图像去噪中几类稀疏变换的矩阵表示

摘要

我们在标准的*Cameraman*图加上标准偏差为10的高斯噪声，然后将加噪图像分割为相互重叠大小为8×8的小块，以小块图像为研究对象进行稀疏表示。

针对问题一，分析图像去噪的稀疏表示方法。首先我们分析加性噪声中高斯噪声的特点，然后研究稀疏表示图像去噪的基本原理和方法。稀疏表示的两个关键点是稀疏分解算法和训练字典的构造，我们详细介绍了*OMP*算法的原理和实现方法。最后提出构造小块图像稀疏模型和重构整体图像稀疏模型的思路。

针对问题二，分别建立基于*DCT*、*DWT*、*PCA*和*SVD*稀疏变换的图像去噪模型。对*DCT*的稀疏变换，在一维离散余弦变换公式的基础上，按行、列推导出二维正逆离散余弦变换公式，并构造*DCT*过完备字典，建立离散余弦变换稀疏表示的图像去噪模型。对*DWT*稀疏变换，首先分析离散小波变换和*DBn*小波的运行机制，连续小波函数的离散化、小波系数的阈值计算和小波系数阈值化，然后以*DB*4小波作为小波基建立*DB*4小波去噪模型。对*PCA*稀疏变换，我们根据图像的局部相似性，通过块匹配寻找出相似块作为训练样本，然后利用主成分分析提取信号的重要特征，对每一块进行自适应阈值去噪，并同时实现对边缘细节的有效保护，最后介绍利用*PCA*字典进行图像稀疏表示去噪的流程。对*SVD*的稀疏变换，根据奇异值分解原理推导出奇异值稀疏分解算法的公式，研究并实现$K-SVD$算法高效训练出一个全局性通用字典的方法步骤。

针对问题三，编程实现基于*DCT*、*DWT*、*PCA*和*SVD*算法的稀疏表示。稀疏表示方法的一般实现步骤为图像分块处理、选择自适应完备字典、求得稀疏表示系数、对所有块图像在有重叠的地方做平均处理最终得到去噪图像。编写*MATLAB*程序实现了对原始图像的加噪和分割处理，将分割情况以图片形式显现，然后以第4行第4列图像小块为研究对象，分别实现了四种稀疏表示图像去噪方法，最后采用重叠部分取中值的方法重构为整幅去噪图像。

针对问题四，提出基于*PSNR*和*SSIM*的稀疏去噪性能评价方法。我们从空间域特性和结构化特性两个方面考虑，以*MSE*、*PSNR*、*SSIM*三个指标来综合评价四种稀疏表示去噪方法的优劣。我们对*DCT*、*DWT*、*PCA*和*SVD*算法的稀疏表示图像去噪方法进行评价从而得出基于*SVD*算法的稀疏表示方法要优于其他几种方法。

针对问题五，提出基于$K-SVD$和残差比的图像去噪算法，并用问题四中的稀疏去噪性能评价方法进行评估。我们结合*DCT*、*DWT*、*PCA*和*SVD*的稀疏表示方法的优点，采用过完备*DCT*字典作为$K-SVD$算法的初始化字典D，用*OMP*算法对含噪图像在初始字典D上进行稀疏分解，将残差比作为*OMP*算法迭代的终止条件，建立基于$K-SVD$和残差比的图像去噪算法。最后用问题四中的稀疏去噪性能评价方法进行评估，得出本文算法明显优于以上四种算法。

最后我们对稀疏去噪性能评估方法进行改进，建立*SSIM*优化方案，可以起到更好的评价效果。

关键词：离散余弦变换(*DCT*) 离散小波分析(*DWT*) 主成分分析(*PCA*)
奇异值分解(*SVD*) 峰值信噪比(*PSNR*）

目录

1. 问题重述

假设一幅二维灰度图像 X 受到加性噪声的干扰：$Y = X + N$,Y 为观察到的噪声图像，N 为噪声。通过对于图像 Y 进行稀疏表示可以达到去除噪声的目的。任务：

(1)将加噪处理后的图像 Y 分割为相互重叠的小块 $\{Y_{ij}\}$，分析图像去噪的稀疏表示方法。

(2)分别讨论 $Y_{ij}\left(\sqrt{m}\times\sqrt{m}\right)$ 四类稀疏变换的矩阵表示：离散余弦变换，离散小波变换，主成分分析和奇异值分解。分为以下两种形式：

①$\left(Y_{ij}\right)_{\sqrt{m}\times\sqrt{m}} = U_{\sqrt{m}\times\sqrt{m}} D_{\sqrt{m}\times\sqrt{m}} V_{\sqrt{m}\times\sqrt{m}}$；

②$\left(Y_{ij}\right)_{m\times 1} = D_{m\times k}\alpha_{k\times 1}$(将 $\left(Y_{ij}\right)$ 堆垒为列向量的形式)；

其中，下标为矩阵或者列向量的行列数。

(3)利用 *Cameraman* 图像中的一个小图像块分别实现基于离散余弦变换、离散小波变换、主成分分析和奇异值分解的矩阵表示。

(4)分析稀疏系数矩阵，比较四种方法的稀疏去噪性能。

(5)提出可能的新的稀疏去噪方法。

2. 问题分析

我们以标准的 *Cameraman* 图为研究对象，首先给原图像加上标准偏差为 10 的高斯噪声，然后将加噪图像分割为相互重叠大小为 8×8 的小块，以小块为研究对象进行去噪处理，最后使用各算法将各小块重构为原始图像。

问题一主要研究图像去噪的稀疏表示方法和去噪前的图像预处理。首先我们对加性噪声中的高斯噪声进行了介绍，了解了高斯噪声的特点。然后分析了稀疏表示图像去噪的基本原理和方法。图像的有用信息具备一定的结构特征,而在基于稀疏表示的图像去噪算法中,有用信息的结构特征和字典中原子结构相吻合,而噪声则因为它的随机性而不具备此特征,所以可以很好的利用稀疏表示将噪声与图像进行分离。稀疏表示的两个关键点是稀疏分解算法和训练字典的构造，我们对几类常用的稀疏分解算法进行了介绍，并详细介绍了 OMP 算法的原理和实现方法。最后我们对小块图像和整体图像稀疏模型进行了说明，详细介绍了构造小块图像稀疏模型和重构整体图像稀疏模型的思路。

问题二分别介绍了基于 *DCT* 、 *DWT* 、 *PCA* 和 *SVD* 算法的稀疏变换的图像矩阵表示。

对基于 *DCT* 的稀疏变换，我们首先分析一维离散余弦变换的正逆离散余弦变换公式，对一维变换公式进行按行、列推导，得出二维正逆离散余弦变换公式，然后分析了运用离散余弦变换对图像进行稀疏表示的方法，最后介绍了 *DCT* 字

典的构造方法。

对基于 DWT 的稀疏变换，首先分析离散小波变换和 *DBn* 小波的运行机制，然后以 *DB4* 小波作为小波基提出 *DWT* 小波去噪算法，具体介绍了连续小波函数的离散化、小波系数的阈值计算和小波系数阈值化，最后对小波字典的构造进行推导。

对基于 *PCA* 的稀疏变换，我们根据图像的局部相似性，通过块匹配寻找出相似块作为训练样本，然后利用主成分分析提取信号的重要特征，对每一块进行自适应阈值去噪，使含有细节内容丰富的子块的阈值小一些，含有细节内容较少的子块的阈值大一些，在去除噪声的同时实现对边缘细节的有效保护，最后介绍利用 *PCA* 字典进行图像稀疏表示去噪的流程。

对基于 *SVD* 的稀疏变换，首先我们根据奇异值分解原理推导出奇异值稀疏分解算法的公式 $A = USV^T = \sum_{i=1}^{r} \alpha_i u_i v_i^T$，然后介绍 $K-SVD$ 算法如何高效地训练出一个全局性通用字典，并对 $K-SVD$ 算法的实现进行详细介绍。

问题三实现了基于 *DCT* 、 *DWT* 、 *PCA* 和 *SVD* 算法的稀疏表示。稀疏表示方法的一般实现步骤为图像分块处理、选择自适应完备字典、求得稀疏表示系数、对所有块图像在有重叠的地方做平均处理最终得到去噪图像。编写 *MATLAB* 程序实现了对原始图像的加噪和分割处理，将分割情况以图片形式显现，然后以第 4 行第 4 列的图像小块为研究对象，分别实现四种稀疏表示方法进行图像去噪处理，然后重构为整幅去噪图像。

问题四提出基于 *PSNR* 和 *SSIM* 的稀疏去噪性能评价方法。我们从空间域特性和结构化特性两个方面考虑，以 *MSE* 、 *PSNR* 、 *SSIM* 三个指标来综合评价四种稀疏表示去噪方法的优劣。我们利用评价方法对基于 *DCT* 、 *DWT* 、 *PCA* 和 *SVD* 算法的稀疏表示图像去噪方法进行评价。

问题五提出基于 $K-SVD$ 和残差比的图像去噪算法，并用问题四中的稀疏去噪性能评价方法进行评估。我们结合 *DCT* 、 *DWT* 、 *PCA* 和 *SVD* 算法的稀疏表示方法的优点，采用过完备 *DCT* 字典作为 $K-SVD$ 算法的初始化字典 D，用算法 *OMP* 算法对含噪图像在初始字典 D 上进行稀疏分解，将残差比作为 *OMP* 算法迭代的终止条件，建立了基于 $K-SVD$ 和残差比的图像去噪算法。最后用问题四中的稀疏去噪性能评价方法进行评估。。

3. 模型假设

(1)假设该文中二维灰度图像 X 只受到加性噪声的干扰。

(2)假设受到的加性噪声的干扰为高斯噪声，其标准偏差为 10.

(3)假设在稀疏表示过程中产生的误差不会引起后续操作的太大反应。

(4)假设在图片读取或存储时无像素点的丢失。

4. 符号说明

符号	符号说明
$g(x,y)$	加入噪声后的图像函数
$f(x,y)$	未被噪声污染的图像函数
$n(x,y)$	噪声函数
$I_0(x)$	没被噪音污染的灰度值
$I(x)$	有噪音污染的灰度值
$v(x)$	噪声的灰度值
k	信号稀疏系数

注：表中没有列出的符号文中使用时会给予说明。

5．模型的建立与求解

5.1 模型准备

5.1.1 加性去噪

我们在获取图像的信息时,避无可避的都会出现各种外因或内因的干扰,致使图像夹杂着许多噪声,使获取的信息不完整甚至是错误的,所谓噪声,通俗意义上说就是对我们没用的信息,在图像处理过程中,必须有效的抑制噪声,提高图像质量和视觉效果。

根据噪声是否与图像相关可分为加性噪声和乘性噪声两类,加性噪声是和图像信号独立、不相关的噪声，乘性噪声又叫卷积噪声，是和图像信号是相关的噪声，其存在情况以及强度大小情况均依赖于图像信号。根据噪声的概率分布可分为高斯噪声、椒盐噪声、瑞利噪声、伽马（爱尔兰）噪声等。

题目中假设一幅二维灰度图像 X 受到加性噪声的干扰，所以我们着重介绍加性噪声中的高斯噪声。

加性噪声是和图像信号独立、不相关的噪声，如开关接触噪声、传输过程中的信道噪声、电子元器件内部产生的热噪声和散粒噪声等，其模型如下:

$$g(x,y)=f(x,y)+n(x,y) \tag{1}$$

式中，$g(x,y)$ 为掺杂了噪声后的图像，$f(x,y)$ 为未被噪声污染的图

像，$n(x,y)$是噪声。高斯噪声是加性噪声的一种，它不论是在空间域中还是在频域中，都很容易在数学上进行处理，所以在很多应用中经常用到这种噪声。以高斯噪声为例，噪声的灰度值z符合高斯分布。若z的期望为u，方差为σ，那它的概率密度可表示为式(2)，在高斯噪声中，方差越大表示噪声对图像影响越大。

$$P(z)=\frac{1}{\sqrt{2\pi\sigma}}e^{-(z-\mu)}/2\sigma^2 \tag{2}$$

我们使用标准偏差为 10 的高斯噪声来给原始图像加噪，然后对加噪后的图像进行基于四类方法的稀疏表示，达到去噪的效果。

5.1.2 稀疏表示图像去噪

信号表示，就是将原始信号在某一已知函数或者矢量集上来进行表示，在图像处理过程中，图像表示是一个非常重要的过程。所谓信号的稀疏表示，是指将信号投影到正交变换基时，绝大部分变换系数的绝对值很小，所得到的变换向量是稀疏或者近似稀疏的。由于图像的有用信息具备一定的结构特征，而且在基于稀疏表示的图像去噪算法中，有用信息的结构特征和字典中原子结构相吻合，而噪声则因为它的随机性而不具备此特征，所以可以很好的利用稀疏表示将噪声与图像进行分离。

1. 稀疏表示理论

稀疏表示是尽量少的使用基函数对原始信号进行表示的过程。对于给定的信号矢量$g_i\in R^N\left(i=1,2,\cdots,k,N\text{表示一维信号的长度}\right)$组成的集合构成了一个过完备原子库即字典$D$，$g$则称为基或原子。由于字典的冗余性，矢量$g$，不满足线性无关，那对于任意给定长度的信号空间，都可以用少数的原子来表示原始信号的主要成分，如下所示：

$$f=\sum_{j=0}^{k-1}a_k g_k \tag{3}$$

上式中，k表示原始信号分解的原子的个数，a_k表示f信号在g_k上的分量。

a_k并不唯一，但一般选择最为稀疏的系数，也就是说在矩阵中只含有很少量的非零系数，并且仅凭这些系数就足以表现出原始信号的特征，为了得到该稀疏解，一般可采用L_0范数进行度量，因此可建立如下的稀疏表示模型：

$$\min\|\alpha\|_0, s.t. f=\sum_{j=0}^{k-1}a_k g_k \tag{4}$$

上式中，$\min\|\alpha\|_0$表示α的L_0范数，所谓L_0范数，就是L_p范数中p趋向于 0 的情况，也即α中非零元素的数量。如果将字典中的原子当作列向量依次进行排列，

则可得到一个 $N\times L\left(L\gg N\right)$ 的矩阵, 记为 Φ, 此时上述模型可表示为:

$$\min\left\|\alpha\right\|_0, s.t. f=\Phi\alpha \tag{5}$$

如图 1 所示为超完备稀疏表示的完整示意图。从图中可看出 α 只含有极少数非零数, 因此是稀疏的。(各个小方格中空白则为零)

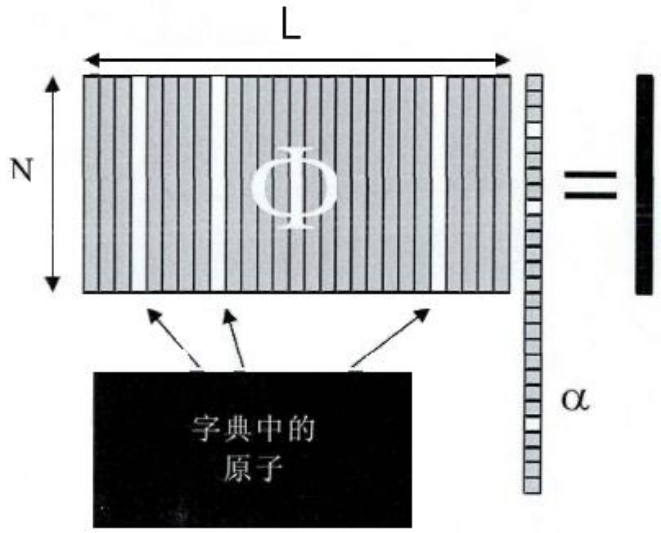

图 1　超完备系数表

2. 唯一性

从上图中还可以得知，过完备稀疏表示模型等价于一个欠定系统。所以，可将其转为求解欠定系统的解，判断最优解是否是最稀疏且唯一的。

①定理 1(唯一性)对于字典 D, $spark\left(D\right)$ 是指字典 D 中任一组原子线性相关时所需的最小个数, 如果欠定方程的某一稀疏解满足条件 $\left\|\alpha\right\|_0<spark\left(D\right)/2$, 那表示此解为唯一且稀疏的。

