基于最小化联合l2,1范数的非凸秩估计的红外小目标检测

IPI模型

IPI模型:

$$D = B + T + N. \tag{4}$$

原图像D由背景B、目标T、噪声N组成。

IPI模型基于两个假设，**背景图像是一个低秩矩阵，目标图像是一个稀疏矩阵。**论文提到该假设较为符合物理实际，并且现在有很多高效的低秩矩阵恢复的方法，所以这个模型效率和泛用性极高。

该模型通过滑动窗将原图像进行提取，将得到的每个面片（Patch）拉伸成一维列向量，n个列向量组合成新的矩阵，即公式中的D。

T是一个稀疏矩阵，即

$$\|T\|_0 < k, \tag{5}$$

T的非零元个数小于k，k远小于T矩阵的元素个数

B是一个低秩矩阵，即

$$\text{rank}(B) \leq r, \tag{6}$$

r是一个常数，对于越高复杂度的背景，r越高。实验中，背景图像的奇异值总是迅速收敛到0，印证了该假设的正确性。

由于一张图像较远像素也往往有较高相关性，提取出的D通常可以使用现有的许多非局部自相似性的方法。

N假设为一个i.i.d(独立同分布白噪声)

In this paper, we just assume that the random noise is i.i.d. and $\|N\|_F \leq \delta$ for some $\delta > 0$. Thus we have:

$$\|D - B - T\|_F \leq \delta, \tag{7}$$

where $\|\cdot\|_F$ is the Frobenius norm (i.e. $\|X\|_F = \sqrt{\sum_{ij} X_{ij}^2}$).

在该模型中的k,r,δ,对不同图像不同，但好消息是我们不需要直接计算出这些值。

通过该模型小目标检测实际上是从数据矩阵中恢复低秩分量和稀疏分量的问题

即：

$$\min_{B,T} \|B\|_* + \lambda\|T\|_1 \quad \text{s.t.} \quad D = B + T, \tag{8}$$

可转换为对应问题

$$\min_{B,T} \|B\|_* + \lambda\|T\|_1 + \frac{1}{2\mu}\|D - B - T\|_F^2, \tag{10}$$

因为该问题是一个凸问题，可以使用 `Accelerated Proximal Gradient (APG)`求解

Algorithm 1 Solution via Accelerated Proximal Gradient

Input: Infrared patch-image matrix $D \in R^{m\times n}$, λ.

$B_0 = B_{-1} = 0; T_0 = T_{-1} = 0; a_0 = a_{-1} = 1; \mu_0 > 0; \bar{\mu} > 0; \eta < 1.$

while not converged **do**

$Y_k^B = B_k + \frac{a_{k-1}-1}{a_k}(B_k - B_{k-1}), Y_k^T = T_k + \frac{a_{k-1}-1}{a_k}(T_k - T_{k-1}).$

$G_k^B = Y_k^B - \frac{1}{2}(Y_k^B + Y_k^T - D).$

$(U, S, V) = \text{svd}(G_k^B), B_{k+1} = U\mathcal{S}_{\frac{\mu_k}{2}}[S]V^t.$

$G_k^T = Y_k^T - \frac{1}{2}(Y_k^B + Y_k^T - D).$

$T_{k+1} = \mathcal{S}_{\frac{\lambda\mu_k}{2}}[G_k^T].$

$a_{k+1} = \frac{1+\sqrt{4a_k^2+1}}{2}; \mu_{k+1} = \max(\eta\mu_k, \bar{\mu}).$

$k = k + 1.$

end while

Output: $B = B_k, T = T_k.$

其中

$$\mathcal{S}_\varepsilon[x] = \begin{cases} x - \varepsilon, & \text{if } x > \varepsilon, \\ x + \varepsilon, & \text{if } x < -\varepsilon, \\ 0, & \text{otherwise}, \end{cases} \tag{11}$$

该模型完整求解过程

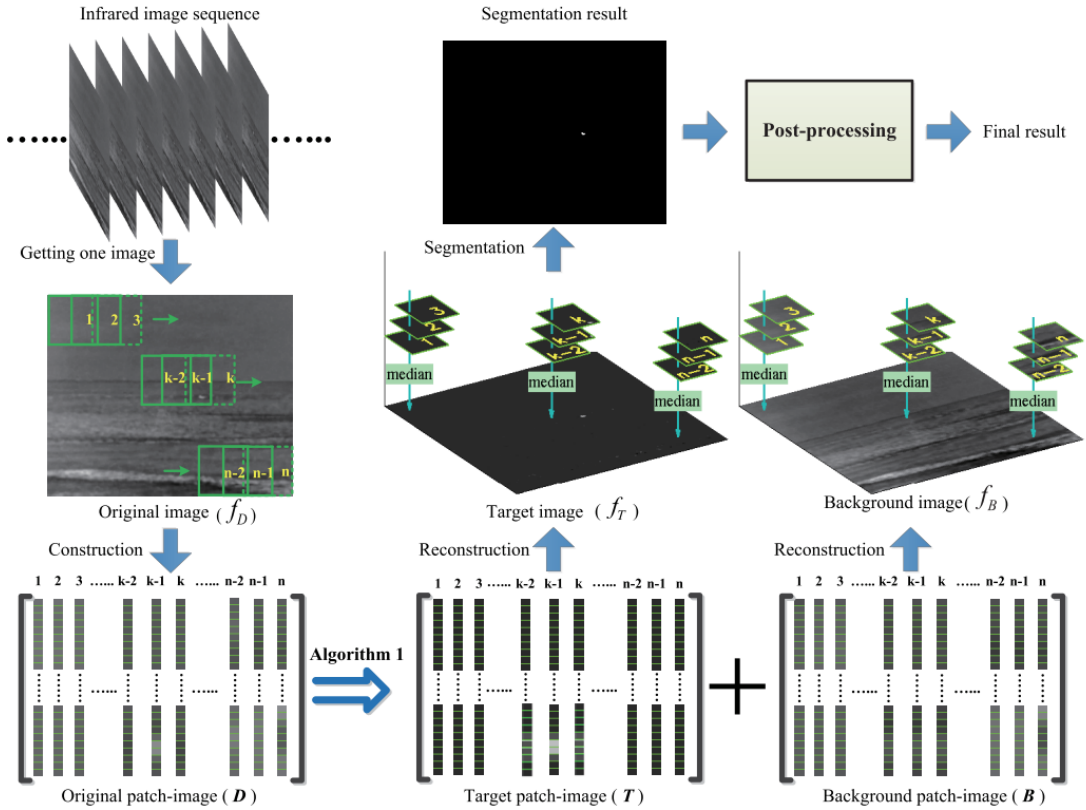

Fig. 5. The overview of the proposed method in this paper.

