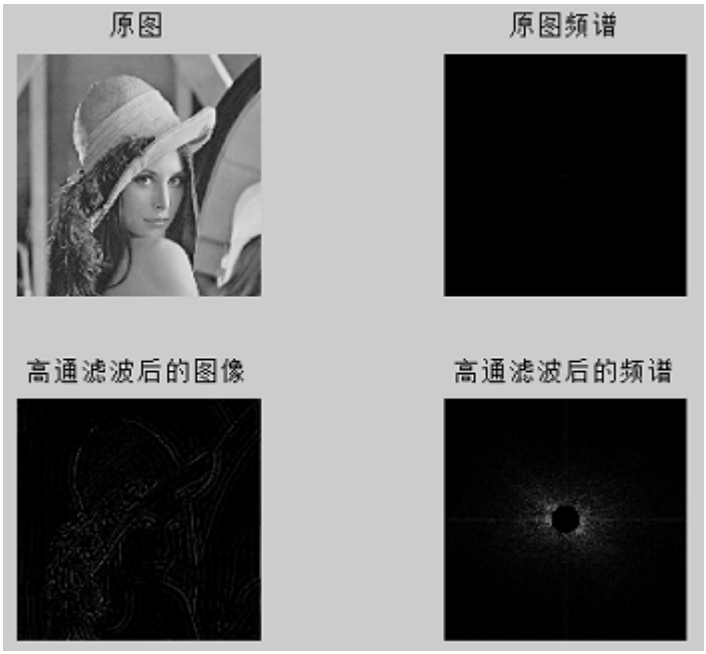
source code:

```
%% 低通滤波, ref[2]
img_origin=rgb2gray(imread('lena.png'));
d0=50;  %阈值
img_noise=imnoise(img_origin,'salt'); % 加椒盐噪声
img_f=fftshift(fft2(double(img_noise)));  %傅里叶变换得到频谱
[m n]=size(img_f);
m_mid=fix(m/2);
n_mid=fix(n/2);
img_lpf=zeros(m,n);

for i=1:m
    for j=1:n
        d=sqrt((i-m_mid)^2+(j-n_mid)^2);    %理想低通滤波，求距离
        if d<=d0
            h(i,j)=1;
        else
            h(i,j)=0;
        end
        img_lpf(i,j)=h(i,j)*img_f(i,j);
    end
end

img_lpf=ifftshift(img_lpf);     %反傅里叶变换
img_lpf=uint8(real(ifft2(img_lpf)));  %取实数部分

subplot(1,3,1);imshow(img_origin);title('原图');
subplot(1,3,2);imshow(img_noise);title('噪声图');
subplot(1,3,3);imshow(img_lpf);title('理想低通滤波');
```

3.2 High pass filtering

高通滤波与低通滤波类似，区别在于低于截止频率的抑制，高于截止频率的通过。

实验结果：



source code:

```
%% 高通滤波
img_origin8 = rgb2gray(imread('lena.png'));
g= fftshift(fft2(double(img_origin8)));
[N1,N2]=size(g);
n=2;
d0=30;
%d0是终止频率
n1=fix(N1/2);
n2=fix(N2/2);
%n1, n2指中心点的坐标，fix()函数是往0取整
for i=1:N1
  for j=1:N2
      d=sqrt((i-n1)^2+(j-n2)^2);
    if d>=d0
        h=1;
    else
        h=0;
    end
    result(i,j)=h*g(i,j);
  end
end
final=ifft2(ifftshift(result));
final=uint8(real(final));
figure();
subplot(2,2,1); imshow(img_origin8); title('原图');
subplot(2,2,2); imshow(abs(g),[]); title('原图频谱');
subplot(2,2,3); imshow(final); title('高通滤波后的图像');
subplot(2,2,4); imshow(abs(result), []); title('高通滤波后的频谱');
```