②如果方程 $f=\Phi\alpha$ 的解满足 $\left\|\alpha\right\|_0<\frac{1}{2}\left(1+1/\mu\left(\Phi\right)\right)$，那么该解必定是有可能最稀疏。

以上两个定理相辅相成, 定理 1 的唯一性若满足, 定理 2 则一定可以确定唯一且稀疏的解。

3. 稀疏分解算法

作为过完备稀疏表示理论的基本问题之一, 稀疏分解算法是指如何获取信号在超完备字典下的最稀疏表示。目前，信号的稀疏分解算法主要可分三类：松弛优化算法、贪婪追踪算法和组合优化算法。我们使用正交匹配追踪(*OMP*)稀疏分解算法

正交匹配追踪(*OMP*)算法是在匹配逼近(*MP*)算法的基础上提出的, 匹配逼近(*MP*)算法的基本原理是根据某一规则在迭代过程中从相应的字典里选择用来进行稀疏表示的信号的某一列向量(原子), 最后得到图像的稀疏表达。而正交匹配追踪(*OMP*)算法在匹配逼近(*MP*)算法的基础上进行了优化, 在每一步

迭代的过程中,对选择的原子进行施密特正交化处理,正交化后的原子构成了相应的空间,然后将信号投影到该空间上,即可得到信号在每个已经选择的原子上残余分量、投影分量,然后再依次循环对残余的分量进行分解。

正交匹配追踪(*OMP*)算法的过程如下:

先初始化 $\varphi_0 = d_{j0}$，对 $m \geq 0$，正交匹配追踪挑选 d_{jm}，使得

$$\left|\left(R_f^m, d_{jm}\right)\right|^2 \geq \mu \max_{j \in K} \left|\left(R_f^m, d_j\right)\right| \tag{6}$$

Gram – Schmidt 算法将 d_{jp} 关于 $\left\{\varphi_{jp}\right\}_{0 \leq p \leq m}$ 正交化，定义为

$$\varphi_m = d_{jm} - \sum_{p=0}^{m-1} \frac{\left(d_{jm}, \varphi_p\right)}{\left\|\varphi_p\right\|} \varphi_p \tag{7}$$

将余项 R_f^m 投影到 φ_m 上，得到：

$$R_f^m = \frac{\left(R_f^m, \varphi_m\right)}{\left\|\varphi_m\right\|^2} \varphi_m + R_f^{m+1} \tag{8}$$

将此方程对 $0 \leq m < k$ 求和，得：

$$f = \sum_{m=0}^{k-1} \frac{\left(R_f^m, \varphi_m\right)}{\left\|\varphi_m\right\|^2} \varphi_m + R_f^k = P_{Vk} + R_f^k \tag{9}$$

其中，P_{Vk} 是在 $\left\{\varphi_{jm}\right\}_{0 \leq p \leq m}$ 所在的空间 V_k 上的正交投影。*Gram – Schmidt* 算法保证 $\left\{d_{jm}\right\}_{0 \leq m \leq M}$ 也是 V_k 的一组基。对任意 $k \geq 0$，余项 R_f^k 是 f 正交与 V_k 的部分。对于 $m = k$，由式得

$$\left|R_f^m, \varphi_m\right| = \left|R_f^m, d_{jm}\right| \tag{10}$$

因 V_k 的维数为 k，故存在 $M \leq N$ 使得 $f \in V_M$，从而 $R_f^M = 0$。将其代入并令 $k = M$，得：

$$f = \sum_{m=0}^{M-1} \frac{\left(R_f^m, d_{jm}\right)}{\left\|\varphi_m\right\|^2} \varphi_m \tag{11}$$

也就是说，对于上述过程进行有限 M 次的迭代即可收敛，它是 f 在一个正交向量族上的分解，故

$$\left\|f\right\|^2 = \sum_{m=0}^{M-1} \frac{\left|\left(R_f^m, d_{jm}\right)\right|^2}{\left\|\varphi_m\right\|^2} \varphi_m \tag{12}$$

为了将 f 在原来的字典向量 $\left\{d_{jm}\right\}_{0 \leq m \leq M}$ 上展开，必须将基做一些修改。三角

形 $Gram-Schmidt$ 关系式（7）可以转化为用 $\left\{d_{jp}\right\}_{0\le p\le m}$ 将 φ_m 展开：

$$\varphi_m=\sum_{p=0}^{m}b[p,m]d_{jp} \tag{13}$$

将此表达式代入的

$$f=\sum_{p=0}^{M-1}a[jp]d_{jp} \tag{14}$$

其中

$$a[jp]=\sum_{p=0}^{m}b[p,m]d_{jp}\frac{\left(R_f^m,d_{jm}\right)}{\left\|\varphi_m\right\|^2} \tag{15}$$

分析可知，由于在前几次迭代中追踪法挑选出的原子是近似正交的，正交追踪的优势不明显。但当迭代次数持续增大时，正交匹配追踪（OMP）算法会因余项范数快速下降而收敛，正交匹配追踪（OMP）算法运算简单，且保证了每次迭代的最优性，减少了迭代的次数。

4. 过完备字典

对于 M 维的矢量空间，空间中的任何一个矢量 $f\in F$ 均可以用一组矢量 $\Phi=\left\{\varphi_i\right\}_{i=1}^{N}\left(N\ge M\right)$ 来线性表示，表达式如式(16)所示

$$f=\Phi\alpha=\left[\varphi_1,\varphi_2,\cdots,\varphi_N\right]\left[\alpha_1,\alpha_2,\cdots,\alpha_N\right]^T=\sum_{i=1}^{N}\alpha_i\varphi_i \tag{16}$$

我们要对图像用一组正交基进行线性表示，就要对字典进行构造，因此字典选择的优劣也间接影响了图像的稀疏表示。信号稀疏表示的两大主要任务就是字典的生成和信号的稀疏分解，对于字典的选择，一般有分析字典和学习字典两大类。分析字典是由特定的解析式或者固定的正交基产生的，在整个运算过程是一成不变的，存在很多缺点。而学习字典是通过样本训练生成的。针对给定的训练样本集，训练出能够稀疏表示此样本集的字典，下次遇到同样类型的新样本时，不早需要重新训练。若待处理图像是本身，这样得到的学习型字典对于待处理图像来说是自适应，更匹配于图像的各种特征变化。

一般的构造学习字典的方法有最优方向（MOD）、主成分分析（PCA）、$K-SVD$、离散余弦变换（DCT）等算法。下面的篇幅，我们将介绍过完备离散余弦变换（DCT）字典、（$DB4$）小波字典、主成分分析（PCA）字典和奇异值分解（SVD）字典的构造方法。

5.1.3 小块图像和整体图像稀疏模型

1. 小块图像稀疏模型

将图像 Y 分割为相互重叠的小块 Y_{ij}，大小为 $m*m$，将此分块图像排成一个列向量 $x\in R^N$。首先需定义一个字典 $D\in R^{n\times N}\left(K>n\right)$，可知字典 D 具备冗余性，字典 D 已知，因此可建立小块图像 u 在字典 D 下的稀疏模型表达式，即：

$$\hat{a} = \arg\min_{\alpha} \|\alpha\|_0 \ s.t. D\alpha = X \tag{17}$$

为了使上述模型更加有利于运算, 将约束式 $D\alpha = X$ 用 $\|D\alpha - y\|_2^2 \le \varepsilon$ 代替并定义稀疏度 L, 使其满足 $\|\hat{a}\| \le L \le n$。, 表示块图像 u 中的对应字典 D 的原子的个数不能超过 L, 为了后续方便, 定义了一个三元组 (ε, L, D) 来表示此块图像的稀疏模型。

现有块图像 x 满足三元组分块模型, 而 y 则表示掺杂有均值为 0 方差为 σ 的高斯白噪声的块图像, 由贝叶斯最大后验概率 (MAP) 则可建立式(18)的模型:

$$\hat{a} = \arg\min_{\alpha} \|\alpha\|_0 \ s.t. \|D\alpha - y\|_2^2 \le T \tag{18}$$

在上式中, T 的值和 ε, σ 相关, 没有噪声的图像能够有准确的稀疏表示, 而掺杂了噪声的图像则会影响图像的稀疏表示, 上式先估计含噪图像中的干净部分的稀疏表示系数, 进而可恢复干净的图像以达到去噪的目的。去除噪声后的块图像表达式为:

$$\hat{x} = D\hat{a} \tag{19}$$

如果将式(17)中的约束项用惩罚项代替可得到模型如式(19)所示, 式中 μ 为惩罚因子即:

$$\hat{a} = \arg\min_{\alpha} \|D\alpha - y\|_2^2 + \mu\|\alpha\|_0 \tag{20}$$

2. 整体图像稀疏模型

对于整体图像 Y, 采用分块的方法对整体图像进行分块重叠。假定是提取图像分块的矩阵, 那么则表示相对应的分块图像。可知每个块图像都是满足上小节的 (ε, L, D) 模型, 所以整体图像的稀疏表示模型可由块图像的稀疏模型得到, 如式(20)所示

$$\left\{\hat{a}_{ij}, \hat{X}\right\} = \arg\min_{\alpha_{ij}} \lambda\|X - Y\|_2^2 + \sum_{i,j}\|D\alpha_{ij} - R_{ij}X\|_2^2 \tag{21}$$

在式(21)所示的模型中 $\lambda\|X - Y\|_2^2$ 表示原图像 X 与含噪图像 Y 的总体相似度, 第二项和第三项分别表示稀疏性约束和误差约束, $D\alpha_{ij}$ 则表示对应块图像 $R_{ij}X$ 经重构后所得到的近似值。

5.2 四类稀疏变换的图像矩阵表示

首先使用 *MATLAB* 程序将图像分为 R 个相同大小且重叠的区域。每个小图像块的大小为 8×8，重叠度为 25%。我们将先以某个小块为研究对象，进行去

噪处理，然后利用整合图像稀疏模型将已去噪的小块图像合成为整体图像。

5.2.1 基于离散余弦变换的矩阵表示

1. 离散余弦变换简介

一维正向离散余弦变换 N 采样的公式为

$$F(k)=C(k)\sqrt{\frac{2}{N}}\sum_{N=0}^{N-1}f(n)\cos\frac{\pi(2n+1)}{2N} \tag{22}$$

其中，

$$C(k)=\begin{cases}\dfrac{1}{\sqrt{2}}, & k=0\\ 1, & 1\le k\le N-1\end{cases} \tag{23}$$

函数 $f(n)$ 表示输入信号第 x 次的采样值，$F(k)$ 表示对于 $k=0,1,2,\cdots,n-1$ 的 DCT 系数。

一维逆离散余弦变换的公式为：

$$f(n)=\frac{1}{\sqrt{N}}C(0)+\sqrt{\frac{2}{N}}\sum_{N=0}^{N-1}f(n)\cos\left[\frac{\pi(2n+1)u}{2N}\right] \tag{24}$$

二维 DCT 可以使用水平一维 DCT 计算，然后计算垂直信号，因为 DCT 是一个可分离的函数。

首先按行方式引用一维离散余弦变换，即，

$$f(u,v)=\sqrt{\frac{2}{N}}C(u)\sum_{y=0}^{N-1}f(x,y)\left[\frac{\pi(2x+1)u}{2N}\right] \tag{25}$$

然后按列方式应用一维离散余弦变换，即，

$$f(u,v)=\sqrt{\frac{2}{M}}C(u)\sum_{y=0}^{M-1}f(x,y)\left[\frac{\pi(2x+1)u}{2M}\right] \tag{26}$$

结合按行方式和按列方式得到的离散余弦变换公式，得到二维正向离散余弦变换公式为

$$F(u,v)=\frac{2}{\sqrt{MN}}C(u)C(v)\sum_{x=0}^{N-1}\sum_{y=0}^{M-1}f(x,y)\cos\left[\frac{\pi(2x+1)u}{2N}\right]\cos\left[\frac{\pi(2y+1)u}{2M}\right]$$

函数 $f(x,y)$ 代表二维信号在第 y 行的第 x 次采样点，$F(u,v)$ 是对于 $u=0,1,\cdots,N-1$ 和 $v=0,1,\cdots,M-1$ 的二维变换系数。

该公式符合题目中给出的公式(27)的要求

$$\left(Y_{ij}\right)_{\sqrt{m}\times\sqrt{m}}=D_{\sqrt{m}\times\sqrt{m}}U_{\sqrt{m}\times\sqrt{m}}V_{\sqrt{m}\times\sqrt{m}} \tag{27}$$

同理，二维逆离散余弦变换公式为

$$f(x,y)=\frac{2}{\sqrt{MN}}\sum_{x=0}^{N-1}\sum_{y=0}^{M-1}C(u)C(v)\,f(u,v)\cos\left[\frac{\pi(2x+1)\mu}{2N}\right]\cos\left[\frac{\pi(2y+1)\mu}{2M}\right]$$

2.离散余弦变换的矩阵表示

以Y_{ij}小块为研究对象，对这些小图像块进行正规化得到，然后对每个正规化后的图像块使用离散余弦变换。离散余弦变换具有能量集中特性，可以将图像的能量都集中在变换矩阵的低频部分，即变换矩阵的左上角部分，利用这一特性，取变换矩阵最左上角4×4的数据，并把第一个数据舍去，因为在正规化后，这一维度的数据将不包含任何信息。这样可以从每个小图像块得到一个15×1的低维度的特征向量$x_{r,i}$，然后对这些特征向量进行稀疏分解得到稀疏系数$\alpha_{r,i}$。

对每个区域，使用下面的式子来获得该区域的特征向量：

$$h_r=\frac{1}{n_r}\sum_{i=1}^{n_r}\alpha_{r,i} \qquad (28)$$

式中n_r为第r个区域的取样小图像块总数。

由于在稀疏表示时α中可能含有负数，而如果直接用这种带负数的稀疏系数代入到式(28)中会损失很多信息，对此问题找到了 3 种解决方法：

（1）使用非负的稀疏分解

（2）将负数系数与正数系数分为两个向量，这样我们要处理的数据维度将增加一倍，但实际数据量的增加不大；

（3）简单的对每个低维特征向量求绝对值。

最终通过实验发现简单易行的第 3 种方法就可以得到良好的结果。所以我们选用第三种方法来处理负数的系数。

图像二维 *DCT* 变换有许多优点:图像信号经过变换后， 变换系数几乎不相关，经过反变换重构图像，信道误差和量化误差将象随机噪声一样分散到块中的各个像素中去，不会造成误差累积;并且变换能将数据块中的能量压缩到为数不多的部分低频系数中去(即 *DCT* 矩阵的左上角)。