首先，根据从图像序列获得的原始红外图像fD构建补丁图像D。
其次，将算法1应用于斑块图像D以同时估计低秩背景斑块图像B和稀疏目标斑块图像T。
第三，我们分别从补丁图像B和T重建背景图像fB和目标图像fT。
第四，我们使用一种简单的分割方法来自适应地分割目标图像fT，因为它包含一些小值的误差。最后，通过后处理，对分割结果进行细化，得到最终的检测结果。

在算法1中，选择$\lambda = 1/\sqrt{max(m,n)}$，$\eta = 0.99$，$\mu_0 = s_2$，$\bar{\mu} = 0.05s_4$，$s_2, s_4$是D的第二和第四奇异值。

第三步中重叠部分的像素使用中值滤波器，比均值滤波器鲁棒性更好。

第四步中设置阈值确定目标：

$$t_{up} = \max(v_{min}, \mu + k\sigma), \tag{12}$$

可按需要选择双边阈值：

$$t_{up} = \max(v_{min}, \mu + k\sigma), \tag{12}$$

$$t_{down} = \min(v_{max}, \mu - k\sigma), \tag{13}$$

v_{max}、v_{min}、k为经验确定的常数，μ、σ为f_T的均值和标准差

最后一步的后处理中可使用区域分析方法去删除错误检测目标，可使用形态学方法去提炼目标区域。并且使用统计技术估计目标在重建背景图像中的对应局部区域的复杂度，然后利用估计结果评估器可靠性。

参考链接：Infrared Patch-Image Model for Small Target Detection in a Single Image

然而由于核范数和1范数平等对待所有奇异值，算出来的秩与真秩有些许偏差，所以迭代结果可能只是局部最优解，也就是说，不能精确分离复杂图像的背景和目标。

后人通过 non-convex rank approximation 来改进 principal component analysis (PCA)，同时新提出的 alternating direction method of multipliers (ADMM) 比 accelerated proximal gradient (APG) 收敛更快更精确，可以进一步优化算法。

ReWIPI模型

传统IPI模型由于l1范数不能完全描述稀疏性的缺陷，会过于缩小小目标或在目标图像中留下背景成分。并且由于强边缘也有可能是稀疏的，无法简单地与小目标进行区分。

通过结合结构先验信息，为每个patch自适应权重，提出了名为加权IPI weighted IPI (WIPI) 的方法，然而每个patch都要进行计算，十分费时。并且仅能分离某些特定类型的强边缘。

WIPI的作者分析，其性能不令人满意的原因是**缺少相似的边缘样本**。虽然在奇异值部分和最小化partial sum minimisation of singular values (PSSV) 方法的帮助下，可以保留较大的奇异值。然而，该方法仍需准确估计目标的秩，这实际上很难实现。

而且现有的基于低秩的方法没有考虑到亮度较低的非目标稀疏点的存在，很容易误认成为目标。

进一步分析，基于强边缘是否属于核范数最小化假设的相似边缘样本，可以将强边缘划分为强势的强边缘与弱势的强边缘。这两种强边缘都是全局稀疏的，但是只有弱势的强边缘是面片patch之间仍具有稀疏性。通过最后分离的结果来看，只有弱势的强边缘留在了目标图像中，也就是说，**面片之间的稀疏性比面片内部的稀疏性更容易使得目标留在目标图像**。所以**IPI模型性能不足的真实原因是存在具有面片间稀疏性的弱势强边缘同小目标进行混淆**。

所以我们急需一种方法可以抑制目标图像中的非目标稀疏点的同时保持背景强边缘。

基于**加权核范数最小化weighted nuclear norm minimisation (WNNM)可以通过较小的权重惩罚较大的奇异值**的特点，可以用来得到更为准确的背景图像。并且此方法并不需要准确计算出背景图像的秩。还可以**使用加权l1范数weighted l1 norm，通过较大的权重惩罚非目标图像**，得到更为准确的目标图像。根据这两种想法，可以提出一种新的IPI模型，reweighted IPI。

加权核范数定义为：

$$\| \boldsymbol{B} \|_{\boldsymbol{w},*} = \sum\nolimits_j w_j \sigma_j(\boldsymbol{B}) \tag{6}$$

$$w_j^{k+1} = \frac{1}{\sigma_j^k(\boldsymbol{B}) + \varepsilon_{\boldsymbol{B}}} \tag{7}$$

加权1范数定义为：

$$\| \boldsymbol{T} \|_{\boldsymbol{W},1} = \| \boldsymbol{W} \odot \boldsymbol{T} \|_1 \tag{8}$$

$$W_{ij}^{k+1} = \frac{1}{\left|T_{ij}^{k}\right| + \varepsilon_T} \tag{9}$$

对于噪声图像，我们假设其符合高斯分布，所以有：

$$\| \boldsymbol{D} - \boldsymbol{B} - \boldsymbol{T} \|_{\mathrm{F}} \leq \delta \tag{10}$$