3.3 Band pass filtering

带通滤波有两个截止频率d_0, d_1，其中d_0是较低的频率，d_1是较高的频率，图像频谱在$[d_0, d_1]$之间的通过，在区间之外的抑制。

实验结果：

source code:

```
%% 带通滤波
img_origin9=rgb2gray(imread('lena.png'));
g= fftshift(fft2(double(img_origin9)));
[N1,N2]=size(g);
n=2;
d0=0;
d1=200;
n1=floor(N1/2);
n2=floor(N2/2);
for i=1:N1
   for j=1:N2
    d=sqrt((i-n1)^2+(j-n2)^2);
    if d>=d0 || d<=d1
        h=1;
    else
        h=0;
    end
    result(i,j)=h*g(i,j);
   end
end
final=ifft2(ifftshift(result));
final=uint8(real(final));
figure();
subplot(2,2,1); imshow(img_origin9); title('原图');
subplot(2,2,2); imshow(abs(g),[]); title('原图频谱');
subplot(2,2,3); imshow(final); title('带通滤波后的图像');
subplot(2,2,4); imshow(abs(result), []); title('带通滤波后的频谱');
```

3.4 Homomorphic filtering

同态滤波是把频率过滤和灰度变换结合起来的一种图像处理方法，它依靠图像的照度/ 反射率模型作为频域处理的基础，利用压缩亮度范围和增强对比度来改善图像的质量。使用这种方法可以使图像处理符合人眼对于亮度响应的非线性特性，避免了直接对图像进行傅立叶变换处理的失真。[8]

同态滤波的基本原理是：将像元灰度值看作是照度和反射率两个组份的产物。由于照度相对变化很小，可以看作是图像的低频成份，而反射率则是高频成份。通过分别处理照度和反射率对灰度值的影响，达到揭示阴影区细节特征的目的。[8]

流程：$S(x,y)-->Log-->FFT-->filter-->IFFT-->Exp-->T(x,y)$[8]

实验结果：


```
%% 同态滤波 ref[8]
img_origin10 = rgb2gray(imread('water.png'));
[M,N]=size(img_origin10);
rL=0.5;
rH=4.7;
c=2;
d0=10;
log_img=log(double(img_origin10)+1);
FI=fft2(log_img);
n1=floor(M/2);
n2=floor(N/2);
for i=1:M
    for j=1:N
        D(i,j)=((i-n1).^2+(j-n2).^2);
        H(i,j)=(rH-rL).*(exp(c*(-D(i,j)./(d0^2))))+rL;%高斯同态滤波
    end
end
G = H.*FI;
final=ifft2(G);
final=real(exp(final));
figure();
subplot(2,2,1); imshow(img_origin10); title('原图');
subplot(2,2,2); imshow(abs(FI),[]); title('原图频谱');
subplot(2,2,3); imshow(final, []); title('同态滤波后的图像');
subplot(2,2,4); imshow(abs(G), []); title('同态滤波后的频谱');
```

4. Coloring Operations

4.1 False coloring

将彩色图像转换为灰度图像是一个不可逆的过程，灰度图像也不可能变换为原来的彩色图像。而某些场合需要将灰度图像转变为彩色图像；伪彩色处理主要是把黑白的灰度图像或者多波段图像转换为彩色图像的技术过程。其目的是提高图像内容的可辨识度。

伪彩色图像的含义是，每个像素的颜色不是由每个基色分量的数值直接决定，而是把像素值当作彩色查找表(事先做好的)的表项入口地址，去查找一个显示图像时使用的R，G，B强度值，用查找出的R，G，B强度值产生的彩色称为伪彩色。

实验结果：

source code:

```
%% 伪彩色
img_origin11 = rgb2gray(imread('lena.png'));
FalseRGB = label2rgb(gray2ind(img_origin11, 255),jet(255));
figure();
subplot(1,2,1); imshow(img_origin11); title('原图');
subplot(1,2,2); imshow(FalseRGB); title('伪彩色');
```

4.2 Full color processing

上面处理的都是灰度图像，如果要处理全彩色图像，则需要对彩色的每个通道分别处理，然后叠加在一起。下面以中值滤波为例，对彩色图像进行处理。

实验结果:

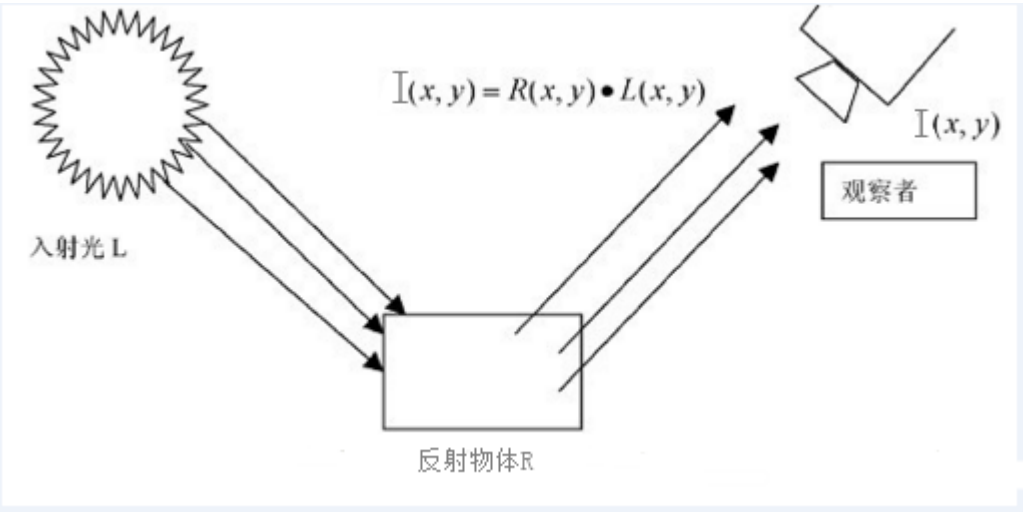
source code:

```
%% 全彩色处理，中值滤波为例
img_orig6 = imread('lena.png');
for i = 1:3
img_noise6(:, :, i) = imnoise(img_orig6(:, :, i), 'salt & pepper', 0.02);
img_recover(:, :, i) = medfilt2(img_noise6(:, :, i));
end
figure();
subplot(1, 3, 1);
imshow(img_orig6);
title('Origin');
subplot(1, 3, 2);
```

```
imshow(img_noise6);
title('img_salt & pepper');
subplot(1, 3, 3);
imshow(img_recover);
title('img_recover');
```

5. Retinex

视网膜-大脑皮层（Retinex）理论认为世界是无色的，人眼看到的世界是光与物质相互作用的结果，也就是说，映射到人眼中的图像和光的长波（R）、中波（G）、短波（B）以及物体的反射性质有关。基于这个理论，可以抽象下图中的公式$I(x,y)=R(x,y)\bullet L(x,y)$，$I(x,y)$代表观察到的图像，$R(x,y)$代表物体的反射属性，$L(x,y)$代表物体表面的光照。



5.1 Single-Scale Retinex

在Retinex理论中，一个假定是光照$I(x,y)$是缓慢变化的，也就是低频的，要从$I(x,y)$中还原出$R(x,y)$，所以可以通过低通滤波器得到光照分量。

具体做法：

- 对源公式两边同时取对数，得到$\log(R)=\log(I)-\log(L)$
- 通过高斯模糊低通滤波器对原图进行滤波，公式为$L=F*I$
- 得到L后，利用第一步的公式得到$log(R)$，然后通过exp函数得到R

5.2 Multi-Scale Retinex

上面是单个尺度下的Retinex算法，当然也存在多尺度的Retinex算法，最为经典的就是3尺度的，大、中、小，既能实现图像动态范围的压缩，又能保持色感的一致性较好。[11]

具体做法: 对每个尺度分别进行单尺度的Retinex算法，将每个尺度的结果相加取平均得到最终结果(这里是简单地取权重相同，当然还可以设立权重不等）.

5.3 Multi-Scale Retinex with Color Restoration

在前面的增强过程中，图像可能会因为增加了噪声，而使得图像的局部细节色彩失真，不能显现出物体的真正颜色，整体视觉效果变差。带色彩恢复因子C的多尺度算法是在多个固定尺度的基础上考虑色彩不失真恢复的结果，在多尺度Retinex算法过程中，通过引入一个色彩因子C来弥补由于图像局部区域对比度增强而导致的图像颜色失真的缺陷，通常情况下所引入的色彩恢复因子C的表达式为：

$$R_{MSRCR_i}(x,y)=C_i(x,y)R_{MSR_i}(x,y) \tag{1}$$

$$C_i(x,y) = f\left[\frac{I_i(x,y)}{\sum_{j=1}^{N} I_j(x,y)}\right] \tag{2}$$

其中，C_i表示为第i个颜色通道地色彩恢复系数，它的作用是调节3个通道颜色的比例，$f(\cdot)$表示颜色空间的映射函数，通常可以采用下面的形式：

$$C_i(x,y) = \beta \log \frac{\alpha I_i(x,y)}{\sum_{j=1}^{N} I_j(x,y)} = \beta\left\{\log[\alpha I_i] - \log\left[\sum_{j=1}^{N} I_j(x,y)\right]\right\} \tag{3}$$