假设图像信号能够被一些基本余弦函数线性表示，即图像经 *DCT* 后，对图像平滑区域覆盖真实图像能量的 *DCT* 系数个数较少，运用这些系数可以将图像有效地估计出来。图像经过二维离散余弦变换被转换成一个 *DCT* 系数矩阵，但图像并未得到真正的压缩。 *DCT* 域中图像稀疏表示依赖于图像的特性，具有将能量集中于少数低频频率系数、各系数互不相关、高频能量衰减很快且能量较小等性质，这些性质使得 *DCT* 在图像压缩与重建中得到了广泛的应用。

3. *DCT* 字典

DCT 字典由 *DCT* 变换获得，给定序号$x(n),n=0,1,\cdots,N-1$，其离散余弦变换：

$$X_c(0)=\frac{1}{\sqrt{N}}\sum_{n=0}^{N-1}x(n) \qquad (29)$$

$$X_c(0)=\sqrt{\frac{2}{N}}\sum_{n=0}^{N-1}x(n)\cos\frac{(2n+1)k\pi}{2N} \tag{30}$$

如果写成矩阵则是 $X_c=C_N x$

其中 C_N 是 $N*N$ 变换矩阵，其行向量为余弦基。

5.2.2 基于 *DB*4 小波变换的矩阵表示

1. 离散小波变换和 *DBn* 小波

在实际应用中, 考虑最多的是离散小波变换, 而不是连续小波变换, 这主要是离散小波变换更容易在计算机上实现″离散小波变换相对应傅里叶级数, 正如连续小波变换对应于傅里叶变换。所谓的离散即指尺度和位移的离散, 而对待分析信号和分析小波中的时间变量 t 并没有被离散化。

尺度离散化:取一个合理的值 a_0, 使尺度因子 a 只取 a_0 的整数幂, 即 a 仅取

$$a_0^0=1,a_0^1,a_0^2,\cdots,a_0^j,\cdots \tag{31}$$

位移离散化:当尺度 $a=a_0^0$ 时, 取位移 $b=b_0$。各位移为 $a=a_0^j$ 时, 相应取

$b=ka_0^jb_0$。

在这些离散位置的小波伸缩平移系构成

$$\left\{a_0^{-\frac{j}{2}}\psi\left(a_0^j\left(t-ka_0^jb_0\right)\right),j,k\in\mathbb{Z}\right\}=\left\{a_0^{-\frac{j}{2}}\psi\left(a_0^{-j}t-kb_0\right),j,k\in\mathbb{Z}\right\}$$

这里 $\mathbb{Z}$ 表示全部整数的集合。用 *CWT* 公式得到在离散尺度和位移处的小波变换

$$WT_x=\left(a_0^j,kb_0\right)=\int x(t)\psi^*_{a_0^jkb_0}(t)\,dt \tag{32}$$

最典型的 a_0,b_0 取值是 $a_0=2,b_0=1$，则得到小波伸缩平移系为

$$\psi_{jk}(t)\triangleq 2^{-\frac{j}{2}}\psi\left(2^{-\frac{j}{2}}t-k\right) \tag{33}$$

这种情况称尺度因子为 2，在相应的离散尺度和位移点的小波变换值记为

$$WT_x=(j,k)=\left\langle x(t),\psi_{jk}(t)\right\rangle \tag{34}$$

DBn 小波是正交小波的一种， *DBn* 中的 N 表示 *DB* 小波的阶次, $N=2-10$。*DB*1 小波即为 *Haar* 小波。我们将选用 *DB*4 小波来作为小波基进行去噪。

2. *DB* 小波去噪算法

在数学上，小波去噪问题的本质是一个函数逼近，即如何在由小波母函数伸缩和平移版本所展成的函数空间中, 根据提出的衡量准则, 寻找对原信号的最佳

逼近, 以完成原信号和噪声信号的区分, 可以表述为：

$$\begin{cases} \beta_{opt}=\arg\min\limits_{\beta\in T}\left(\left\|\beta(f)-f_s\right\|\right) \\ f_{opt}=\beta_{opt}(f) \\ f(t)=f_s(t)+f_n(t) \\ I=\left\{f\,\middle|\,f\text{为实际信号}\right\},\ W=span\left\{(\psi_{2i})_{j=1}^{J},\psi_{2j}\right\} \\ T=\left\{\beta\,\middle|\,\beta\text{为}I\to W\text{的函数空间映射}\right\} \end{cases} \tag{35}$$

其中： opt 代表最优解， f_s 为原信号， f_n 为噪声信号。

由此可见, 小波去噪方法也就是寻找从实际信号空间到小波函数空间的最佳映射, 以便得到原信号的最佳恢复。基于 DB 小波变换的图像去噪算法，流程如图 2 所示。

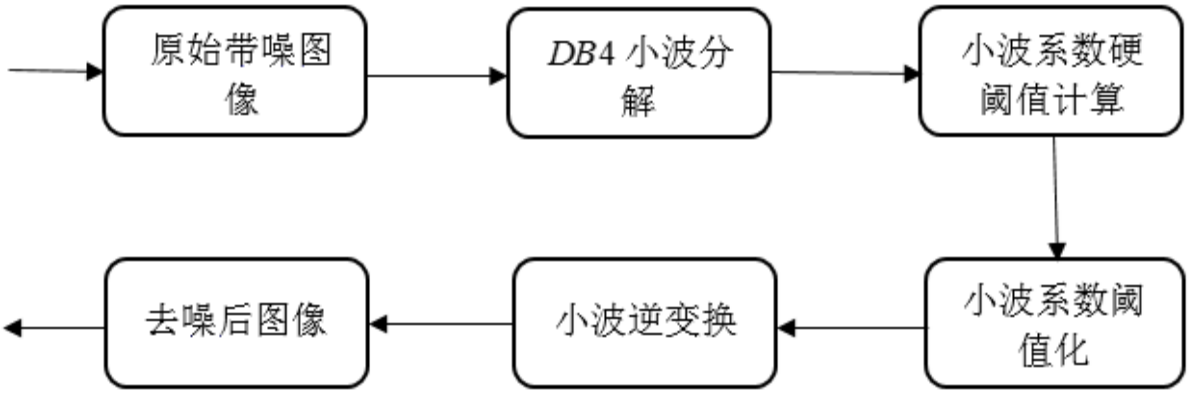

图 2 小波去噪流程图

(1)连续小波函数的离散化

将连续小波函数 $\psi_{a,b}(x)=\frac{1}{\sqrt{a}}\psi\left(\frac{x-b}{a}\right)$ 中的伸缩因子 a 和平移因子 b 按下列规律离散取样： $a=a_0^{-m};b=nb_0a_0^{-m}$ ，则得离散小波函数：

$$\psi_{m,n}(x)=a_0^{\frac{m}{2}}\psi\left(a_0^m x-nb_0\right) \tag{36}$$

其中， a,b 离散， x 仍连续变化。小波变换由下式给出：

$$\begin{aligned} W_{(m,n)}&=\int_{-\infty}^{\infty}f(x)\overline{\psi}_{m,n}(x)dx=a_0^{\frac{m}{2}}\int_{-\infty}^{\infty}f(x)\overline{\psi}\left(a_0^m x-nb_0\right)dx \\ &=\left\langle f,\psi_{m,n}\right\rangle \end{aligned} \tag{37}$$

现在考虑反变换，由于 a,b 是离散取样，可用整数 m,n 来表示，所以原反变

换式中的二重积分变形为求和计算：

$$f(x)=\sum_{m=-\infty}^{\infty}\sum_{n=-\infty}^{\infty}W(m,n)\psi_{m,n}(x) \tag{38}$$

现在的情况相当于把 $f(x)$ 进行级数展开，因此上式被称为“小波级数展开”。进一步地，如果 x 也离散化，则有如下的“离散小波变换”；

正交换：$W(m,n)=\sum_{i=-\infty}^{\infty}f(i)\overline{\psi}_{m,n}(x)$

反交换：$f(i)=\sum_{m=-\infty}^{\infty}\sum_{n=-\infty}^{\infty}W(m,n)\psi_{m,n}(i)$

(2)小波系数的阈值计算：根据 DJ 阈值，即由式(39)计算每一层的小波系数阈值 δ_k，其中，σ_k 为噪声标准方差，N 为原始图像像素点的个数。

$$\delta_k=\sigma_k\sqrt{2\ln N}\qquad k=(1,2,3) \tag{39}$$

$$\sigma_k=\frac{Median\left(\left|W_{ij}\right|\right)}{0.6745},W_{ij}\in subband\quad W_{k,HH},k=(1,2,3) \tag{40}$$

(3)小波系数阈值化：由于硬阈值方法可以很好保留图像边缘，所以阈值函数选用“硬阈值函数”。根据每一层小波系数阈值，利用式(41)对每层小波系数进行阈值化，得到新的小波系数阈值 $W_{k\delta}$。

$$W_{k\delta}=\begin{cases}W, & |W|\geq\delta_k\\ 0, & |W|<\delta_k\end{cases}\qquad k=(1,2,) \tag{41}$$

3. 小波字典的构造

在小波字典的构造中，首先利用小波分解提取图像训练样本的低频子图像，然后提取该子图像的特征向量作为字典原子保存。由全部训练样本的特征向量构造小波字典，有效地降低了特征维数、提高了识别效率。

假设训练样本小波分解后，低频子图像集为 $\{I_1,\cdots,I_n\}$，且 $I_i\in R^{h\times l},i=1,\cdots,n$，$n$ 为训练样本总数。首先将图像进行中心化处理，即 $\tilde{I}_i=I_i-\tilde{I}$，其中 $\tilde{I}=\frac{1}{n}\sum_{i=1}^{n}I_i$ 为所有训练样本的均值图像。训练图像的协方差矩阵 D 定义为

$$D=\sum_{i=1}^{n}\tilde{I}_i^T\tilde{I}_i=\sum_{i=1}^{n}\left(I_i-\bar{I}\right)^T\left(I_i-\bar{I}\right)\in R^{l\times l} \tag{42}$$

投影矩阵 V 可由 D 的前 r 个较大特征值 $\lambda_1,\cdots\lambda_r$ 对应的特征向量 $[v_1,\cdots,v_r]$ 组成。因此，对于任意样本图像 $I_i\in R^{h\times l}$，其中 $2DPCA$ 特征

$$F_{2DPCA}=\left[f_1,\cdots,f_r\right]=\tilde{I}_i\left[v_1,\cdots,v_r\right]=\left(I_i-\bar{I}\right)V\in R^{h\times r} \tag{43}$$

5.2.3 基于主成分分析变换的矩阵表示

1. 主成分分析稀疏表示

在高斯白噪声中，主成分分析法能很好的对高维数据进行降维处理, 简化了复杂的高维计算难度。我们根据图像的局部相似性，通过块匹配寻找出相似块作为训练样本，然后利用主成分分析提取信号的重要特征，最后对每一块进行自适应阈值去噪，使含有细节内容丰富的子块的阈值小一些，含有细节内容较少的子块的阈值大一些，在去除噪声的同时实现对边缘细节的有效保护，获得视觉效果良好、性噪比更高的恢复图像。

图像的每一个像素都是由其位置及其灰度值来描述，图像的局部信息可以表示为相邻像素组成的图像块，我们利用主成分分析（PCA）对图像的每个像素点进行去噪，就需要找到一个与其对应的合适的训练集 X 。我们以像素点 x 为中心，取大小为 $K\times K$ 的小块作为中心块，在以 x 为中心的大窗 $L\times L$ 内选取与中心块相似的块，我们将这些块均用列向量表示。

为了从噪声图像中获得更好的原始图像 X ，我们对每一个像素点进行 PCA 去噪处理，我们将寻找块相似块的方法称为块匹配法，在窗 $L\times L$ 内有 $\left(L-K+1\right)^2$ 个 X 变量的相似训练块，记 $I_0\left(x\right)$ 为没被噪声污染的灰度值，$I\left(x\right)$ 为有噪声污染的灰度值，$v\left(x\right)$ 为噪声的灰度值，由于噪声与信号时相互独立的，容易计算出方差为：

$$e_i\approx\frac{1}{m}\sum\left(I\left(x\right)-I\left(x_i\right)^2\right)+2\sigma^2 \tag{44}$$

若

$$e_i<T+2\sigma^2 \tag{45}$$

成立，我们就选择 x_i 为一个训练样本，我们利用这种方法来选取训练样本，从所选的样本集中最近似的 n 个样本构成样本集 X ，X 为 $m\times n$ 矩阵，用行向量表示为，x_i 为 X 的第 $i+1$ 行，同时我们记没被噪声污染的灰度值为：

$$I_0\left(X\right)=\left[I_0\left(x_0\right)^T,I_0\left(x_1\right)^T,I_0\left(x_2\right)^T,\cdots,I_0\left(x_{m-1}\right)^T\right]^T \tag{46}$$

被噪声污染的灰度值为：

$$I\left(X\right)=\left[I\left(x_0\right)^T,I\left(x_1\right)^T,I\left(x_2\right)^T,\cdots,I\left(x_{m-1}\right)^T\right]^T \tag{47}$$

将上式 (46) (47) 分别中心化，我们近似得到：

$$\bar{I}\left(X\right)=\bar{I}_0\left(X\right)+v\left(X\right) \tag{48}$$

由于噪声与原始图像相互独立，且噪声为高斯白噪声，由计算我们近似协方

差可以表示为：$\Omega \gg \Omega_0 + \Omega_v$

式中：$\Omega_0 = (1/n)\bar{I}_0(X)\bar{I}_0(X)^T = \Phi_0 \Lambda_0 \Phi_0^T$

Φ_0 为 $m \times m$ 的正交阵；Λ_0 为对角阵，对角元素是 Φ_0 的特征根，我们知道噪声的协方差是一个 $m \times m$ 的对角阵，它的所有对角元素为 σ^2，从而我们有 $\Omega_v = \sigma^2 I$，I 为单位 $m \times m$ 的单位矩阵，所以有：

$$\Omega = \Phi \Lambda \bar{\Phi} \approx \Omega_0 + \Omega_v = \Omega_0 \left(\Lambda_0 + \sigma^2 I \right) \Phi_0^T = \Phi_0 \Lambda_0 \Phi_0^T \tag{49}$$

由上式可知，Ω 与 Ω_0 有相同的特征向量，即 $\Phi_0^T = \Phi^T$。因此我们得到 *PCA* 的变换矩阵为：

$$P_0 = \Phi_0^T = \Phi^T \tag{50}$$

由 *PCA* 变换公式，我们将 P_0 作用于 $\bar{I}(X)$，于是得到：

$$Y = P_0 \bar{I}(X) = P_0 \bar{I}_0(X) + P_0 v(X) \tag{51}$$

经过 *PCA* 变换后，高斯噪声能量分布在整个变换域，而图像的能量集中分布在几个重要分量组成的子空间里。然后利用图像的稀疏成分重建图像，则重建的图像即为去除噪声后的图像。

2. *PCA* 字典

PCA 基在图像的稀疏表示方面有着重要应用，运用 *PCA* 基进行图像稀疏表示去噪的基本思想是, 使每个图像块学习一个 *PCA* 基, 然后对图像块稀疏表示去噪。

假设一组信号 $\{f\}$ 满足同一种高斯分布模型, 那么通过对角化协方差矩阵可以求解出这组信号的 *PCA* 基, 即

$$\sum\nolimits_k = E\left[f_i f_i^T \right] = B_k S_k B_k^T \tag{52}$$

其中 B_k 是 *PCA* 基，$S_k = diag\left(\lambda_1^k, \lambda_2^k, \cdots, \lambda_N^k\right)$ 是对角矩阵。

假设一共有 K 个方向的 *PCA* 基, 那么信号分别在 K 个 *PCA* 基上求解稀疏表示系数, 可以求出 K 组系数, 由于信号总对应一个最优方向的 *PCA* 基, 分别将每一个方向上求解的稀疏表示系数代入如下公式, 求解最优方向。

$$k = \arg\min_k \left(\left\| AB_k \alpha_k - y \right\|^2 \right) + \sigma^2 \sum_{m=1}^{N} \frac{\left| \alpha_k \right|}{\lambda_k^m} + \sigma^2 \log \left| \sum\nolimits_k \right| \tag{53}$$

选择方向 k 对应的系数作为信号稀疏表示系数。

我们使用的 *PCA* 方向基字典, 主要是 K 个方向 *PCA* 基的联合字典, 可以构造仅有方向性的灰度图像, 获得图像的方向规律性对稀疏表示是重要的, 字典中大

多数主导原子是边缘和规则的纹理原子,能够很好的稀疏表示图像轮廓和规则的纹理信息。

利用 *PCA* 字典进行图像稀疏表示去噪的流程如下：

⑴输入： *PCA* 方向基字典 *D*,每个 *PCA* 基对应的特征值,原图像,随机观测矩阵,稀疏度 K.