于是ReWIPI可表示成：

$$\min_{\boldsymbol{B},\boldsymbol{T}} \| \boldsymbol{B} \|_{\boldsymbol{w},*} + \lambda \| \boldsymbol{T} \|_{\boldsymbol{W},1} \quad \text{s.t.} \quad \| \boldsymbol{D} - \boldsymbol{B} - \boldsymbol{T} \|_{\mathrm{F}} \leq \delta \tag{11}$$

该问题可以用拉格朗日乘子法等多种方法求解，见下文

参考：Small target detection based on reweighted infrared patch-image model

Robust PCA via Nonconvex Rank Approximation

主成分分析principal component analysis(PCA)是一种很好的将原始高维数据投影到低维空间的降维技术。然而，一旦存在一个严重偏离实际的数据，PCA的结果将会不尽人意。

为了增强对异常值或观测时损坏的数据的鲁棒性，我们需要一种算法进行Robust PCA（RPCA），并尽可能保证算法复杂度较低。

通常来说，问题可以建模成：

$$\min_{L,S} rank(L) + \lambda \|S\|_0 \quad s.t. \quad X = L + S, \tag{1}$$

找到实际低秩矩阵的秩和实际稀疏矩阵的非零元个数，也就是最小化所认定为低秩矩阵的秩以及所认定的稀疏矩阵的非零元个数。

以上问题是一个NP-Hard问题。但可以通过将非凸的秩函数放松成核函数，将l0范数放松成l1范数进行简化成凸函数：

$$\min_{L,S} \|L\|_* + \lambda \|S\|_1 \quad s.t. \quad X = L + S, \tag{2}$$

在不相干假设下，低秩矩阵和稀疏分量可以以压倒性的概率准确恢复。参考：Robust principal component analysis?

遇到的问题：

1. 不是所有矩阵都能有一致性保证（满足不相干假设），数据可能会严重损坏，这样求出的最优解会明显偏离真值。

2. 核函数本质上是矩阵奇异值的l1范数，然而l1范数本身就有收缩效应，这会导致得到的估计是有偏的。也就是说，RPCA将所有奇异值平均加权实际上过度惩罚了大的奇异值，导致结果偏离了较多。

虽然我们可以利用对l1范数的非凸惩罚，如截断的l1范数进行修正。但是这些方法只适用于特殊场景。

于是提出利用一种新的非凸函数进行对秩的逼近，通过增强的拉格朗日乘子法 Augmented Lagrange
Multiplier (ALM) 求解此非凸函数。

定义的新γ范数：

We define γ-norm of matrix L as

$$\|L\|_\gamma = \sum_i \frac{(1+\gamma)\sigma_i(L)}{\gamma + \sigma_i(L)}, \quad \gamma > 0. \tag{6}$$

问题变成：

$$\min_{L,S} \|L\|_\gamma + \lambda\|S\|_l \quad s.t. \quad X = L + S, \tag{5}$$

ALM：

$$\begin{aligned}\mathcal{L}(L,S,Y,\mu) &= \|L\|_\gamma + \lambda\|S\|_l + \\ &\langle Y, L+S-X\rangle + \frac{\mu}{2}\|L+S-X\|_F^2,\end{aligned} \tag{7}$$

Y是拉格朗日乘子，用于消除等式约束，μ是一个正参数，用来稍加约束误差，引入 Frobenius 范数计算误差作为二次惩罚项。<·, ·>是两个矩阵的内积，也可以表示成 $tr(A^TB)$.

通过以下方程更新L，Y，μ直至收敛：

$$L^{t+1} = \arg\min_L \ \|L\|_\gamma + \frac{\mu^t}{2}\left\|L - (X - S^t - \frac{Y^t}{\mu^t})\right\|_F^2. \tag{8}$$

$$S^{t+1} = \arg\min_S \ \lambda\|S\|_l + \frac{\mu^t}{2}\left\|S - (X - L^{t+1} - \frac{Y^t}{\mu^t})\right\|_F^2. \tag{13}$$

$$Y^{t+1} = Y^t + \mu^t(L^{t+1} - X + S^{t+1}), \tag{16}$$

$$\mu^{t+1} = \rho\mu^t, \tag{17}$$

L的迭代求解：

因为对于优化问题：$\min_{Z} F(Z)+\frac{\mu}{2}||Z-A||_F^2$的最优解$Z^*$可SVD成$U\sum_Z^* V^T$，$\sum_Z^*=diag(\sigma^*)$,$\sigma*$是优化问题$arg\min_{\sigma\geq 0} f(\sigma)+\frac{\mu}{2}\left\|\sigma-\sigma_A\right\|_F^2$的最优解。

而上述问题又是凹函数和凸函数的联合，可以用差分凸规划 `difference of convex (DC) programing`迭代优化,直至收敛：

$$\sigma^{k+1}=(\sigma_A-\frac{\omega_k}{\mu^t})_+, \tag{12}$$

其中w_k是$f(\sigma_k)$的梯度

最后$L^{t+1}=Udiag(\sigma^*)V^T$

S的迭代求解：

若方程13用的是S的联合2,1范数，则解可以表示成：

$$[S^{t+1}]_{:,i}=\begin{cases}\frac{\|Q_{:,i}\|_2-\frac{\lambda}{\mu^t}}{\|Q_{:,i}\|_2}Q_{:,i}, & \text{if } \|Q_{:,i}\|_2>\frac{\lambda}{\mu^t};\\ 0, & \text{otherwise,}\end{cases} \tag{14}$$

其中：$Q=X-L^{t+1}-\frac{Y^t}{\mu^t}$,$[S^{t+1}]_{:,i}$ 是S^{t+1}的第i列，

$||S||_{2,1}=\sum_i\sqrt{\sum_j S_{ij}^2}$

若用的S的1范数，则解可以表示成：

$$[S^{t+1}]_{ij}=\max\left(|Q_{ij}|-\frac{\lambda}{\mu^t},0\right)sign(Q_{ij}). \tag{15}$$