其中，β是增益常数，α是受控制的非线性强度，带色彩恢复的多尺度Retinex算法通过色彩恢复因子C这个系数来调整原始图像中3个颜色通道之间的比例关系，从而把相对有点暗的区域的信息凸显出来，以达到消除图像色彩失真缺陷的目的。处理后的图像局域对比度得以提高，而且其亮度与真实的场景很相似，图像在人们的视觉感知下显得更为逼真。因此，MSRCR算法会具有比较好的颜色再现性、亮度恒常性与动态范围压缩等特性。[12]

5.4 Experiment

source code:

```python
#ref [11]
import argparse
import numpy as np
import cv2


def singleScaleRetinexProcess(img, sigma):
    temp = cv2.GaussianBlur(img, (0, 0), sigma)
    gaussian = np.where(temp == 0, 0.01, temp)
    retinex = np.log10(img + 0.01) - np.log10(gaussian)
    return retinex

def multiScaleRetinexProcess(img, sigma_list):
    retinex = np.zeros_like(img * 1.0)
    for sigma in sigma_list:
        retinex = singleScaleRetinexProcess(img, sigma)
    retinex = retinex / len(sigma_list)
    return retinex

def colorRestoration(img, alpha, beta):
    img_sum = np.sum(img, axis=2, keepdims=True)
    color_restoration = beta * (np.log10(alpha * img) - np.log10(img_sum))
    return color_restoration

def multiScaleRetinexWithColorRestorationProcess(img, sigma_list, G, b, alpha, beta):
    img = np.float64(img) + 1.0
    img_retinex = multiScaleRetinexProcess(img, sigma_list)
    img_color = colorRestoration(img, alpha, beta)
    img_msrcr = G * (img_retinex * img_color + b)
    return img_msrcr

def simplestColorBalance(img, low_clip, high_clip):
    total = img.shape[0] * img.shape[1]
    for i in range(img.shape[2]):
        unique, counts = np.unique(img[:, :, i], return_counts=True)
        current = 0
        for u, c in zip(unique, counts):
            if float(current) / total < low_clip:
                low_val = u
```

```
            if float(current) / total < high_clip:
                high_val = u
            current += c
        img[:, :, i] = np.maximum(np.minimum(img[:, :, i], high_val), low_val)
    return img

def touint8(img):
    for i in range(img.shape[2]):
        img[:, :, i] = (img[:, :, i] - np.min(img[:, :, i])) / \
                       (np.max(img[:, :, i]) - np.min(img[:, :, i])) * 255
    img = np.uint8(np.minimum(np.maximum(img, 0), 255))
    return img

def SSR(img, sigma=300):
    ssr = singleScaleRetinexProcess(img, sigma)
    ssr = touint8(ssr)
    return ssr

def MSR(img, sigma_list=[15, 80, 250]):
    msr = multiScaleRetinexProcess(img, sigma_list)
    msr = touint8(msr)
    return msr

def MSRCR(img, sigma_list=[15, 80, 250], G=5, b=25, alpha=125, beta=46, low_clip=0.01, high_clip=0.99):
    msrcr = multiScaleRetinexWithColorRestorationProcess(img, sigma_list, G, b, alpha, beta)
    msrcr = touint8(msrcr)
    msrcr = simplestColorBalance(msrcr, low_clip, high_clip)
    return msrcr

def main():
    ap = argparse.ArgumentParser()
    ap.add_argument('--image', required=True)
    args = vars(ap.parse_args())

    image = cv2.imread(args["image"])

    ssr = SSR(image)
    msr = MSR(image)
    msrcr = MSRCR(image)

    cv2.imshow("Retinex", np.hstack([image, ssr, msr, msrcr]))
    cv2.waitKey(0)


if __name__ == "__main__":
    main()
```

实验结果：

6. Dark Channel Prior

暗通道先验（Dark Channel Prior）是说在绝大多数非天空的局部区域内，某一些像素至少一个颜色通道具有很低的值，这是何凯明等人基于5000多张自然图像的统计得到的定理。根据这一定理，何凯明等人提出了暗通道先验的去雾算法[13]，