输出:重构信号

⑵ *Step*1:构造灰度图像:过 64*64 大小的全白图像的中心点,画一条直线,直线斜率角度从 0 度到170度,以10度一间隔,位于直线下方的点取值为1,其余点保持0值不变,得到18个方向的黑白图像;

⑶ *Step*2:得到训练样本:分别对每个方向的黑白图像隔点取8×8的块,得到每一个方向的训练样本;

⑷ *Step*3： 获得 *PCA* 方向基字典：分别对每个方向的训练样本进行 *PCA* 分解,得到 *PCA* 正交基和特征值,每个方向分别保留前 *K* 个最大特征值和其对应的基,得到特征值向量和 *PCA* 方向基字典;

⑸ *Step*4:分块观测：对输入图像的每一个图像块使用随机观测矩阵观测,

⑹ *Step*5： 求解最优方向:对每一块的块观测向量 分别在18个 *PCA* 方向基上对稀疏表示模型利用 *EM* 算法求解稀疏表示系数,并记录每一块最优方向;

⑺ *Step*6:得到最优稀疏表示图像:将每一块的最优方向对应的稀疏表示系数与此方向的 *PCA* 方向基相乘,得到图像块,将图像块依次排列组成图像;

⑻ *Step*7:统计重新学习:以非边缘块每一块为中心,统计其周围相邻8块的方向,如果有5个以上的块方向一致,则保存此方向,查看中心块的方向与此记录的方向是否一致,如果不一致则需用此方向的 *PCA* 基根据稀疏表示模型来重新求解稀疏表示系数,将系数与其对应的 *PCA* 方向基相乘,得到修改的图像块;

⑼ *Step*8:输出图像:优化的图像块依次排列组成图像,并输出图像。

5.2.4 基于奇异值分解的矩阵表示

1.奇异值分解

SVD 分解是一种将矩阵对角化的数值方法，是线性代数中最有用和最有效的工具之一，在统计分析、图像处理，数字水印中被广泛应用。从线性代数的角度看，一幅图像可以看成是由许多非负标量组成的矩阵。若一幅图像用 A 表示，定义为 $A \in R_{m\times n}$，其中 R 表示实数域。则矩阵 A 的奇异值分解定义如下：

$$A = USV^T \tag{54}$$

其中， S 是一个对角矩阵 $diag(\sigma_1, \sigma_2, \cdots, \sigma_n, 0, \cdots 0)$，对角线上的元素成为矩

阵 A 的奇异值。 r 是 A 的秩，它等于且满足 $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > \sigma_{r+1} = \cdots = \sigma_m = 0$，$U$ 和 V 分别为 $m \times m$ 和 $n \times n$ 的正交阵，U 和 V 的每列分别称为 A 的左奇异向量和右奇异向量。

数值分析当中的奇异值分解是将矩阵进行对角化的数值算法。当对图像进行 SVD 分解之后，所得图像的奇异值表现了图像的本质特性。$A \in R^{m \times n}\left(r \geq 1\right)$ 的非零奇异值为 $\alpha_1, \alpha_2, \alpha_3, \cdots, \alpha_r$，并记 $\sum_r = diag\left(\alpha_1, \alpha_2, \alpha_3, \cdots, \alpha_r\right)$，则有正交阵 $U^{m \times m}$ 和正交阵 $V^{m \times m}$，使矩阵 A 的奇异值分解可以表示为

$$A = USV^T = \sum_{i=1}^{r} \alpha_i u_i v_i^T \tag{55}$$

其中

$$S = \begin{pmatrix} \sum_r & 0 \\ 0 & 0 \end{pmatrix}_{m \times n} \tag{56}$$

式中：U 的第 i 列为与 A 相对应的 α_i 的左奇异向量，用 u_i 表示 V 的第 i 列为与 A 相对应的 α_i 的右奇异向量，用 v_i 表示 S 为奇异值对角阵。

由式(55)得 A 的分解式为

$$A = M_1 + M_1 + \cdots + M_1 = \alpha_1 u_1 v_1^T + \alpha_2 u_2 v_2^T + \cdots + \alpha_r u_r v_r^T \tag{57}$$

式(57)说明图像 A 是由 r 幅子图像 $u_1 v_1^T, \cdots, u_r v_r^T$ 叠加而成，而相应的奇异值 α 即是 r 幅子图对 A 图的相应叠加因子。由于较小奇异值对图像文件的贡献较小，图像大部分能量集中在前面较大奇异值所对应的子图像中，所以如果按照奇异值由大到小的顺序选取 q 个值 $\left(q < r\right)$ 来生成矩阵，近似表示原矩阵 A，即 $A_q = \sum_{i=1}^{q} \alpha_i u_i \vec{v}_i^T$c，就达到了图像去噪的目的。选取的奇异值个数 q 越少，说明组成 A_q 的数据量越小，压缩率越高；q 越接近于 r，压缩率越低，还原图像越接近于真实图像。

对图像矩阵做奇异值分解，得到的奇异值具有以下显著的特性：图像奇异值的稳定性非常好，即当图像被施加小的扰动时图像的奇异值不会有很大的变化；奇异值对应于图像的亮度特性，而奇异向量对则表征了图像的几何特性，奇异值所表现的是图像的内蕴特性而非视觉特性，反映的是图像矩阵元素之间的关系；图像分解得到的奇异值序列中第一个奇异值要比其它的大得多，忽略这些较小的奇异值项重构的图像质量不会发生太大的退化。如果改变第一个较大的奇异值会导致图像视觉质量的严重下降。

2. $K-SVD$字典

$K-SVD$算法，该算法可以高效地训练出一个全局性通用字典，其核心环节是奇异值分解。该算法与MOD算法的思想类似，但优势在于：一方面，字典更新是以一种更简单高效的方式逐个原子进行的，避免了矩阵求逆运算；另一方面能同时更新当前原子和相关的系数，加速了字典学习过程。

$K-SVD$算法分两步实现。

首先假设$D\in R^{n\times K}$是过完备字典，$x\in R^n$是训练样本信号，$\alpha\in R^K$是训练样本信号的稀疏表示系数向量，$X=\left\{x_i\right\}_{i=1}^N$是$N$个训练样本信号集合，$m$是每个训练样本对应的系数向量。

第一步，采用稀疏分解算法求解下式：

$$\min_{\alpha_i}\left\|x_i-D\alpha_i\right\|_2^2 \quad s.t.\forall_i,\left\|\alpha_i\right\|_0\le T_0, i=1,2,\cdots,N \tag{58}$$

式中T_0表示稀疏表示系数α_i中非零分量个数的最大值。

第二步，根据更新的系数，对字典D进行迭代训练。设d_k为字典D待更新的第k列向量，即第k个原子，此时信号集的分解形式可表示为

$$\left\|X-DA\right\|_F^2=\left\|X-\sum_{j\neq k}d_j\alpha_j^T-d_k\alpha_k^T\right\|_F^2=\left\|E_k-d_k\alpha_k^T\right\|_F^2 \tag{59}$$

其中，α_k^T是d_k对应的系数矩阵A中的第k行向量，而E_k是去掉d_k后信号集X的分解误差，$\left\|\cdot\right\|_F^2$是矩阵的谱范数。为了准确高效的进行SVD分解，只选择非零有效的部分，并作了如下定义:

$$\omega_k=\left\{i\middle|1\le i\le K,\alpha_k^T(i)\neq 0\right\},\alpha_k^R=\alpha_k^T\Omega_k,X_k^R=X\Omega_k,E_k^R=E_k\Omega_k \tag{60}$$

$K-SVD$算法主要分为两个步骤，算法框架图如图 3 所示

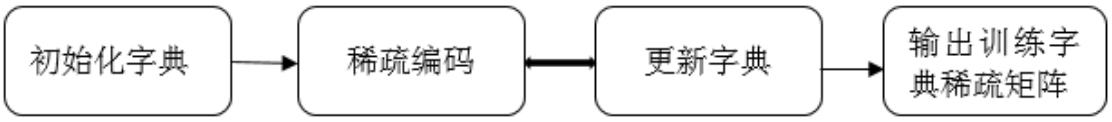

图 3 $K-SVD$算法框架

(1)稀疏编码步骤。使用跟踪算法对信号进行稀疏分解

(2)字典更新步骤。假设稀疏变换矩阵给定，采用奇异值分解方法对残差矩阵进行分解，更新字典中对应的原子；与此同时更新对应的稀疏表示系数。再回到稀疏编码步骤，循环完成整体的迭代次数。

$K-SVD$算法的细节如下所示

1). 参数初始化：设$X=Y,D=C$超完备DCT字典 。

2). J 次迭代

(1)稀疏编码

采用 OMP 算法，针对图像训练集 $Z=\left\{z_j\right\}_{j=1}^{M}$ 中的每一个图像分块 z_j 求解 α_j

$$\forall j \quad \min_{\alpha j}\left\|\alpha_j\right\|_0 \quad s.t. \quad \left\|z_j-D\alpha_j\right\|_2^2 \le (c\sigma)^2 \tag{61}$$

(2)字典更新

通过下面的步骤，对字典 D 中的每一列 $l=1,2,\cdots,k$，依次升级：

（a）找出所有满足 $\omega_l=\left\{j\middle|\alpha_j(l)\neq 0\right\}$ 的图像分块 z_j

（b）对每个 $j\in\omega_l$，计算残差 $e_j^l=z_j-\sum_{m\neq l}dm\alpha_j(m)$

（c）设残差矩阵 $E_l=\left\{e_j^l\right\}_{j\in\omega_l}$

（d）对 E_l 进行奇异值分解得到 $E_l=U\Lambda V^T$，用矩阵 U 的第一列更新字典 D 中第一列为 d_l，系数 $\alpha_j(l)_{j\in\omega}\Lambda(l,l)$ 乘以 V 的第一列。

5.3 四种稀疏表示算法的分析与实现

5.3.1 稀疏表示方法的一般实现步骤

在基于稀疏表示的图像去噪算法中，有用信息的结构特征和字典中原子结构相吻合，而噪声则因为它的随机性而不具备此特征，所以可以很好的利用稀疏表示将噪声与图像进行分离。去噪的具体流程如图 4 所示。

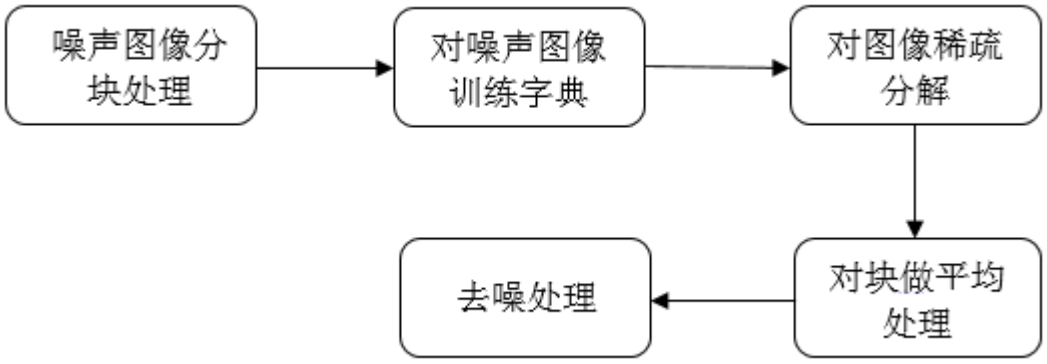

图 4 图像去噪流程

图像去噪主要有以下四个思路：

(1)对含噪声的图像进行分块，各个分块部分可以重叠；

(2)选取适量个数的块图像，选择自适应完备字典，根据不同的稀疏表示方法选择不同的字典；

(3)分别使用各稀疏表示方法得到每个图像块在字典上的稀疏表示系数；

(4)对所有块图像在有重叠的地方做平均处理最终得到去噪图像。

假设字典 D 已知，则需要计算最终输出图像 X 与局部稀疏表示系数 α，在此使用块协调最小化算法进行计算。首先进行初始化，令 $X=Y$，再寻找最优的局部

稀疏表示系数。在对每个图像块进行迭代的过程中,一旦误差小于预先设定的阈值T时则停止,就可以得到所需要的图像块的稀疏表示系数α_{ij}。按照以上算法进行计算,当所有的图像块稀疏表示系数都确定后,再依据(62)所示模型去更新X。

$$X = \arg\min_X \lambda \|X - Y\|_2^2 + \sum_{i,j} \left\| D\alpha_{ij} - R_{ij}X \right\|_2^2 \tag{62}$$

式(62)是一个较为简单的二项式，它的近似解为：

$$X = \left(\lambda I + \sum_{i,j} R_{ij}^T R_{ij} \right)^{-1} \left(\lambda Y + \sum_{i,j} R_{ij}^T D\alpha_{ij} \right) \tag{63}$$

一般图像去噪包括两个基本步骤：

（1）计算出块图像的稀疏表示系数;

（2）对去除噪声后的块图像做平均处理得到最终图像。重复(1)(2)过程知道得到想要的去噪效果。

5.3.2 图像分割和加噪处理

首先对图像进行分块处理，使用*MATLAB*程序将图像分为R个相同大小且重叠的区域。每个小图像块的大小为8×8，重叠度为25%。我们选取的图像大小为 512*507，如果一步分解为8×8大小的小块很难实现。所以我们先分解为64*64 大小的小块共 81 个，然后再将每个小块分解为8×8大小的目标块。第一步分解如图 5 所示，然后我们选用第一步分解的第 4 行 4 列的块图为例继续进行分解如图 6 所示。