参数设置

1. λ

   λ太大会导致S迭代成0，最后L仍为一个高秩矩阵，λ太小会导致L最后为0，可以将λ设置成$1/\sqrt{\max(m,n)}$邻域的任意值，实验证明λ在相当范围内不敏感，可以设置成`10e-3`

2. ρ

   $\rho>1$，若ρ较大，则收敛更快，若ρ较小，则结果更精确。通常取`1.1`。

3. μ

   可分别取`1e-4, 3e-3, 0.5,4`进行实验确定效果

参考论文：Robust PCA via Nonconvex Rank Approximation

根据以上工作，提出以下方法

Infrared Small Target Detection via Non-Convex Rank Approximation Minimization Joint l2,1 Norm

算法原理

$$f_D = f_B + f_T + f_N \tag{3}$$

引用一种 γ norm

$$\|B\|_\gamma = \sum_i \frac{(1+\gamma)\sigma_i(B)}{\gamma + \sigma_i(B)}, \qquad \gamma > 0 \tag{4}$$

γ趋近0是B的秩，趋于无穷是B的核范数。

显然，γ范数几乎与真秩一致（此处使用γ=0.002），解决了传统凸核范数中不同奇异值的不平衡惩罚

weighted nuclear norm (WNN) 的逼近效果也很好，但每次都要重新计算权重时都要重新进行奇异值分解，增加了计算量。总体性能并不能比上γ范数。

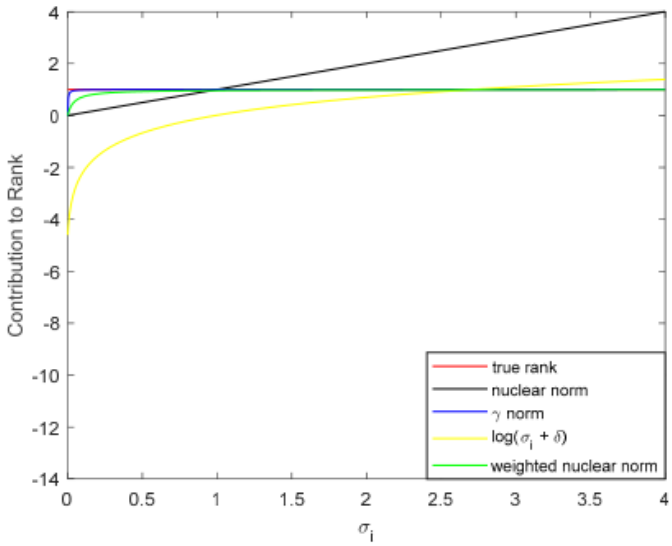

Figure 2. The contribution of different surrogates to the rank with respect to a varying singular value. The true rank is 1 for a nonzero σ_i.

加权1范数

$$\|T\|_{W,1} = \sum_{ij} W_{ij}|T_{ij}| \tag{5}$$

$$W_{ij} = \frac{C}{|T_{ij}| + \varepsilon_T} \tag{6}$$

联合2,1范数

$$\|E\|_{2,1} = \sum_i \sqrt{\sum_j E_{ij}^2} \tag{7}$$

于是目标问题就变成

$$\begin{aligned} \min \quad & \|B\|_\gamma + \lambda\|T\|_{w,1} + \beta\|E\|_{2,1} \\ s.t. \quad & X = B + T + E \end{aligned} \tag{8}$$

引入拉格朗日乘子项和二次惩罚项将问题简化

$$L(D,B,T,E,Y,\mu) = \|B\|_\gamma + \lambda\|T\|_{w,1} + \beta\|E\|_{2,1} + \langle Y, D-B-T-E\rangle + \frac{\mu}{2}\|D-B-T-E\|_F^2 \tag{9}$$

问题进一步转换成一下子问题：

$$B^{k+1} = \underset{B}{\operatorname{argmin}} \|B\|_\gamma + \frac{\mu^k}{2}\left\|D - B - T^k - E^k - \frac{Y^k}{\mu^k}\right\|_F^2 \tag{10}$$

$$T^{k+1} = \underset{T}{\operatorname{argmin}} \lambda\|T\|_{W,1} + \frac{\mu^k}{2}\left\|D - B^{k+1} - T - E^k - \frac{Y^k}{\mu^k}\right\|_F^2 \tag{11}$$

$$E^{k+1} = \underset{E}{\operatorname{argmin}} \beta\|E\|_{2,1} + \frac{\mu^k}{2}\left\|D - B^{k+1} - T^{k+1} - E - \frac{Y^k}{\mu^k}\right\|_F^2 \tag{12}$$

可以通过最小化加权核范数逐渐求得B

At the $(t+1)$th inner iteration

$$\sigma^{t+1} = \underset{\sigma \geq 0}{\operatorname{argmin}} \langle \omega_t, \sigma\rangle + \frac{\mu^k}{2}\|\sigma - \sigma_A\|_2^2 \tag{13}$$

which admits a closed-form solution

$$\sigma^{t+1} = \max(\sigma_A - \frac{\omega_t}{\mu^k}, 0) \tag{14}$$

对于联合2,1范数，T的迭代解可以表示成：

According to [55] and [28], Equations (11) and (12) can be solved as

$$T^{k+1} = S_{\lambda W/\mu^k}(D - B^{k+1} - E^k - \frac{Y^k}{\mu^k}) \tag{15}$$

$$[E^{k+1}]_{:,i} = \begin{cases} \frac{\|Q_{:,i}\|_2 - \frac{\beta}{\mu^k}}{\|Q_{:,i}\|_2} Q_{:,i} & \text{if } \|Q_{:,i}\|_2 > \frac{\beta}{\mu^k} \\ 0 & \text{otherwise} \end{cases} \tag{16}$$

where $Q = D - B^{k+1} - T^{k+1} - \frac{Y^k}{\mu^k}$ and $[E^{k+1}]_{:,i}$ is the i-th column of E^{k+1}.