以自然图像和雾气图像为例[14]（左边是原图，右边是暗通道）：




暗通道先验公式如下所示：

$$J^{dark}(\mathbf{x}) = \min_{c\in\{r,g,b\}}\left(\min_{\mathbf{y}\in\Omega(\mathbf{x})}\left(J^c(\mathbf{y})\right)\right) \tag{4}$$

上述公式的意义用代码表达也很简单，首先求出每个像素RGB分量中的最小值，存入一副和原始图像大小相同的灰度图中，然后再对这幅灰度图进行最小值滤波，滤波的半径由窗口大小决定。暗通道先验的理论指出：$J^{\text{dark}} \to 0.$

去雾公认的模型为：

$$\mathbf{I}(\mathbf{x}) = \mathbf{J}(\mathbf{x})\mathrm{t}(\mathbf{x}) + \mathbf{A}(1-\mathbf{t}(\mathbf{x})) \tag{5}$$

其中I为观测强度，J为场景亮度，A为全球大气光，t为描述非散射到达相机的光部分的介质透射率。去雾的目的是从I中恢复无雾的J.

对上述公式进行变形，得到如下公式：

$$\frac{I^c(\mathbf{x})}{A^c} = t(\mathbf{x})\frac{J^c(\mathbf{x})}{A^c} + 1 - t(\mathbf{x}) \tag{6}$$

上标c表示RGB三个通道的意思，假设t在一个窗口下为常数，对上述公式两边同时取两次最小值，得到：

$$\min_{\mathbf{y}\in\Omega(\mathbf{x})}\left(\min_c \frac{I^c(\mathbf{y})}{A^c}\right) = \tilde{t}(\mathbf{x}) \min_{\mathbf{y}\in\Omega(\mathbf{x})}\left(\min_c \frac{J^c(\mathbf{y})}{A^c}\right) + 1 - \tilde{t}(\mathbf{x}) \tag{7}$$

根据暗原色先验理论有：

$$J^{\text{dark}}(\mathbf{x}) = \min_{\mathbf{y}\in\Omega(\mathbf{x})}\left(\min_c J^c(\mathbf{y})\right) = 0 \tag{8}$$

所以前述公式可以化简为：

$$\tilde{t}(\mathbf{x}) = 1 - \min_{\mathbf{y}\in\Omega(\mathbf{x})}\left(\min_c \frac{I^c(\mathbf{y})}{A^c}\right) \tag{9}$$

在现实生活中，即使是晴天白云，空气中也存在着一些颗粒，因此，看远处的物体还是能感觉到雾的影响，另外，雾的存在让人类感到景深的存在，因此，有必要在去雾的时候保留一定程度的雾，这可以通过在上述式子中引入一个在 [0,1] 之间的因子，则上式修正为：

$$\tilde{t}(\mathbf{x}) = 1 - \omega \min_{\mathbf{y}\in\Omega(\mathbf{x})}\left(\min_c \frac{I^c(\mathbf{y})}{A^c}\right) \tag{10}$$