然后我们对原图像加上标准偏差为 10 的高斯噪声，按上面方法进行分割。仍然以第一步分解的第 4 行 4 列的块图为例继续进行分解如图 7 所示。

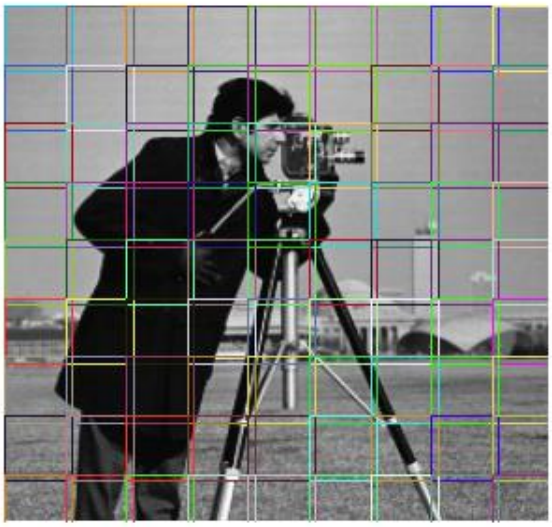

图 5 原图像重叠分割

图 6 对 4 行 4 列. *bmp* 图像分割

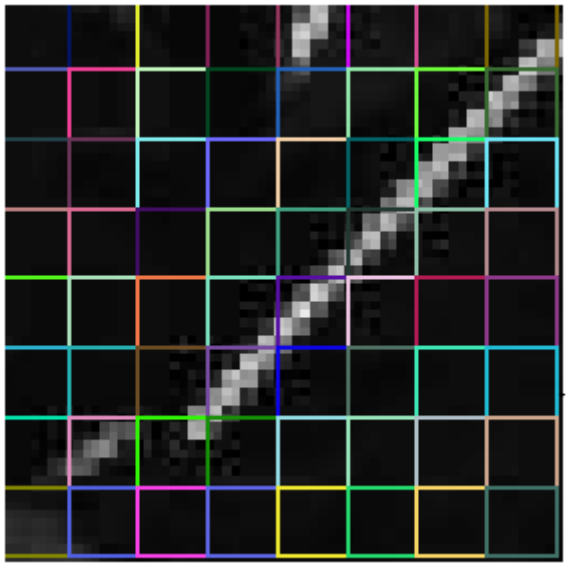

图 7 4 行 4 列高斯噪声图

我们选取图 6 图 7 中的第 5 行第 5 列小块为研究对象，具体研究各种系数分解方法对其作用，然后选用一种评价性能的方法进行去噪性能评估。第 5 行第 5 列小块图像的灰度值如下，原图像矩阵为 $f(x,y)$，加噪图像矩阵为 $g(x,y)$。

$$f(x,y)=\begin{bmatrix} 0 & 64 & 90 & 207 & 173 & 112 & 83 & 0 \\ 8 & 82 & 76 & 211 & 190 & 86 & 63 & 19 \\ 69 & 180 & 201 & 93 & 93 & 15 & 21 & 16 \\ 83 & 183 & 235 & 85 & 85 & 19 & 12 & 11 \\ 181 & 125 & 91 & 22 & 22 & 9 & 15 & 17 \\ 193 & 118 & 106 & 21 & 6 & 13 & 14 & 16 \\ 122 & 18 & 22 & 12 & 12 & 13 & 14 & 15 \\ 106 & 21 & 23 & 11 & 11 & 12 & 13 & 14 \end{bmatrix}$$

$$g(x,y)=\begin{bmatrix} 23 & 86 & 64 & 203 & 200 & 99 & 124 & 17 \\ 0 & 49 & 51 & 185 & 180 & 127 & 59 & 22 \\ 84 & 137 & 207 & 105 & 83 & 0 & 48 & 0 \\ 68 & 181 & 255 & 43 & 134 & 4 & 0 & 7 \\ 192 & 129 & 88 & 23 & 21 & 17 & 0 & 50 \\ 209 & 146 & 70 & 15 & 5 & 0 & 4 & 0 \\ 81 & 30 & 0 & 0 & 0 & 15 & 6 & 0 \\ 82 & 13 & 56 & 43 & 0 & 15 & 47 & 0 \end{bmatrix}$$

下面我们将先以此小块为研究对象，进行去噪处理，然后利用整合图像稀疏模型将已去噪的小块图像合成为整体图像。*MATLAB* 程序见附件。

5.3.3 四种稀疏表示算法的实现

5.3.3.1 离散余弦变换稀疏表示的实现

首先对重叠分割后的小块图像进行离散余弦变换，然后用 *DCT* 字典训练，在对某一小块去噪时，需要周围的图像小块构造过完备字典。最后将各个重叠分割的小块组合在一起，重合部分采用取中值的方法。用 *MATLAB* 程序进行模拟，得到第 4 行第 4 列的原图、加噪图和去噪图如图 8 所示，其 *DCT* 字典如图 9，因为第 4 行第 4 列的块图很小，为了观察在这里已经对其做过放大处理。

图 8 第 4 行第 4 列原图、加噪图和 *DCT* 去噪图

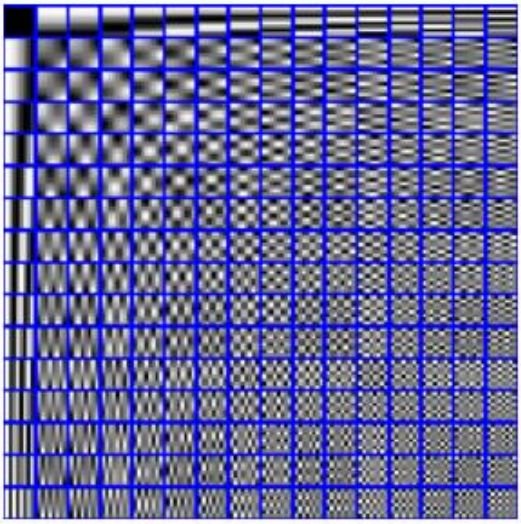

图 9 第 4 行第 4 列 *DCT* 字典

对去噪后的各小块进行重新整合，在重叠部分取中值，得到的原图像的加噪图像和经 *DCT* 稀疏表达去噪后的图像如图 10 所示，对整幅图训练的字典如图 11 所示。

图 10 原图、噪声图和 *DCT* 去噪图

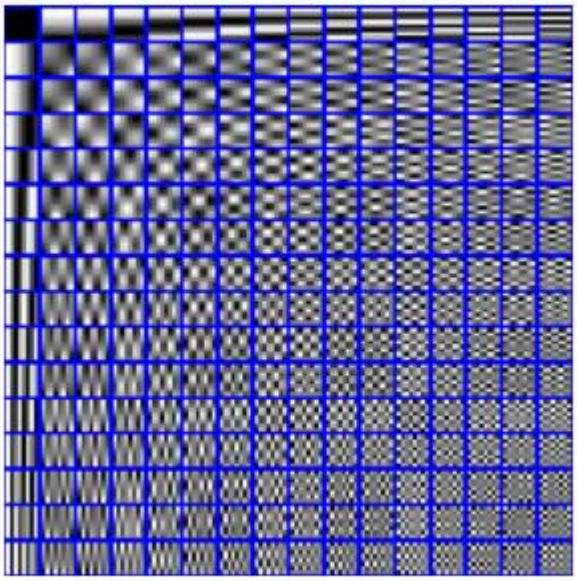

图 11 整幅图 *DCT* 字典

5.3.3.2 基于 *DB*4 小波变换的稀疏表示的实现

首先对重叠分割后的小块图像进行以 *DB*4 小波为小波基的小波变换，然后用小波字典训练。最后将各个重叠分割的小块组合在一起，重合部分采用取中值的方法。用 *MATLAB* 程序进行模拟，得到第 4 行第 4 列的原图、加噪图和去噪图如图 12 所示。

图 12 第 4 行第 4 列的原图、加噪图和 *DWT* 算法去噪图

对去噪后的各小块进行重新整合，在重叠部分取中值，得到的原图像的加噪图像和经 *DB*4 小波稀疏表示去噪后的图像如图 13 所示。

图 13 原图、噪声图和 *DWT* 去噪图

5.3.3.3 基于主成分分析变换的稀疏矩阵表示实现

首先对重叠分割后的小块图像进行主成分分析特征值提取，然后用 *PCA* 字典进行训练。最后将各个重叠分割的小块组合在一起，重合部分采用取中值的方法。用 *MATLAB* 程序进行模拟，得到第 4 行第 4 列的原图、加噪图和去噪图如图 14 所示。

图 14 原图、加噪图和 *PCA* 去噪图

对去噪后的各小块进行重新整合，在重叠部分取中值，得到的原图像的加噪图像和经稀疏表示去噪后的图像如图 15 所示。

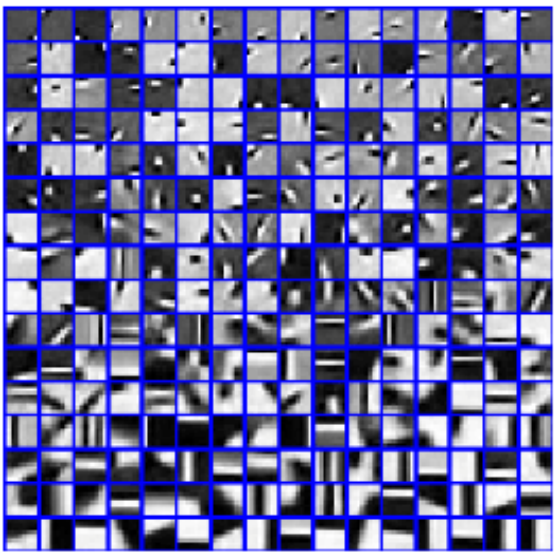

图 15 整幅图 *PCA* 字典

5.3.3.4 基于奇异值分解的稀疏矩阵表示的实现

首先对重叠分割后的小块图像进行奇异值稀疏表示，然后用 *SVD* 字典进行自适应训练。最后将各个重叠分割的小块组合在一起，重合部分采用取中值的方法。用 *MATLAB* 程序进行模拟，得到第 4 行第 4 列的原图、加噪图和去噪图如图 16 所示，其 *SVD* 字典如图 17。

图 16 第 4 行第 4 列原图、加噪图和 *SVD* 去噪图

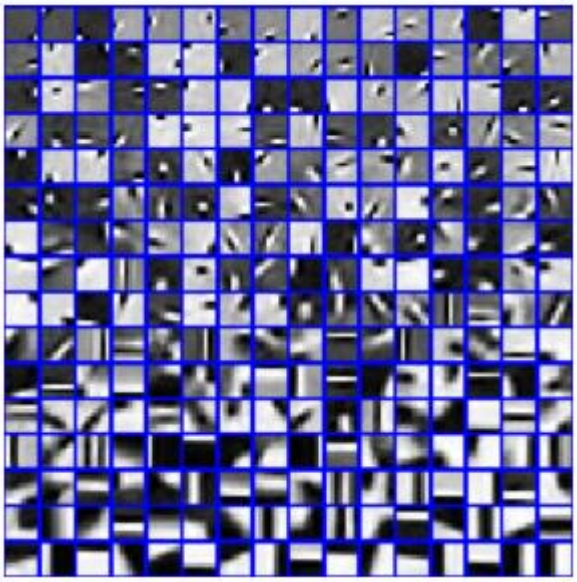

图 17 第 4 行第 4 列 *SVD* 字典

对去噪后的各小块进行重新整合，在重叠部分取中值，得到的原图像的加噪图像和经 *SVD* 稀疏表示去噪后的图像如图 18 所示，对整幅图训练的字典如图 19 所示

图 18 原图、加噪图和 *SVD* 去噪图

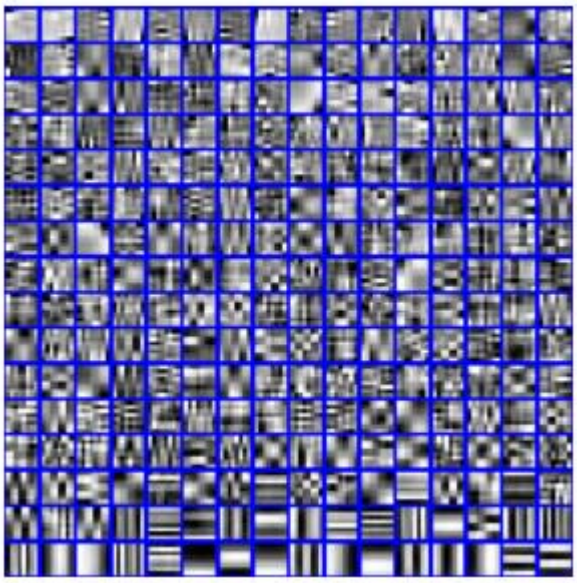

图 19 整幅图 *SVD* 字典

5.4 基于 *PSNR* 和 *SSIM* 稀疏去噪性能评价方法

对图像进行降噪处理后，需要通过图像质量评价来知道降噪效果的好坏，是否符合人眼特性。

从评价内容上分为根据图像的相似度或者理解度来评价图像质量。图像的相似度就是评价带评图像接近原始图像的程度，图像的理解度是待评图像提供给人或机器用于分析理解的图像信息的多少。

我们选用基于空间域特性的评价方法 *PSNR* 和基于结构特性的图像质量评价方法 *SSIM* 来评价去噪性能，下面介绍一下这两个方法。

5.4.1 基于空间域特性的评价方法

基于空间域特性的评价方法是基于像素同级特性的评价算法，由于计算方便，因此是评价图像质量的常用的评价标准。主要啊有两种算法--*MSE*（均方差）和 *PSNR*（峰值信噪比）。均方差衡量的是得到的图像与原始图像之间的差异，而峰值信噪比 *PSNR* 衡量的是评价图像与原始图像的相似度。下面将分别介绍这两种算法。

1. *MSE*（均方差）

均方差 *MSE* 的方法是逐点计算两个信号或图像之间绝对差，数字图像 $\left[g(i,j)\right]_{M\times N}$ 对 $\left[f(i,j)\right]_{M\times N}$ 逼真度的计算公式为

$$\varepsilon_1=\sum_{i=0}^{M-1}\sum_{j=0}^{N-1}\left[f(i,j)-g(i,j)\right]^2\Big/MN \tag{64}$$

式中 $f(i,j)$ 为原始图像，$g(i,j)$ 为对加噪图像做出的估计，图像降噪的目的就是使得两者的均方差（*MSE*）最小。均方差（*MSE*）方法没有考虑图像的像素间的相关性，它不能表达人眼对比敏感特性和人眼的掩饰特性。

2. 峰值信噪比（*PSNR*）

峰值信噪比定义为

$$PSNR=10\lg\left\{\max\left[f^2(i,j)\right]\Big/\frac{1}{MN}\sum_{i=0}^{M-1}\sum_{j=0}^{N-1}\left[f(i,j)-g(i,j)\right]^2\Big/MN\right\} \tag{65}$$

PSNR 是最常用的评价图像的客观评价法，但是，有时 *PSNR* 的评价结果与人眼看到的效果不一致，有可能 *PSNR* 较高者的视觉效果却比 *PSNR* 较低者差。这是因为人眼对于误差的敏感度会受到很多方面的影响。

5.4.2 基于结构特性的图像质量评价方法

设 $\left\{x_i\,|\,i=1,\cdots,N\right\}$ 表示原始图像像元 x 邻域中的像素，$\left\{y_i\,|\,i=1,\cdots,N\right\}$ 表示降噪处理后图像像元 y 邻域中的像素，定义邻域期望值(平均灰度)

$$\mu_x=\frac{1}{N}\sum_{i=1}^{N}x_i,\mu_y=\frac{1}{N}\sum_{i=1}^{N}y_i \tag{66}$$

定义邻域方差

$$\begin{aligned}\sigma_x^2&=\frac{1}{N-1}\sum_{i=1}^{N}\left(x_i-\mu_x\right)^2\\ \sigma_y^2&=\frac{1}{N-1}\sum_{i=1}^{N}\left(y_i-\mu_y\right)^2\\ \sigma_{xy}^2&=\frac{1}{N-1}\sum_{i=1}^{N}\left(x_i-\mu_x\right)\left(y_i-\mu_y\right)\end{aligned} \tag{67}$$