Y and μ updates in the standard way:

$$Y^{k+1} = Y^k + \mu^k(D - B^{k+1} - T^{k+1} - E^{k+1}) \tag{17}$$

$$\mu^{k+1} = \rho\mu^k \tag{18}$$

where $\rho > 1$.

算法流程如下：

Algorithm 1: ALM/DC solver to the NRAM model

Input: Original patch-image D, λ, β, μ^0, γ;
Output: $B = B^k + E^k$, $T = T^k$;
Initialize: $B^0 = T^0 = E^0 = Y^0 = 0$, $\rho = 1.1$, $\varepsilon = 10^{-7}$, $k = 0$, $W^0 = 1 * 1^{\mathrm{T}}$, $\Lambda^0 = 0 \in R^{\min(m,n)\times 1}$;
While not converged **do**
1: Fix the others and update B by DC programming;
2: Fix the others and update T by

$$T^{k+1} = S_{\lambda W/\mu^k}(D - B^{k+1} - E^k - \frac{Y^k}{\mu^k});$$

3: Fix the others and update E by

$$[E^{k+1}]_{:,i} = \begin{cases} \dfrac{\|Q_{:,i}\|_2 - \dfrac{\beta}{\mu^k}}{\|Q_{:,i}\|_2} Q_{:,i} & \text{if } \|Q_{:,i}\|_2 > \dfrac{\beta}{\mu^k} \\ 0 & \text{otherwise} \end{cases};$$

4: Fix the others and update Y by

$$Y^{k+1} = Y^k + \mu^k(D - B^{k+1} - T^{k+1} - E^{k+1});$$

5: Update W by

$$W_{ij} = \frac{C}{\left|T_{ij}\right| + \varepsilon_T};$$

6: Update μ by

$$\mu^{k+1} = \rho\mu^k;$$

7: Check the convergence conditions

$$\frac{\|D - B^{k+1} - T^{k+1} - E^{k+1}\|_F}{\|D\|_F} < \varepsilon;$$

8: Update k;

$$k = k + 1;$$

End while

Algorithm 2: DC programming

Input: B^k, γ, $A = D - T^k - E^k - \dfrac{Y^k}{\mu^k}$, Λ^0;
Output: B^{k+1}, Λ^{t+1};
Initialize: $t = 0$, $[U, S, V] = \mathrm{svd}(A)$;
While not converged **do**
1: Calculate

$$\omega_t = \frac{(1+\gamma)\gamma}{(\gamma + \Lambda^t)^2}, \Lambda^{t+1} = \max(S - \frac{\omega_t}{\mu^k}, 0);$$

2: Check the convergence conditions

$$\|\Lambda^{t+1} - \Lambda^t\|_2 < \varepsilon;$$

End while
3: $B^{k+1} = U\Lambda^{t+1}V$;