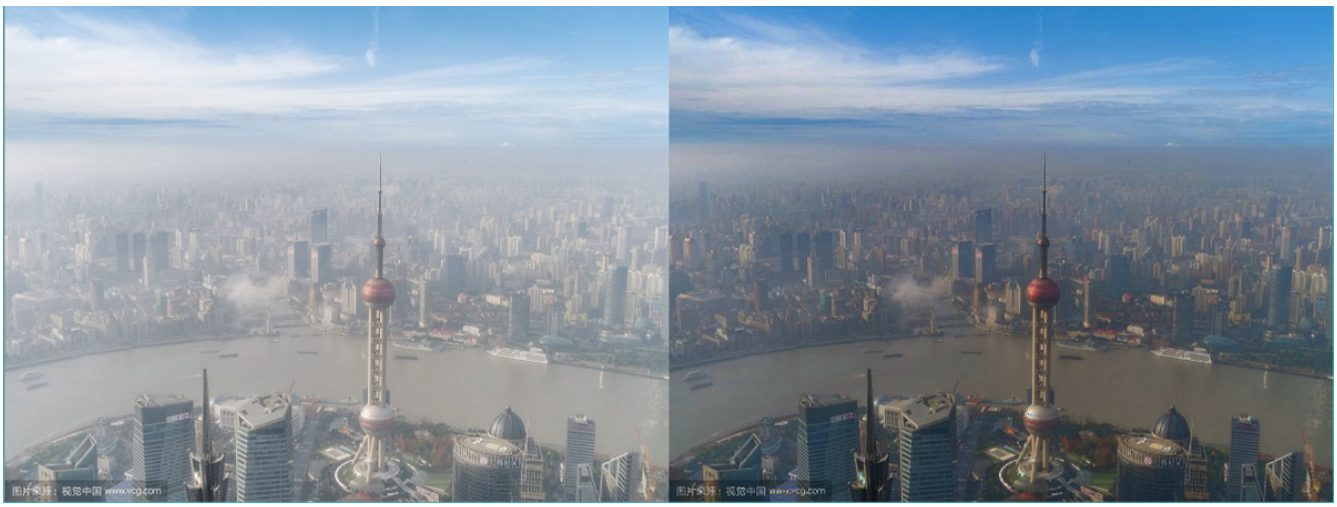
论文中给出的ω等于0.95，对A，论文中给出了一个计算方法： 从暗通道图中按照亮度的大小取前0.1%的像素，在这些位置中，在原始有雾图像I中寻找对应的具有最高亮度的点的值，作为A值。这样A知道了，利用上述公式t也就知道了，在根据原始去雾公式，J也就可以计算了。公式为$\mathrm{J} = (\mathrm{I} - \mathrm{A})/\mathrm{t} + \mathrm{A}$。

当t 的值很小时，会导致J的值偏大，从而使得图像整体向白场过度，因此一般可设置一阈值t0，当t值小于t0时，令t=t0，论文中取t0=0.1。

$$\mathbf{J}(\mathbf{x}) = \frac{\mathbf{I}(\mathbf{x}) - \mathbf{A}}{\max\left(t(\mathbf{x}), t_0\right)} + \mathbf{A} \tag{11}$$

下面实现了一个最简单的版本，简化了很多，没有取窗口，没有用导向滤波等等，更多复杂的操作参考原始论文 [13]。

实验结果：



source code:

```python
import cv2
import argparse
import numpy as np


def hazeRemoval(img, w=0.7, t0=0.1):
    #求每个像素的暗通道
    darkChannel = img.min(axis=2)
    #取暗通道的最大值最为全球大气光
    A = darkChannel.max()
    darkChannel = darkChannel.astype(np.double)
    #利用公式求得透射率
    t = 1 - w * (darkChannel / A)
    #设定透射率的最小值
    t[t < t0] = t0

    J = img
    #对每个通道分别进行去雾
    J[:, :, 0] = (img[:, :, 0] - (1 - t) * A) / t
    J[:, :, 1] = (img[:, :, 1] - (1 - t) * A) / t
    J[:, :, 2] = (img[:, :, 2] - (1 - t) * A) / t
    return J

def main():
    ap = argparse.ArgumentParser()
    ap.add_argument('--image', required=True)
    args = vars(ap.parse_args())

    hazeImage = cv2.imread(args["image"])

    result = hazeRemoval(hazeImage.copy())

    cv2.imshow("HazeRemoval", np.hstack([hazeImage, result]))
    cv2.waitKey(0)


if __name__ == '__main__':
    main()
```

Reference

[1] http://homepages.inf.ed.ac.uk/rbf/HIPR2/hipr_top.htm

[2] https://blog.csdn.net/ytang2_/article/details/75451934

[3] https://baike.baidu.com/item/Sobel%E7%AE%97%E5%AD%90/11000092?fr=aladdin

[4] http://www.eie.polyu.edu.hk/~enyhchan/imagee.pdf

[5] http://www.eletel.p.lodz.pl/mstrzel/imageproc/enhancement1.PDF

[6] https://arxiv.org/ftp/arxiv/papers/1003/1003.4053.pdf

[7] https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8076993

[8] https://blog.csdn.net/scottly1/article/details/42705271#commentBox

[9] https://zhuanlan.zhihu.com/p/44918476

[10] http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.294.8423&rep=rep1&type=pdf

[11] https://www.cnblogs.com/wangyong/p/8665434.html

[12] https://blog.csdn.net/baimafujinji/article/details/73824787#commentBox

[13] http://kaiminghe.com/publications/cvpr09.pdf

[14] http://www.cnblogs.com/Imageshop/p/3281703.html