假如 x 是参考图像， y 为噪声图像，用均值 $\left(\mu_x,\mu_y\right)$ 来估计图像的亮度，用标准差 $\left(\sigma_x,\sigma_y\right)$ 来估计图像的对比度，协方差 σ_{xy} 估计图像结构相似度，则结构相似度($SSIM$)定义为

$$SSIM\left(x,y\right)=\left[1\left(x,y\right)\right]^{\alpha}\left[c\left(x,y\right)\right]^{\beta}\left[s\left(x,y\right)\right]^{\gamma} \tag{68}$$

式中亮度 1，对比度 c 、结构相似度 s 为

$$1\left(x,y\right)=\frac{2\mu_x\mu_y+C_1}{\mu_x^2+\mu_y^2+C_1},\ c\left(x,y\right)=\frac{2\sigma_x\sigma_y+C_2}{\sigma_x^2+\sigma_y^2+C_2},\ s\left(x,y\right)=\frac{2\sigma_{xy}+C_3}{\sigma_x\sigma_y+C_3}$$

常数 C_1,C_2,C_3 的作用是使分母的取值有意义，计算结果有效，然后进行估算

$$C_i=\left(K_iL\right)^2 \tag{69}$$

其中 L 为图像像素值的变化范围， $0<K_i\le 1,i=1,2,3$ 。

在式(70)中， σ_{xy} 可估计为

$$\sigma_{xy}=\frac{1}{N-1}\sum_{i=1}^{NI}\left(x_i-\mu_x\right)\left(y_i-\mu_i\right) \tag{70}$$

从几何角度看，相关系数与向量 $x-\mu_x$ 和 $y-\mu_i$ 之间的余弦角对应。

当取 $\alpha=\beta=\gamma=1,C_3=C_2/2$ 时，可以得到简化的 $SSIM$ 计算公式

$$SSIM\left(x,y\right)=\frac{\left(2\mu_x\mu_y+C_1\right)\left(2\sigma_{xy}+C_2\right)}{\left(\mu_x^2+\mu_y^2+C_1\right)\left(\sigma_x^2+\sigma_y^2+C_2\right)} \tag{71}$$

在结构相似度($SSIM$)评价方法的实际应用中，在进行图像质量评价时，考虑到图像失真的空间可变性和图像统计特征的空间非平稳性，一般采用局部的结构相似度($SSIM$)评价算法。将图像分为大小相等的无重叠或重叠的 M 个子块，之后分别计算每一块结构相似度($SSIM$)值，最后求平均得到对图像的最终评价，

即

$$MSSIM\left(x,y\right)=\frac{1}{M}\sum_{i=1}^{M}SSIM\left(x_i,y_i\right) \tag{72}$$

假设结构相似度满足以下条件：

(1)对称性： $S\left(x,y\right)=S\left(y,x\right)$

(2)有界性： $0\le S\left(x,y\right)\le 1$

(3)固定的最大值：如果 $x=y,S\left(x,y\right)=1,SSIM\left(x,y\right)$ 越大，两幅图像越相似。

5.4.3 四种方法的稀疏去噪性能评估

1.基于空间域特性的评价

分别比较四种方法求得的去噪图像与原图像，求出 *MSE*（均方差）和 *PSNR*（峰值信噪比）。用 *MATLAB* 程序(**程序见附件**)进行计算，求得第 4 行第 4 列图像小块与在四种稀疏表示去噪方法下得到的去噪小块比较所得的表如表 1。整个图像与在四种稀疏表示去噪方法下得到的去噪小块比较所得的表如表 2。

表 1 第 4 行第 4 列图像与去噪图像比较

	MSE	*PSNR*（单位： *dB*）
DCT	218.7344	24.7316
DWT	45.3865	31.5615
PCA	81.6768	299978
SVD	149．9216	26.3722

表 2 整幅图像与去噪图像比较

	MSE	*PSNR*（单位： *dB*）
DCT	84.9855	29.0026
DWT	289.3417	19.7431
PCA	91.6768	30.9978
SVD	69.0867	30.0952

2.基于结构特性的图像质量评价

分别比较四种方法求得的去噪图像与原图像的结构相似度 *SSIM*，求得第 4 行第 4 列图像小块、整个图像与在四种稀疏表示去噪方法下得到的去噪小块比较所得的表如表 3 所示。

表 3 原图像与去噪图像的结构相似度 *SSIM*

	第 4 行第 4 列图像小块	整个图像

DCT	0.7791	0.8981
DWT	0.6857	0.5922
PCA	0.5179	0.5879
SVD	0.8329	0.8914

从第 4 行第 4 列图像与去噪图像比较表中，我们观察到 *DWT* 方法的去噪效果较好，*DCT* 的去噪效果最弱。可见，在信息量很少时，*DCT* 字典构造不完全，故不能起到很好的去噪效果，而小波变换运用自适应字典，能起到好的效果。

从整幅图像与去噪图像比较表中，我们可以得到 *SVD* 方法的去噪效果最好，*DCT* 稀疏表示次之，*PCA* 和 *DWT* 去噪效果最弱。在对整幅图进行分析时，*SVD* 稀疏表示运用自适应字典进行不断迭代，尽可能的与原始图像相近，故可以起到良好的效果。而 *DCT* 稀疏变换在整幅图时可以构造比较完备的字典，可以较好的完成图像去噪的功能。

5.5 基于 $K-SVD$ 和残差比的图像去噪算法

5.5.1 新的稀疏去噪算法设计

超完备字典在图像稀疏表示中是十分重要的，它在很大程度上决定于图像的稀疏表示能否有效地进行。$K-SVD$ 算法将过完备字典的构建和优化结合起来，用待分解图像来训练原字库，能更加有效地产生反应图像各种特征的字典。

在基于稀疏表示图像去噪中，图像的有用信息具有一定的结构特征，且这些特征与原子结构相吻合，而噪声却不具有此特征，因而通过稀疏表示可将有用信息和噪声进行有效的分离，从而达到去噪的目的。对于低信噪比图像，一般将硬阈值作为稀疏表示的终止条件。当图像信噪比极低时，图像中噪声的方差相比于有用信号的方差较大，采用硬阈值作为稀疏分解迭代终止条件会产生：分解残差的阈值不易设置，重构过程中引入大量噪声派生成分，严重影响图像重构精度的问题。为得到视觉效果更好地去噪图像，我们提出了一种基于 $K-SVD$ 和残差比的稀疏表示图像去噪算法。该算法不再采用硬阈值而是采用一般残差比作为去噪算法的终止条件。

该算法流程图如图 20 所示

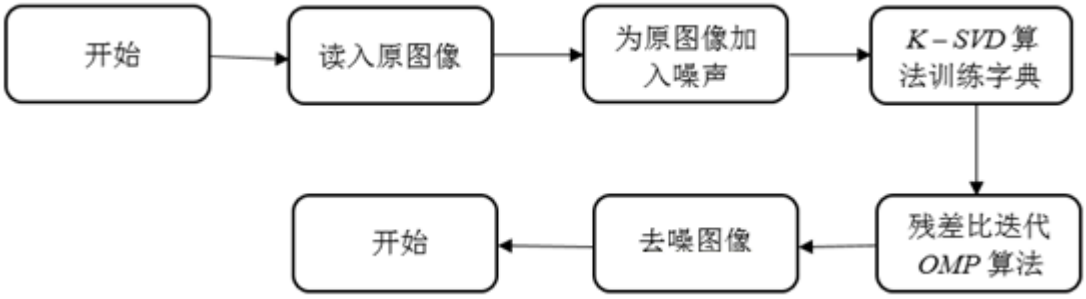

图 20 改进的算法流程图

流程图具体步骤可描述为：

(1)采用过完备 *DCT* 字典作为 $K-SVD$ 算法的初始化字典 D，该字典如图所示。用算法 *OMP* 算法对含噪图像在初始字典 D 上进行稀疏分解。在此过程中，将 ε 作为 *OMP* 算法迭代的终止条件，即当分解残余小于 ε 时，迭代终止。从而获得 $K-SVD$ 算法所需的稀疏系数矩阵。

(2)在带噪图像中按一定步长从图像矩阵的顶点至终点抽取8×8的像素块组成训练样本集。在获取训练样本集合稀疏稀疏矩阵后，用$K-SVD$算法对初始字典D进行训练，生成优化后的字典D。

(3)对带噪图像进行稀疏分解去噪。设y_i为含噪图像转化的列向量，y_i为图像的有用信息，$\Delta\omega$为y_i的频带带宽，$\hat{e}_{\Delta\omega}$为$\Delta\omega$频带内的噪声，$e_{\Delta\omega}$为$\Delta\omega$频道外的噪声。$e_{\Delta\omega}$可认为是与字典中所有原子正交的，则第k步稀疏分解的表达式可表示为

$$\begin{aligned}\left\|R^k y_i\right\|_2^2 &= \left\|R^k\left(y_i+\hat{e}_{\Delta\omega}+e_{\Delta\omega}\right)\right\|_2^2 \\ &= \left\|R^k\left(y_i+\hat{e}_{\Delta\omega}\right)\right\|_2^2+\left\|e_{\Delta\omega}\right\|_2^2\end{aligned} \tag{73}$$

由于第$k+1$步迭代出的新原子，在低信噪比下将受到一定程度的噪声影响，可能导致$\left\|R^k y_i\right\|_2^2$和$\left\|R^{k+1} y_i\right\|_2^2$的均值不同，这将影响$\left\|R^{k+1} y_i\right\|_2^2-\left\|R^{k+1} y_i\right\|_2^2$的收敛性。为增强迭代终止条件的鲁棒性，这里选用残差比作为迭代终止条件，其定义为

$$q\left(R^k y_i\right)=\frac{\left\|R^{k+1} y_i-\xi R^k y_i\right\|_i^2}{\left\|\xi R^k y_i\right\|_k^2} \tag{78}$$

其中

$$\xi=\sqrt{\frac{E\left(R^{k+1} y_i\right)^2}{E\left(R^k y_i\right)^2}} \tag{79}$$

E为取期望值。由式(73)(78)(79)，得到去噪后的图像稀疏表示系数矩阵X'。

(4)最后，有方程(80)获得去噪后的图像

$$Y'=DX' \tag{80}$$

5.5.2 算法实现与结果分析

用*MATLAB*工具对算法进行仿真模拟，将经过本算法去噪后得到的图像进行显示。得到第 4 行第 4 列小块图像的原图、加噪图和去噪图如图 21 所示。对去噪后的各小块进行重构，在重叠部分取中值，得到的整幅图像的原图、加噪图和去噪图如图 22 所示。

图 21 第 4 行第 4 列小块图像的原图、加噪图和去噪图

图 22 整幅图像的原图、加噪图和去噪图

比较本方法求得的去噪图像与原图像，求出 *MSE*（均方差）、*PSNR*（峰值信噪比）、结构相似度 *SSIM*，与上面四种进行比较如表 4 表 5 所示。

表 4 第 4 行第 4 列图像与去噪图像比较

	MSE	*PSNR*（单位：*dB*）	*SSIM*
DCT	218.7344	24.7316	0.7791
DWT	45.3865	31.5615	0.6857
PCA	81.6768	29.9978	0.5179
SVD	149.9216	26.3722	0.8329
本文方法	94.3384	28.3839	0.6475

表 5 整幅图像与去噪图像比较

	MSE	*PSNR*（单位：*dB*）	*SSIM*
DCT	84.985	29.0026	0.8981
DWT	289.3417	19.7431	0.5922
PCA	91.6768	30.9978	0.5879
SVD	69.0867	30.0952	0.8914
本文方法	80.9351	32.5915	0.7908

观察图 22 图 23，与 5.3 中所得的图比较可知，从视觉角度来说，我们提出的算法去噪效果较好，基于 *SVD* 的算法去噪效果次之，基于 *DCT* 字典的去噪效果一般。这可能是 *DCT* 字典是一种过完备字典，一旦确定就不能改变，而其他

几类算法都采用全局字典和自适应字典，收到了良好的效果。我们提出的基于*SVD*和残差比图像稀疏表示去噪算法将以上方法的优点结合，取得了好的去噪效果。

比较表 4 和表 5，从客观上对各种系数表述去噪算法进行分析，可以得到本文算法在*SSIM* 和*PSNR* 上均优于其他四种算法，达到了良好的去噪效果。

6. 模型的评价

6.1 基于$K-SVD$和残差比的图像去噪算法的优缺点

1.优点

（1）模型采用过完备*DCT* 字典作为$K-SVD$算法的初始化字典D，用算法*OMP* 算法对含噪图像在初始字典D上进行稀疏分解，并将残差比作为*OMP* 算法迭代的终止条件，可以比较正确的计算出稀疏系数矩阵。

（2）吸取了四种稀疏表示的优点，克服了存在的缺点，经过评价方法评估得出，该方法是一种良好的稀疏表示去噪方法。

2. 缺点

此模型构造方法繁琐，计算时间消耗很大，大量占用 CPU，甚至出现电脑死机的现象，还有很大的提升空间。

6.2 基于*PSNR* 和*SSIM* 的稀疏去噪性能评价方法的优缺点

1. 优点

(1)此模型从空间域特性和结构特性两个方面来评价图像去噪性能，可以较好的反应真实的去噪效果。

(2) *PSNR* 和*SSIM* 的计算原理都比较简便，一般只需要知道原图像与去噪后图像就可计算出结果，操作较方便。

2. 缺点

PSNR 和*SSIM* 方法分别从空间和结构两个方面进行评价，有时会存在矛盾的情况，同时没有考虑人类视觉感受。针对这一缺点，我们设计出优化方案。

7. 模型的优化

结构相似度*SSIM* 在图像编码、恢复、融合、水印及生物识别技术等已经有很广泛的应用，但仍旧存在很大的弊端，所以我们对*SSIM* 评估方法进行优化。而对*SSIM* 进行优化的出发点是要让降噪图像与原始图像的*SSIM* 值尽量大。图像降噪是图像处理研究者特别感兴趣的经典问题，不仅是因为它的实用价值，还因为它为图像建模、表示和估计理论提供了很好的测试平台。对*SSIM* 进行优化的出发点是要让降噪图像与原始图像的*SSIM* 值尽量大。*SSIM* 优化方案X_1、X_2为原始的清晰图像块，Y_1、Y_2是由X_1、X_2得到的噪声块，$Y_1=X_1+N_1, Y_2=X_2+N_2$，

N_1、N_2是相应的高斯噪声块，标准差为σ_n。通过Y_1、Y_2估计$S(X_1,X_2)$。它的近似值为

$$S(X_1,X_2)\approx\frac{\left(2\mu_{Y_1}\mu_{Y_2}+C_1\right)\left(2\sigma_{Y_1Y_2}+C_2\right)}{\left(\mu_{Y_1}^2+\mu_{Y_2}^2+C_1\right)\left(\sigma_{Y_1}^2+\sigma_{Y_2}^2-2\sigma_n^2+C_2\right)} \tag{81}$$

假设噪声N_1、N_2、N_2均值为0，信号X_1、X_2与噪声不相关，噪声N_1、N_2是添加在不同位置的不相关噪声。

方程(81)估计的$S(X_1,X_2)$没有达到我们期望的准确性，因为小图像块并没有被估计。同时，当噪声块比图像块明显时，*SSIM*更倾向于噪声块的值，而忽略了图像的结构。

为了改善上述问题，我们提出两阶段方法，首先，计算噪声的位置估计，假设通过$\hat{N}_1$、$\hat{N}_2$得到噪声估计，通过下式得到X_1、X_2估计

$$\begin{aligned}\hat{X}_1&=Y_1-\hat{N}_1\\ \hat{X}_2&=Y_2-\hat{N}_2\end{aligned} \tag{82}$$

之后通过$\hat{X}_1$、$\hat{X}_2$估计$S(X_1,X_2)$），定义基于*SSIM*的加权如式

$$\omega_{SSIM}=S(\hat{X}_1,\hat{X}_2) \tag{83}$$

在计算每个图像块的加权平均之前，对每个块Y进行均值和对比度的进一步调整

$$Y=\frac{\sigma_{X_c}+c}{\sigma_Y+c}\left(Y-\mu_Y\right)+\mu_{X_c} \tag{84}$$

其中μ_Y,σ_Y是当前块的均值，μ_{X_c},σ_{X_c}是当前块的对比度值，通过方程（84）对块进行降噪（估计），c是常数。这种调整的出发点是受*SSIM*的启发，独立于均值，对比度，结构的测量。实际上，基于*SSIM*加权计算可能对收集这些块有帮助，它与降噪块的结构相似，但是与对比度和均值不同，为了避免均值和对比度的偏差，首先对这些块进行规范化，这样只对块的结构部分进行降噪。最后我们通过下式在i位置建立最终的降噪块

$$X_{(i)}=\frac{\sum_{j\in N_i}\omega_{SSIM}(i,j)Y'(j)}{\sum_{j\in N_i}\omega_{SSIM}(i,j)} \tag{85}$$

其中N_i表示i周围邻域的结合，ω_{SSIM}是块在位置i和j的*SSIM*加权计算。

8. 参考文献

[1]王瑶. 基于稀疏表示的图像去噪算法研究[D]. 安徽大学. 2014.