对应matlab程序

```matlab
% 主程序
clc;clear;
close all;

addpath('functions/')
addpath('tools/')
saveDir = 'results/';
imgpath = 'images/';
imgDir = dir([imgpath '*.bmp']);

% patch parameters
patchSize = 40;
slideStep = 40;
```

```
lambdaL = 0.7;  %tuning

len = length(imgDir);
% for i=1:len
for i=1:len
    img = imread([imgpath imgDir(i).name]);
    figure,subplot(131)
    imshow(img),title('Original image')

    if ndims( img ) == 3
        img = rgb2gray( img );
    end
    img = double(img);

    %% constrcut patch tensor of original image
    tenD = gen_patch_ten(img, patchSize, slideStep);
    [n1,n2,n3] = size(tenD);

    %% calculate prior weight map
    %      step 1: calculate two eigenvalues from structure tensor
    [lambda1, lambda2] = structure_tensor_lambda(img, 3);
    %      step 2: calculate corner strength function
    cornerStrength = (((lambda1.*lambda2)./(lambda1 + lambda2)));
    %      step 3: obtain final weight map
    maxValue = (max(lambda1,lambda2));
    priorWeight = mat2gray(cornerStrength .* maxValue);
    %      step 4: constrcut patch tensor of weight map
    tenW = gen_patch_ten(priorWeight, patchSize, slideStep);

    %% The proposed model
    lambda = lambdaL / sqrt(max(n1,n2)*n3);
    [tenB,tenT] = trpca_pstnn(tenD,lambda,tenW);

    %% recover the target and background image
    tarImg = res_patch_ten_mean(tenT, img, patchSize, slideStep);
    backImg = res_patch_ten_mean(tenB, img, patchSize, slideStep);

    maxv = max(max(double(img)));
    E = uint8( mat2gray(tarImg)*maxv );
    A = uint8( mat2gray(backImg)*maxv );
    subplot(132),imshow(E,[]),title('Target image')
    subplot(133),imshow(A,[]),title('Background image')
    % save the results
    imwrite(E, [saveDir 'target/' imgDir(i).name]);
    imwrite(A, [saveDir 'background/' imgDir(i).name]);

end


% 滑动窗提取Patch
function patchTen = gen_patch_ten(img, patchSize, slideStep)

% 2017-07-31
```

```
% This matlab code implements the RIPT model for infrared target-background
% separation.
%
% Yimian Dai. Questions? yimian.dai@gmail.com
% Copyright: College of Electronic and Information Engineering,
%            Nanjing University of Aeronautics and Astronautics


if ~exist('patchSize', 'var')
    patchSize = 50;
end

if ~exist('slideStep', 'var')
    slideStep = 10;
end

% img = reshape(1:9, [3 3])
% img = reshape(1:12, [3 4])
% patchSize = 2;
% slideStep = 1;
[imgHei, imgWid] = size(img);

rowPatchNum = ceil((imgHei - patchSize) / slideStep) + 1;
colPatchNum = ceil((imgWid - patchSize) / slideStep) + 1;
rowPosArr = [1 : slideStep : (rowPatchNum - 1) * slideStep, imgHei - patchSize + 1];
colPosArr = [1 : slideStep : (colPatchNum - 1) * slideStep, imgWid - patchSize + 1];

%% arrayfun version, identical to the following for-loop version
[meshCols, meshRows] = meshgrid(colPosArr, rowPosArr);
idx_fun = @(row,col) img(row : row + patchSize - 1, col : col + patchSize - 1);
patchCell = arrayfun(idx_fun, meshRows, meshCols, 'UniformOutput', false);
patchTen = cat(3, patchCell{:});

%% for-loop version
% patchTen = zeros(patchSize, patchSize, rowPatchNum * colPatchNum);
% k = 0;
% for col = colPosArr
%     for row = rowPosArr
%         k = k + 1;
%         tmp_patch = img(row : row + patchSize - 1, col : col + patchSize - 1);
%         patchTen(:, :, k) = tmp_patch;
%     end
% end
end


% 计算结构化张量的特征值
function [lambda_1, lambda_2] = structure_tensor_lambda(img, sz)
```

```
115
116 G = fspecial('gaussian', [sz sz], 2); % Gaussian kernel
117 u = imfilter(img, G, 'symmetric');
118 [Gx, Gy] = gradient(u);
119
120 K = fspecial('gaussian', [sz sz], 9); % Gaussian kernel
121 J_11 = imfilter(Gx.^2, K, 'symmetric');
122 J_12 = imfilter(Gx.*Gy, K, 'symmetric');
123 J_21 = J_12;
124 J_22 = imfilter(Gy.^2, K, 'symmetric');
125
126 sqrt_delta = sqrt((J_11 - J_22).^2 + 4*J_12.^2);
127 lambda_1 = 0.5*(J_11 + J_22 + sqrt_delta);
128 lambda_2 = 0.5*(J_11 + J_22 - sqrt_delta);
129
130 % 计算目标和背景
131 function [L,S] = trpca_pstnn(X, lambda, tenW, opts)
132
133 tol = 1e-3;
134 max_iter = 500;
135 rho = 1.05;
136 mu = 2*1e-3;
137 max_mu = 1e10;
138 DEBUG = 1;
139 N = rankN(X,0.1);
140
141
142 if ~exist('opts', 'var')
143     opts = [];
144 end
145 if isfield(opts, 'tol');         tol = opts.tol;              end
146 if isfield(opts, 'max_iter');    max_iter = opts.max_iter;    end
147 if isfield(opts, 'rho');         rho = opts.rho;              end
148 if isfield(opts, 'mu');          mu = opts.mu;                end
149 if isfield(opts, 'max_mu');      max_mu = opts.max_mu;        end
150 if isfield(opts, 'DEBUG');       DEBUG = opts.DEBUG;          end
151 if isfield(opts, 'N');           N = opts.N;                  end
152
153 dim = size(X);
154 L = zeros(dim);
155 S = zeros(dim);
156 Y = zeros(dim);
157 weightTen = ones(dim);
158 for iter = 1 : max_iter
159
160     preT = sum(S(:) > 0);
161
162     % update L
163     R = -S+X-Y/mu;
164     L = prox_pstnn(R,N,mu);
165
166     % update S
167     T = -L+X-Y/mu;
168     S = prox_l1(T, weightTen*lambda/mu);
```

```
    weightTen = N./ (abs(S) + 0.01)./tenW;

    dY = L+S-X;
    err = norm(dY(:))/norm(X(:));
    if DEBUG
        if iter == 1 || mod(iter, 1) == 0
            disp(['iter ' num2str(iter) ', mu=' num2str(mu) ...
                   ', err=' num2str(err)...
                    ',|T|0 = ' num2str(sum(S(:) > 0))]);
        end
    end
    currT = sum(S(:) > 0);
    if err < tol || (preT>0 && currT>0 && preT == currT)
        break;
    end
    Y = Y + dY*mu;
    mu = min(rho*mu,max_mu);
end


% 估计矩阵的秩
function N = rankN(X, ratioN)
    [~,~,n3] = size(X);
    D = Unfold(X,n3,1);
    [~, S, ~] = svd(D, 'econ');
    [desS, ~] = sort(diag(S), 'descend');
    ratioVec = desS / desS(1);
    idxArr = find(ratioVec < ratioN);
    if idxArr(1) > 1
        N = idxArr(1) - 1;
    else
        N = 1;
    end
end


% 核范数的优化
function [X] = prox_pstnn(Y,N,mu)

[n1,n2,n3] = size(Y);
X = zeros(n1,n2,n3);
Y = fft(Y,[],3);
tau = 1/mu;


% first frontal slice
[U,S,V] = svd(Y(:,:,1),'econ');
diagS = diag(S);
[desS, sIdx] = sort(diagS, 'descend');
[desU, desV] = deal(U(:, sIdx), V(:, sIdx));
[U1, diagS1, V1] = deal(desU(:, 1:N), desS(1:N), desV(:, 1:N));
[U2, diagS2, V2] = deal(desU(:, N+1:end), desS(N+1:end), desV(:,
N+1:end));
threshS2 = max(diagS2-tau, 0);
```

```
X(:,:,1) = U1*diag(diagS1)*V1' + U2*diag(threshS2)*V2';


% i=2,...,halfn3
halfn3 = round(n3/2);
for i = 2 : halfn3
    [U,S,V] = svd(Y(:,:,i),'econ');
    diagS = diag(S);
    [desS, sIdx] = sort(diagS, 'descend');
    [desU, desV] = deal(U(:, sIdx), V(:, sIdx));
    [U1, diagS1, V1] = deal(desU(:, 1:N), desS(1:N), desV(:, 1:N));
    [U2, diagS2, V2] = deal(desU(:, N+1:end), desS(N+1:end), desV(:,
N+1:end));
    threshS2 = max(diagS2-tau, 0);
    X(:,:,i) = U1*diag(diagS1)*V1' + U2*diag(threshS2)*V2';
    X(:,:,n3+2-i) = conj(X(:,:,i));
end

% if n3 is even
if mod(n3,2) == 0
    i = halfn3+1;
    [U,S,V] = svd(Y(:,:,i),'econ');
    diagS = diag(S);
    [desS, sIdx] = sort(diagS, 'descend');
    [desU, desV] = deal(U(:, sIdx), V(:, sIdx));
    [U1, diagS1, V1] = deal(desU(:, 1:N), desS(1:N), desV(:, 1:N));
    [U2, diagS2, V2] = deal(desU(:, N+1:end), desS(N+1:end), desV(:,
N+1:end));
    threshS2 = max(diagS2-tau, 0);
    X(:,:,i) = U1*diag(diagS1)*V1' + U2*diag(threshS2)*V2';
end

X = ifft(X,[],3);

end


% 1范数的优化
function x = prox_l1(b,lambda)

x = max(0,b-lambda)+min(0,b+lambda);
x = max(x,0);
end


% 合并面片为图像
function recImg = res_patch_ten_mean(patchTen, img, patchSize,
slideStep)

% 2017-07-31
% This matlab code implements the RIPT model for infrared target-
background
% separation.
%
```

```
% Yimian Dai. Questions? yimian.dai@gmail.com
% Copyright: College of Electronic and Information Engineering,
%            Nanjing University of Aeronautics and Astronautics

[imgHei, imgWid] = size(img);

rowPatchNum = ceil((imgHei - patchSize) / slideStep) + 1;
colPatchNum = ceil((imgWid - patchSize) / slideStep) + 1;
rowPosArr = [1 : slideStep : (rowPatchNum - 1) * slideStep, imgHei - patchSize + 1];
colPosArr = [1 : slideStep : (colPatchNum - 1) * slideStep, imgWid - patchSize + 1];

%% for-loop version
accImg = zeros(imgHei, imgWid);
weiImg = zeros(imgHei, imgWid);
k = 0;
onesMat = ones(patchSize, patchSize);
for col = colPosArr
    for row = rowPosArr
        k = k + 1;
        tmpPatch = reshape(patchTen(:, :, k), [patchSize, patchSize]);
        accImg(row : row + patchSize - 1, col : col + patchSize - 1) = tmpPatch;
        weiImg(row : row + patchSize - 1, col : col + patchSize - 1) = onesMat;
    end
end

recImg = accImg ./ weiImg;
end

```

根据matlab代码自己写的python代码：

```python
import numpy as np
import matplotlib.pyplot as plt
import cv2
import math
import scipy


def init(path):
    pic = plt.imread(path)
    plt.figure()
    plt.subplot(131)
    plt.title('Original')
    if pic.ndim == 2:
        plt.imshow(pic, cmap='gray')
    if pic.ndim == 3:
        plt.imshow(pic)
        pic = cv2.cvtColor(pic, cv2.COLOR_RGB2GRAY)
    pic = np.float64(pic)
```

```
    return pic


def gen_patch(pic, patch_size=40, slide_step=40):

    pic_H, pic_W = np.shape(pic)[0:2]

    row_patch_num = math.ceil((pic_H - patch_size) / slide_step + 1)
    col_patch_num = math.ceil((pic_W - patch_size) / slide_step + 1)
    row_pos_arr = list(range(0, slide_step * (row_patch_num - 1),
slide_step))
    row_pos_arr.append(pic_H - patch_size)
    col_pos_arr = list(range(0, slide_step * (col_patch_num - 1),
slide_step))
    col_pos_arr.append(pic_W - patch_size)

    Patch = np.zeros((patch_size, patch_size, row_patch_num *
col_patch_num))
    k = 0
    for j in col_pos_arr:
        for i in row_pos_arr:
            patch_cell = pic[i:i + patch_size, j:j + patch_size]
            Patch[:, :, k] = patch_cell
            k += 1

    return Patch


def res_patch(patch, pic, patch_size=40, slide_step=40):

    pic_H, pic_W = np.shape(pic)[0:2]

    row_patch_num = math.ceil((pic_H - patch_size) / slide_step + 1)
    col_patch_num = math.ceil((pic_W - patch_size) / slide_step + 1)
    row_pos_arr = list(range(0, slide_step * (row_patch_num - 1),
slide_step))
    row_pos_arr.append(pic_H - patch_size)
    col_pos_arr = list(range(0, slide_step * (col_patch_num - 1),
slide_step))
    col_pos_arr.append(pic_W - patch_size)

    accImg = np.zeros((pic_H, pic_W))
    weiImg = np.zeros((pic_H, pic_W))
    onesMat = np.ones((patch_size, patch_size))
    k = 0
    for j in col_pos_arr:
        for i in row_pos_arr:
            tempPatch = np.reshape(patch[:, :, k], (patch_size,
patch_size))
            accImg[i:i+patch_size, j:j+patch_size] = tempPatch
            weiImg[i:i+patch_size, j:j+patch_size] = onesMat
            k += 1