[2]乔雅莉. 基于稀疏表示的图像去噪算法研究[D]. 中国交通大学. 2009.
[3]胡光书. 数字信号处理. 清华大学出版社[M]. 2003
[4]李俊秀, 姜三平. 基于主成分分析的图像自适应阈值去噪算法[J]. 红外技术. 2014.
[5]肖泉, 丁兴号, 王守觉, 郭东辉, 廖英豪. 基于自适应超完备稀疏表示的图像去噪方法[J]. 2009

附录

代码：

1. *DCT*

(1)主函数

```
clear
bb=8; % block size
RR=4; % redundancy factor
K=RR*bb^2; % number of atoms in the dictionary
sigma =10;
pathForImages ='';
imageName = '4行4列.bmp';
[IMin0,pp]=imread(strcat([pathForImages,imageName]));
IMin0=im2double(IMin0);
if (length(size(IMin0))>2)
    IMin0 = rgb2gray(IMin0);
end
if (max(IMin0(:))<2)
    IMin0 = IMin0*255;
end
  IMin=IMin0+sigma*randn(size(IMin0));
PSNRIn = 20*log10(255/sqrt(mean((IMin(:)-IMin0(:)).^2)));
% DCT
[IoutDCT,output] = denoiseImageDCT(IMin, sigma, K);
PSNROut = 20*log10(255/sqrt(mean((IoutDCT(:)-IMin0(:)).^2)));
figure;
imshow(IoutDCT,[]);
title(strcat(['Image by DCT dictionary, ',num2str(PSNROut),'dB']));
A=im2uint8(mat2gray(IoutDCT));
imwrite(A,'4行4列DCT.bmp','bmp');
subplot(1,3,1); imshow(IMin0,[]); title('Original image');
subplot(1,3,2); imshow(IMin,[]); title(strcat(['Noisy image,
',num2str(PSNRIn),'dB']));
subplot(1,3,3); imshow(IoutDCT,[]); title(strcat(['Image by DCT dictionary,
',num2str(PSNROut),'dB']));
figure;
```

```
I = displayDictionaryElementsAsImage(output.D, floor(sqrt(K)),
floor(size(output.D,2)/floor(sqrt(K))),bb,bb,0);
title('The DCT dictionary');
```

(2) denoiseImage *DCT* 函数

```
function [IOut,output] = denoiseImageDCT(Image,sigma,K,varargin)
Reduce_DC = 1;
[NN1,NN2] = size(Image);
C = 1.15;
waitBarOn = 1;
maxBlocksToConsider = 260000;
slidingDis = 1;
bb = 8;
for argI = 1:2:length(varargin)
    if (strcmp(varargin{argI}, 'slidingFactor'))
        slidingDis = varargin{argI+1};
    end
    if (strcmp(varargin{argI}, 'errorFactor'))
        C = varargin{argI+1};
    end
    if (strcmp(varargin{argI}, 'maxBlocksToConsider'))
        maxBlocksToConsider = varargin{argI+1};
    end
    if (strcmp(varargin{argI}, 'blockSize'))
        bb = varargin{argI+1};
    end
    if (strcmp(varargin{argI}, 'waitBarOn'))
        waitBarOn = varargin{argI+1};
    end
end
errT = C*sigma;
% Create an initial dictionary from the DCT frame
Pn=ceil(sqrt(K));
DCT=zeros(bb,Pn);
for k=0:1:Pn-1,
    V=cos([0:1:bb-1]'*k*pi/Pn);
    if k>0, V=V-mean(V); end;
    DCT(:,k+1)=V/norm(V);
end;
DCT=kron(DCT,DCT);
while (prod(floor((size(Image)-bb)/slidingDis)+1)>maxBlocksToConsider)
    slidingDis = slidingDis+1;
end
[blocks,idx] = my_im2col(Image,[bb,bb],slidingDis);
```

```
if (waitBarOn)
    counterForWaitBar = size(blocks,2);
    h = waitbar(0,'Denoising In Process ...');
end
% go with jumps of 10000
for jj = 1:10000:size(blocks,2)
    if (waitBarOn)
        waitbar(jj/counterForWaitBar);
    end
    jumpSize = min(jj+10000-1,size(blocks,2));
    if (Reduce_DC)
        vecOfMeans = mean(blocks(:,jj:jumpSize));
        blocks(:,jj:jumpSize) = blocks(:,jj:jumpSize) -
repmat(vecOfMeans,size(blocks,1),1);
    end
    %Coefs = mexOMPerrIterative(blocks(:,jj:jumpSize),DCT,errT);
    Coefs = OMPerr(DCT,blocks(:,jj:jumpSize),errT);
    if (Reduce_DC)
        blocks(:,jj:jumpSize)= DCT*Coefs + ones(size(blocks,1),1) * vecOfMeans;
    else
        blocks(:,jj:jumpSize)= DCT*Coefs ;
    end
end
count = 1;
Weight= zeros(NN1,NN2);
IMout = zeros(NN1,NN2);
[rows,cols] = ind2sub(size(Image)-bb+1,idx);
for i    = 1:length(cols)
    col = cols(i); row = rows(i);
    block =reshape(blocks(:,count),[bb,bb]);
    IMout(row:row+bb-1,col:col+bb-1)=IMout(row:row+bb-1,col:col+bb-1)+block;

Weight(row:row+bb-1,col:col+bb-1)=Weight(row:row+bb-1,col:col+bb-1)+ones(bb);
    count = count+1;
end;
if (waitBarOn)
    close(h);
end
IOut = (Image+0.034*sigma*IMout)./(1+0.034*sigma*Weight);
output.D = DCT;
```

2. *DB*4

```
clear all
close all
```

```
clc
im=imread('4行4列.bmp');
im=double(im)/256;
sigma=10/256;
%imn=im+sigma*randn(size(im));
imn=imnoise(im,'gaussian',0,(10/256)^2);
[m,n]=size(imn);
%alfa=3*sigma;
alfa=sigma*sqrt(2*log(m*n));
[ca1,ch1,cv1,cd1]=dwt2(imn,'db4');
[ca2,ch2,cv2,cd2]=dwt2(ca1,'db4');
y={ca2,ch2,cv2,cd2,ch1,cv1,cd1};
[c,s]=lp2vec(y);
cr = c .* (abs(c) >= alfa);
k=prod(s(1,:)+1);
c1=c(k:end);
c2=c(1:k-1);
cr=((abs(c1)-alfa).*((c1 > 0)+(-(c1 < 0)))).*(abs(c1) >= alfa);
cr=[c2',cr'];
k=prod(s(1,:)+1);
for i=k:length(c)
            if abs(c(i))<alfa
                  cr(i)=0;
         else
           cr(i)=c(i);
            end
    end
r = vec2lp(cr, s);
w2 = idwt2(r{1},r{2},r{3},r{4}, 'db4');
%    w3=idwt2(w2,r{5},r{6},r{7},'db4');
imwrite(im,'original.bmp','bmp');
imwrite(imn,'noisey.bmp','bmp');
imwrite(w2,'denoise.bmp','bmp')
figure,subplot(131),imshow(im);
title('Original image');
subplot(132),imshow(imn);
title('Noisy image');
subplot(133),imshow(w2);
title('Image by DB4');
```

3. *PCA*

```
clear
bb=8; % block size
RR=4; % redundancy factor
```

```
K=RR*bb^2; % number of atoms in the dictionary
sigma =10;
pathForImages ='';
imageName = '4行4列.bmp';
[IMin0,pp]=imread(strcat([pathForImages,imageName]));
IMin0=im2double(IMin0);
if (length(size(IMin0))>2)
    IMin0 = rgb2gray(IMin0);
end
if (max(IMin0(:))<2)
    IMin0 = IMin0*255;
end
  IMin=IMin0+sigma*randn(size(IMin0));
PSNRIn = 20*log10(255/sqrt(mean((IMin(:)-IMin0(:)).^2)));
[IoutPca,output] = denoiseImagePca(IMin, sigma,K);
PSNROut = 20*log10(255/sqrt(mean((IoutPca(:)-IMin0(:)).^2)));
figure;
subplot(1,3,1); imshow(IMin0,[]); title('Original image');
subplot(1,3,2); imshow(IMin,[]); title(strcat(['Noisy image,
',num2str(PSNRIn),'dB']));
subplot(1,3,3); imshow(IoutPca,[]); title(strcat(['Image by Pca,
',num2str(PSNROut),'dB']));
A3=im2uint8(mat2gray(IoutPca));
imwrite(A3,'pca.bmp','bmp')
figure;
I = displayDictionaryElementsAsImage(output.D, floor(sqrt(K)),
floor(size(output.D,2)/floor(sqrt(K))),bb,bb);
title('The dictionary trained on Pca');
```

（2）denoiseImagePca.m

```
function [IOut,output] = denoiseImagePca(Image, sigma, varargin)
fildNameForGlobalDictionary = 'globalTrainedDictionary';
Reduce_DC = 1;
[NN1,NN2] = size(Image);
waitBarOn = 1;
C = 1.15;
maxBlocksToConsider = 260000;
slidingDis = 1;
bb = 8;
givenDictionaryFlag = 0;
for argI = 1:2:length(varargin)
    if (strcmp(varargin{argI}, 'slidingFactor'))
        slidingDis = varargin{argI+1};
    end
```

```
        if (strcmp(varargin{argI}, 'errorFactor'))
            C = varargin{argI+1};
        end
        if (strcmp(varargin{argI}, 'maxBlocksToConsider'))
            maxBlocksToConsider = varargin{argI+1};
        end
        if (strcmp(varargin{argI}, 'givenDictionary'))
            D = varargin{argI+1};
            givenDictionaryFlag = 1;
        end
        if (strcmp(varargin{argI}, 'blockSize'))
            bb = varargin{argI+1};
        end
        if (strcmp(varargin{argI}, 'waitBarOn'))
            waitBarOn = varargin{argI+1};
        end
end
if (~givenDictionaryFlag)
        eval(['load ',fildNameForGlobalDictionary]);
        D = currDictionary;
end
errT = C*sigma;
%blocks = im2col(Image,[NN1,NN2],[bb,bb],'sliding');
while (prod(size(Image)-bb+1)>maxBlocksToConsider)
        slidingDis = slidingDis+1;
end
[blocks,idx] = my_im2col(Image,[bb,bb],slidingDis);
if (waitBarOn)
        counterForWaitBar = size(blocks,2);
        h = waitbar(0,'Denoising In Process ...');
end
% go with jumps of 10000
for jj = 1:10000:size(blocks,2)
        if (waitBarOn)
            waitbar(jj/counterForWaitBar);
        end
        jumpSize = min(jj+10000-1,size(blocks,2));
        if (Reduce_DC)
            vecOfMeans = mean(blocks(:,jj:jumpSize));
            blocks(:,jj:jumpSize) = blocks(:,jj:jumpSize) -
repmat(vecOfMeans,size(blocks,1),1);
        end
        %Coefs = mexOMPerrIterative(blocks(:,jj:jumpSize),D,errT);
        Coefs = OMPerr(D,blocks(:,jj:jumpSize),errT);
```

```
        if (Reduce_DC)
            %block=reshape(D*a+mm+MM,[bb,bb]);
            blocks(:,jj:jumpSize)= D*Coefs + ones(size(blocks,1),1) * vecOfMeans;
        else
            blocks(:,jj:jumpSize)= D*Coefs ;
        end
end
count = 1;
Weight=zeros(NN1,NN2);
IMout = zeros(NN1,NN2);
[rows,cols] = ind2sub(size(Image)-bb+1,idx);
for i    = 1:length(cols)
        col = cols(i); row = rows(i);
        block =reshape(blocks(:,count),[bb,bb]);
        IMout(row:row+bb-1,col:col+bb-1)=IMout(row:row+bb-1,col:col+bb-1)+block;
Weight(row:row+bb-1,col:col+bb-1)=Weight(row:row+bb-1,col:col+bb-1)+ones(bb);
        count = count+1;
end;

if (waitBarOn)
        close(h);
end
output.D = D;
IOut = (Image+0.034*sigma*IMout)./(1+0.034*sigma*Weight);
```

4. *SVD*

```
clear
bb=8; % block size
RR=4; % redundancy factor
K=RR*bb^2; % number of atoms in the dictionary
sigma =10;
pathForImages ='';
imageName = '4行4列.bmp';
[IMin0,pp]=imread(strcat([pathForImages,imageName]));
IMin0=im2double(IMin0);
if (length(size(IMin0))>2)
        IMin0 = rgb2gray(IMin0);
end
if (max(IMin0(:))<2)
        IMin0 = IMin0*255;
end
   IMin=IMin0+sigma*randn(size(IMin0));
PSNRIn = 20*log10(255/sqrt(mean((IMin(:)-IMin0(:)).^2)));
% KSVD
```

```
[IoutAdaptive,output] = denoiseImageKSVD(IMin, sigma,K);
PSNROut = 20*log10(255/sqrt(mean((IoutAdaptive(:)-IMin0(:)).^2)));
figure;
imshow(IoutAdaptive,[]); title(strcat(['Image by Adaptive dictionary,
',num2str(PSNROut),'dB']));
  B=im2uint8(mat2gray(IoutAdaptive));
imwrite(B,'4ĐĐ4ÁĐSVD.bmp','bmp');
figure;
subplot(1,3,1); imshow(IMin0,[]); title('Original image');
subplot(1,3,2); imshow(IMin,[]); title(strcat(['Noisy image,
',num2str(PSNRIn),'dB']));
subplot(1,3,3); imshow(IoutAdaptive,[]); title(strcat(['Image by Adaptive dictionary,
',num2str(PSNROut),'dB']));
figure;
I = displayDictionaryElementsAsImage(output.D, floor(sqrt(K)),
floor(size(output.D,2)/floor(sqrt(K))),bb,bb);
title('The dictionary trained on patches from the noisy image');
```