```

```
    return accImg / weiImg


def structure_tensor_lambda(pic, size=3):
    Gaussian_kernel = cv2.getGaussianKernel(size, 2) @
cv2.getGaussianKernel(3, 2).T
    u = scipy.ndimage.convolve(pic, Gaussian_kernel)
    Gx, Gy = np.gradient(u)

    Gaussian_kernel = cv2.getGaussianKernel(size, 9) @
cv2.getGaussianKernel(3, 9).T
    J_11 = scipy.ndimage.convolve(Gx * Gx, Gaussian_kernel)
    J_12 = scipy.ndimage.convolve(Gx * Gy, Gaussian_kernel)
    J_21 = J_12
    J_22 = scipy.ndimage.convolve(Gy * Gy, Gaussian_kernel)

    sqrt_delta = ((J_11 - J_22) ** 2 + 4 * (J_12 ** 2)) ** (1 / 2)
    lambda1 = 0.5 * (J_11 + J_22 + sqrt_delta)
    lambda2 = 0.5 * (J_11 + J_22 - sqrt_delta)

    return lambda1, lambda2


def prox_pstnn(Y, N, mu):
    dim = Y.shape
    X = np.zeros(dim, dtype='complex')
    Y = np.fft.fft(Y)
    tau = 1 / mu

    U, diagS, V = np.linalg.svd(Y[:, :, 0])
    des_S = np.sort(diagS)[::-1]
    idx_S = np.argsort(diagS)[::-1]
    des_U, des_V = U[:, idx_S], V.T[:, idx_S]
    U1, diagS1, V1 = des_U[:, 0:N+1], des_S[0:N+1], des_V[:, 0:N+1]
    U2, diagS2, V2 = des_U[:, N+1:], des_S[N+1:], des_V[:, N+1:]
    threshS2 = np.fmax(diagS2 - tau, 0)
    X[:, :, 0] = U1 @ np.diag(diagS1) @ V1.T + U2 @ np.diag(threshS2) @
V2.T

    half_n3 = round(dim[2] / 2)
    for i in range(1, half_n3 + 1):
        U, diagS, V = np.linalg.svd(Y[:, :, i])
        des_S = np.sort(diagS)[::-1]
        idx_S = np.argsort(diagS)[::-1]
        des_U, des_V = U[:, idx_S], V.T[:, idx_S]
        U1, diagS1, V1 = des_U[:, 0:N+1], des_S[0:N+1], des_V[:, 0:N+1]
        U2, diagS2, V2 = des_U[:, N+1:], des_S[N+1:], des_V[:, N+1:]
        threshS2 = np.fmax(diagS2 - tau, 0)
        X[:, :, i] = U1 @ np.diag(diagS1) @ V1.T + U2 @
np.diag(threshS2) @ V2.T
        X[:, :, dim[2] - i] = np.conj(X[:, :, i])

    if dim[2] % 2 == 0:
        i = half_n3 + 1
```

```
        U, diagS, V = np.linalg.svd(Y[:, :, i])
        des_S = np.sort(diagS)[::-1]
        idx_S = np.argsort(diagS)[::-1]
        des_U, des_V = U[:, idx_S], V.T[:, idx_S]
        U1, diagS1, V1 = des_U[:, 0:N+1], des_S[0:N+1], des_V[:, 0:N + 1]
        U2, diagS2, V2 = des_U[:, N+1:], des_S[N+1:], des_V[:, N+1:]
        threshS2 = np.fmax(diagS2 - tau, 0)
        X[:, :, i] = U1 @ np.diag(diagS1) @ V1.T + U2 @ np.diag(threshS2) @ V2.T

    X = np.fft.ifft(X)

    return X.real


def prox_l1(b, lambda_):
    x = np.fmax(0, b - lambda_) + np.fmin(0, b + lambda_)
    x = np.fmax(x, 0)

    return x


def rankN(x, ration_n):

    num3 = np.shape(x)[2]
    re_x = x.reshape(num3, -1, order='F')
    S_diag = np.linalg.svd(re_x)[1]
    des_S = np.sort(S_diag)[::-1]
    ratioVec = des_S / des_S[0]
    idxArr = np.where(ratioVec < ration_n)[0]
    if idxArr[0] > 0:
        N = idxArr[0] - 1
    else:
        N = 0

    return N


def trpca_pstnn(X, lambda_, tenW, tol=1e-3, max_iter=500, rho=1.05, mu=2e-3, max_mu=1e10, debug=None):

    N = rankN(X, 0.1)

    dim = np.shape(X)
    L = np.zeros(dim)
    S = np.zeros(dim)
    Y = np.zeros(dim)
    weightTen = np.ones(dim)

    for iter in range(max_iter):
        preT = np.sum(S > 0)

        R = -S + X - Y / mu
```

```
        L = prox_pstnn(R, N, mu)

        T = -L + X - Y / mu
        S = prox_l1(T, weightTen * lambda_ / mu)
        weightTen = N+1 / (np.abs(S) + 0.01) / tenW

        dY = L + S - X
        err = np.linalg.norm(dY) / np.linalg.norm(X)
        currT = np.sum(S[:] > 0)

        if debug:
            print("iter: %d , mu: %.4f, err: %.4f, |T|0: %d" % (iter, mu, err, int(currT)))

        if err < tol or (0 < preT == currT > 0):
            break

        Y = Y + dY * mu
        mu = min(rho * mu, max_mu)

    return L, S


lambdaL = 0.7

img = init("./1.bmp")
D = gen_patch(img)

n1, n2, n3 = D.shape

lambDa1, lambDa2 = structure_tensor_lambda(img)
cornerStrength = (lambDa1 * lambDa2) / (lambDa1 + lambDa2)
maxValue = np.maximum(lambDa1, lambDa2)
priorWeight = np.zeros(maxValue.shape, np.float64)
cv2.normalize(cornerStrength * maxValue, priorWeight, 1.0, 0.0, cv2.NORM_MINMAX, dtype=cv2.CV_64F)
W = gen_patch(priorWeight)

lambda0 = lambdaL / math.sqrt(max(n1, n2) * n3)

B, T_ = trpca_pstnn(D, lambda0, W)

tarImg = res_patch(T_, img)
backImg = res_patch(B, img)

maxV = np.max(img)
E = cv2.normalize(tarImg, None, 1, 0, cv2.NORM_MINMAX, dtype=cv2.CV_64F)*maxV
E = E.astype(np.uint8)
A = cv2.normalize(backImg, None, 1, 0, cv2.NORM_MINMAX, dtype=cv2.CV_64F)*maxV
A = A.astype(np.uint8)
plt.subplot(132)
plt.imshow(E, cmap='gray')
```

```
plt.title("Target")
plt.subplot(133)
plt.imshow(A, cmap='gray')
plt.title("Background")

plt.show()

```

自己的疑问

为什么平均对待奇异值会导致无法区分出背景与目标

为什么会这么计算先验权重图