（2）denoiseImage $K-SVD$.m

```
function [IOut,output] = denoiseImageKSVD(Image,sigma,K,varargin)
% first, train a dictionary on the noisy image
reduceDC = 1;
[NN1,NN2] = size(Image);
waitBarOn = 1;
if (sigma > 5)
    numIterOfKsvd = 10;
else
    numIterOfKsvd = 5;
end
C = 1.15;
maxBlocksToConsider = 260000;
slidingDis = 1;
bb = 8;
maxNumBlocksToTrainOn = 65000;
displayFlag = 1;
for argI = 1:2:length(varargin)
    if (strcmp(varargin{argI}, 'slidingFactor'))
        slidingDis = varargin{argI+1};
    end
    if (strcmp(varargin{argI}, 'errorFactor'))
        C = varargin{argI+1};
    end
    if (strcmp(varargin{argI}, 'maxBlocksToConsider'))
        maxBlocksToConsider = varargin{argI+1};
    end
```

```
        if (strcmp(varargin{argI}, 'numKSVDIters'))
            numIterOfKsvd = varargin{argI+1};
        end
        if (strcmp(varargin{argI}, 'blockSize'))
            bb = varargin{argI+1};
        end
        if (strcmp(varargin{argI}, 'maxNumBlocksToTrainOn'))
            maxNumBlocksToTrainOn = varargin{argI+1};
        end
        if (strcmp(varargin{argI}, 'displayFlag'))
            displayFlag = varargin{argI+1};
        end
        if (strcmp(varargin{argI}, 'waitBarOn'))
            waitBarOn = varargin{argI+1};
        end
end
if (sigma <= 5)
        numIterOfKsvd = 5;
end
% first, train a dictionary on blocks from the noisy image
if(prod([NN1,NN2]-bb+1)> maxNumBlocksToTrainOn)
        randPermutation =    randperm(prod([NN1,NN2]-bb+1));
        selectedBlocks = randPermutation(1:maxNumBlocksToTrainOn);
        blkMatrix = zeros(bb^2,maxNumBlocksToTrainOn);
        for i = 1:maxNumBlocksToTrainOn
            [row,col] = ind2sub(size(Image)-bb+1,selectedBlocks(i));
            currBlock = Image(row:row+bb-1,col:col+bb-1);
            blkMatrix(:,i) = currBlock(:);
        end
else
        blkMatrix = im2col(Image,[bb,bb],'sliding');
end
param.K = K;
param.numIteration = numIterOfKsvd ;
param.errorFlag = 1; % decompose signals until a certain error is reached. do not use
fix number of coefficients.
param.errorGoal = sigma*C;
param.preserveDCAtom = 0;
Pn=ceil(sqrt(K));
DCT=zeros(bb,Pn);
for k=0:1:Pn-1,
        V=cos([0:1:bb-1]'*k*pi/Pn);
        if k>0, V=V-mean(V); end;
        DCT(:,k+1)=V/norm(V);
```

```
end;
DCT=kron(DCT,DCT);
param.initialDictionary = DCT(:,1:param.K );
param.InitializationMethod =    'GivenMatrix';
if (reduceDC)
    vecOfMeans = mean(blkMatrix);
    blkMatrix = blkMatrix-ones(size(blkMatrix,1),1)*vecOfMeans;
end

if (waitBarOn)
    counterForWaitBar = param.numIteration+1;
    h = waitbar(0,'Denoising In Process ...');
    param.waitBarHandle = h;
    param.counterForWaitBar = counterForWaitBar;
end
param.displayProgress = displayFlag;
[Dictionary,output] = KSVD(blkMatrix,param);
output.D = Dictionary;
if (displayFlag)
    disp('finished Trainning dictionary');
end
% denoise the image using the resulted dictionary
errT = sigma*C;
IMout=zeros(NN1,NN2);
Weight=zeros(NN1,NN2);
%blocks = im2col(Image,[NN1,NN2],[bb,bb],'sliding');
while (prod(floor((size(Image)-bb)/slidingDis)+1)>maxBlocksToConsider)
    slidingDis = slidingDis+1;
end
[blocks,idx] = my_im2col(Image,[bb,bb],slidingDis);

if (waitBarOn)
    newCounterForWaitBar = (param.numIteration+1)*size(blocks,2);
end
% go with jumps of 30000
for jj = 1:30000:size(blocks,2)
    if (waitBarOn)
        waitbar(((param.numIteration*size(blocks,2))+jj)/newCounterForWaitBar);
    end
    jumpSize = min(jj+30000-1,size(blocks,2));
    if (reduceDC)
        vecOfMeans = mean(blocks(:,jj:jumpSize));
        blocks(:,jj:jumpSize) = blocks(:,jj:jumpSize) -
repmat(vecOfMeans,size(blocks,1),1);
```

```
        end

        %Coefs = mexOMPerrIterative(blocks(:,jj:jumpSize),Dictionary,errT);
        Coefs = OMPerr(Dictionary,blocks(:,jj:jumpSize),errT);
        if (reduceDC)
            blocks(:,jj:jumpSize)= Dictionary*Coefs + ones(size(blocks,1),1) *
vecOfMeans;
        else
            blocks(:,jj:jumpSize)= Dictionary*Coefs ;
        end
end
count = 1;
Weight = zeros(NN1,NN2);
IMout = zeros(NN1,NN2);
[rows,cols] = ind2sub(size(Image)-bb+1,idx);
for i    = 1:length(cols)
        col = cols(i); row = rows(i);
        block =reshape(blocks(:,count),[bb,bb]);
        IMout(row:row+bb-1,col:col+bb-1)=IMout(row:row+bb-1,col:col+bb-1)+block;
Weight(row:row+bb-1,col:col+bb-1)=Weight(row:row+bb-1,col:col+bb-1)+ones(bb);
        count = count+1;
end;

if (waitBarOn)
        close(h);
end
IOut = (Image+0.034*sigma*IMout)./(1+0.034*sigma*Weight);
```

5.基于 *SSIM* 的优化

```
clear
bb=8; % block size
RR=4; % redundancy factor
K=RR*bb^2; % number of atoms in the dictionary
C=2.5;
%****************** pathForImages ='';
imageName = 'original.bmp';
[IMin0,pp]=imread(strcat([pathForImages,imageName]));
IMin=imread('noisey.bmp');
I=IMin;
IMin0=im2double(IMin0);
IMin=im2double(IMin);
if (length(size(IMin0))>2)
        IMin0 = rgb2gray(IMin0);
end
```

```
if (max(IMin0(:))<2)
        IMin0 = IMin0*255;
end
if (length(size(IMin))>2)
        IMin = rgb2gray(IMin);
end
if (max(IMin(:))<2)
        IMin = IMin*255;
end
input=IMin;
AI=0;
L=2;
IMin = pointEnhance(input,AI,L)
  figure;
  imshow(IMin,[]);
  PSNRIn = 20*log10(255/sqrt(mean((IMin(:)-IMin0(:)).^2)));
%******************  *************************
% L=4;
usum=0;
sum=0;
for m=1:L-1
           usum=usum+1/m;
          sum=sum+1/(m^2);
end
    su=0;
  for i=1:L-1
         su=su+1/i;
  end
  u=-0.577215-log(L)+su;
sigma=sqrt(pi*pi/6-sum);
%*****************************************
  if AI==1
         u=u;
         sigma=sigma;
  else if AI==0
                u=1/2*u;
                sigma=1/4*sigma;
         end
  end
  %*****************************************
  IMin=log(IMin+1);
  %      IMin=log(IMin+0.01);
  lamda=1/sigma;
[IoutDCT,output] = denoiseImageDCT(IMin, sigma, K);
```

```
PSNROut = 20*log10(255/sqrt(mean((IoutDCT(:)-IMin0(:)).^2)));
figure;
subplot(1,3,1); imshow(IMin0,[]); title('Original clean image');
subplot(1,3,2); imshow(IMin,[]); title(strcat(['Noisy image,
',num2str(PSNRIn),'dB']));
subplot(1,3,3); imshow(IoutDCT,[]); title(strcat(['Clean Image by DCT dictionary,
',num2str(PSNROut),'dB']));
figure;
I = displayDictionaryElementsAsImage(output.D, floor(sqrt(K)),
floor(size(output.D,2)/floor(sqrt(K))),bb,bb,0);
title('The DCT dictionary');
% Pca
[IoutGlobal,output] = denoiseImageGlobal(IMin, sigma,K);
PSNROut = 20*log10(255/sqrt(mean((IoutGlobal(:)-IMin0(:)).^2)));
figure;
subplot(1,3,1); imshow(IMin0,[]); title('Original clean image');
subplot(1,3,2); imshow(IMin,[]); title(strcat(['Noisy image,
',num2str(PSNRIn),'dB']));
subplot(1,3,3); imshow(IoutGlobal,[]); title(strcat(['Clean Image by Global Trained
dictionary, ',num2str(PSNROut),'dB']));
figure;
I = displayDictionaryElementsAsImage(output.D, floor(sqrt(K)),
floor(size(output.D,2)/floor(sqrt(K))),bb,bb);
title('The dictionary trained on patches from natural images');
% SVD
[Iout,output] = denoiseImageKSVD(IMin,sigma,K,C,lamda);
IoutAdaptive=exp(Iout);
R1=R(I);
[a,b]=size(IoutAdaptive);
for i=1:a
      for j=1:b
            if R1(i,j)>0
                  IoutAdaptive(i,j)=R1(i,j);
            end
      end
end
PSNROut = 20*log10(255/sqrt(mean((IoutAdaptive(:)-IMin0(:)).^2)));
figure;
subplot(1,3,1); imshow(IMin0,[]); title('clean image');
subplot(1,3,2); imshow(I,[]); title(strcat(['Noisy , ',num2str(PSNRIn),'dB']));
subplot(1,3,3); imshow(IoutAdaptive,[]); title(strcat(['KSVD,
',num2str(PSNROut),'dB']));
 ssim=SSIM(IMin0, IoutAdaptive)
 mssim= MSSIM(IMin0, IoutAdaptive)
```

```
AA=im2uint8(mat2gray(IoutAdaptive));
imwrite(AA,'nn.bmp','bmp');
 figure;
subplot(1,3,1); imshow(IMin0,[]); title('clean image');
subplot(1,3,2); imshow(I,[]); title('Noisy');
subplot(1,3,3); imshow(IoutAdaptive,[]); title('KSVD');
```

6. *MSE*、*PSNR*代码

```
close all;
clear all;
clc;
% img=imread('xoriginal.bmp');
img=imread('original.bmp');
% imgn=imread('xDCT.bmp');
% imgn=imread('xdb4denoise.bmp');
% imgn=imread('xSVD.bmp');
% imgn=imread('xnn.bmp');
imgn=imread('pca.bmp');

% imgn=imread('DCT.bmp');
% imgn=imread('db4denoise.bmp');
% imgn=imread('SVD.bmp');
% imgn=imread('nn.bmp');
[h w]=size(img);
img=double(img);
imgn=double(imgn);
B=8;                        %编码一个像素用多少二进制位
MAX=2^B-1;                  %图像有多少灰度级
MES=sum(sum((img-imgn).^2))/(h*w)       %均方差
PSNR=20*log10(MAX/sqrt(MES))                %峰值信噪比
```

7. *SSIM*

（1）主函数

```
% img=imread('xoriginal.bmp');
img=imread('original.bmp');
% imgn=imread('xDCT.bmp');
% imgn=imread('xdb4denoise.bmp');
% imgn=imread('xSVD.bmp');
% imgn=imread('xnn.bmp');
% imgn=imread('DCT.bmp');
% imgn=imread('db4denoise.bmp');
% imgn=imread('SVD.bmp');
imgn=imread('nn.bmp');
% K=[0.01 0.03];
% L = 255;
```

```
[mssim ssim_map] = ssim_index(img, imgn);
```

（2） *SSIM* 函数

```
function [mssim, ssim_map] = ssim_index(img1, img2, K, window, L)

if (nargin < 2 | nargin > 5)
    ssim_index = -Inf;
    ssim_map = -Inf;
    return;
end
if (size(img1) ~= size(img2))
    ssim_index = -Inf;
    ssim_map = -Inf;
    return;
end
[M N] = size(img1);
if (nargin == 2)
    if ((M < 11) | (N < 11))
        ssim_index = -Inf;
        ssim_map = -Inf;
        return
    end
    window = fspecial('gaussian', 11, 1.5);
    K(1) = 0.01;
    K(2) = 0.03;
    L = 255;
end
if (nargin == 3)
    if ((M < 11) | (N < 11))
        ssim_index = -Inf;
        ssim_map = -Inf;
        return
    end
    window = fspecial('gaussian', 11, 1.5);
    L = 255;
    if (length(K) == 2)
        if (K(1) < 0 | K(2) < 0)
            ssim_index = -Inf;
            ssim_map = -Inf;
            return;
        end
    else
        ssim_index = -Inf;
        ssim_map = -Inf;
        return;
```

```
    end
end
if (nargin == 4)
    [H W] = size(window);
    if ((H*W) < 4 | (H > M) | (W > N))
        ssim_index = -Inf;
        ssim_map = -Inf;
        return
    end
    L = 255;
    if (length(K) == 2)
        if (K(1) < 0 | K(2) < 0)
            ssim_index = -Inf;
            ssim_map = -Inf;
            return;
        end
    else
        ssim_index = -Inf;
      ssim_map = -Inf;
        return;
    end
end
if (nargin == 5)
    [H W] = size(window);
    if ((H*W) < 4 | (H > M) | (W > N))
        ssim_index = -Inf;
        ssim_map = -Inf;
        return
    end
    if (length(K) == 2)
        if (K(1) < 0 | K(2) < 0)
            ssim_index = -Inf;
            ssim_map = -Inf;
            return;
        end
    else
        ssim_index = -Inf;
      ssim_map = -Inf;
        return;
    end
end
C1 = (K(1)*L)^2;
C2 = (K(2)*L)^2;
window = window/sum(sum(window));
```

```
img1 = double(img1);
img2 = double(img2);
mu1      = filter2(window, img1, 'valid');
mu2      = filter2(window, img2, 'valid');
mu1_sq = mu1.*mu1;
mu2_sq = mu2.*mu2;
mu1_mu2 = mu1.*mu2;
sigma1_sq = filter2(window, img1.*img1, 'valid') - mu1_sq;
sigma2_sq = filter2(window, img2.*img2, 'valid') - mu2_sq;
sigma12 = filter2(window, img1.*img2, 'valid') - mu1_mu2;
if (C1 > 0 & C2 > 0)
    ssim_map = ((2*mu1_mu2 + C1).*(2*sigma12 + C2))./((mu1_sq + mu2_sq +
C1).*(sigma1_sq + sigma2_sq + C2));
else
    numerator1 = 2*mu1_mu2 + C1;
    numerator2 = 2*sigma12 + C2;
     denominator1 = mu1_sq + mu2_sq + C1;
    denominator2 = sigma1_sq + sigma2_sq + C2;
    ssim_map = ones(size(mu1));
    index = (denominator1.*denominator2 > 0);
    ssim_map(index) =
(numerator1(index).*numerator2(index))./(denominator1(index).*denominator2(index
));
    index = (denominator1 ~= 0) & (denominator2 == 0);
    ssim_map(index) = numerator1(index)./denominator1(index);
end
mssim = mean2(ssim_map)
return
```