Overview on object tracking

Dai Haocheng 2018.2

Generative & Discriminate methods

Discriminate methods (tracking-by-detection) aim to build a model that distinguishes the target object from the background.

$$\begin{cases} \boldsymbol{Generative\ methods}: Kalman\ Filter, Particle\ Filter, ASMS \\ \boldsymbol{Discriminative\ methods}: MOSSE, CSK, DCF, KCF, DSST, MDNet \end{cases}$$

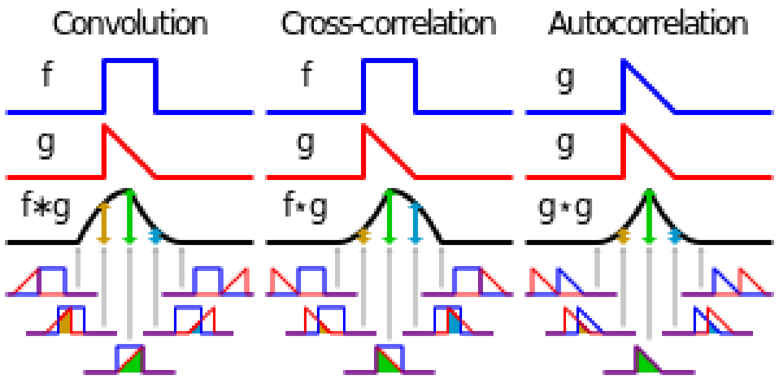
Generative methods describe the target appearances using generative models and search for the target regions that fit the models best.

MOSSE*

① $F = \mathcal{F}(f)$ ② $H = \mathcal{F}(h)$ ③ $g = f \otimes h$ → ④ $G = F \odot H^{*}$

Input image filter Time Domain Frequency Domain

$\mathcal{F}$ denotes the Fast Fourier Transform 快速傅立叶变换



$\otimes$ denotes the convolutional product(卷积)
$\odot$ denotes the element-wise product(点积/数量积).

MOSSE*

① $F = \mathcal{F}(f)$ ② $H = \mathcal{F}(h)$ ③ $g = f \otimes h$ $\rightarrow$ ④ $G = F \odot H^*$

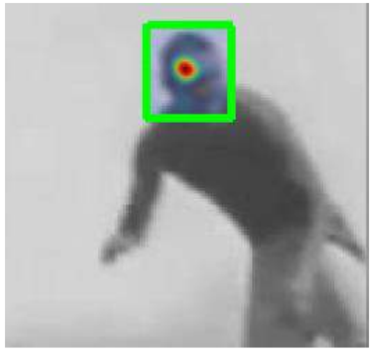
number of training samples

$$⑤ H = \min_{H^*} \sum_{i}^{m} |F_i \odot H^* - G_i|^2 \rightarrow ⑥ H_{\omega v} = \min_{H_{\omega v}} \sum_{i}^{m} |F_{i\omega v} H_{\omega v}^* - G_{i\omega v}|^2$$

傅里叶域中的所有操作都是按元素执行的，
所以H中的每个元素可以独立求解

Minimizing sum of squared error (SSE)

g_i is generated from ground truth such that it has a compact 2-d Gaussian shaped peak centered on the target in training image f_i.



MOSSE*

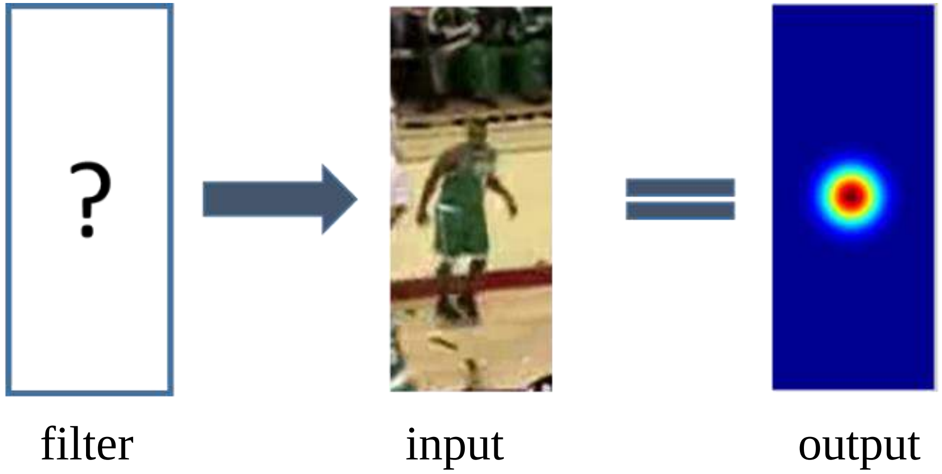
①$F = \mathcal{F}(f)$ ②$H = \mathcal{F}(h)$ ③$g = f \otimes h$ $\rightarrow$ ④$G = F \odot H^*$

⑤$H = \min_{H^*} \sum_{i}^{m} |F_i \odot H^* - G_i|^2$ $\rightarrow$ ⑥$H_{\omega\upsilon} = \min_{H_{\omega\upsilon}} \sum_{i}^{m} |F_{i\omega\upsilon} H^*_{\omega\upsilon} - G_{i\omega\upsilon}|^2$

$\rightarrow$ ⑦$0 = \frac{\partial}{\partial H_{\omega\upsilon}} \sum_{i} |F_{i\omega\upsilon} H^*_{\omega\upsilon} - G_{i\omega\upsilon}|^2$

$\rightarrow$ ⑧$H_{\omega\upsilon} = \frac{\sum_i F_{i\omega\upsilon} G^*_{i\omega\upsilon}}{\sum_i F_{i\omega\upsilon} F^*_{i\omega\upsilon}}$ $\rightarrow$ ⑨$H^* = \frac{\sum_i G_i \odot F_i^*}{\sum_i F_i \odot F_i^*}$ Final result

MOSSE*



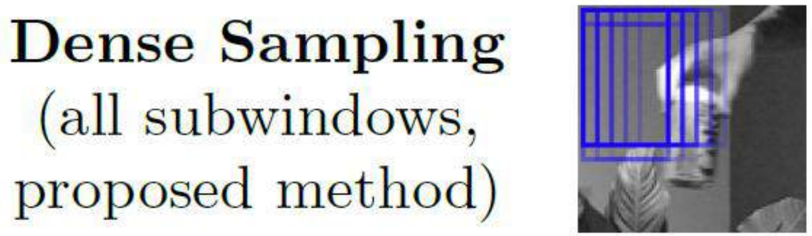
$$H_i^* = \frac{A_i}{B_i}$$

$$A_i = (1 - \eta)A_{i-1} + \eta G_i \odot F_i^*$$

$$B_i = (1 - \eta)B_{i-1} + \eta F_i \odot F_i^*$$

Figure 8: This shows a sample from the **davidin300** sequence for the MOSSE filter. The frame number and PSR are shown in the upper left hand corner. Thin gray box around the face shows starting location of the tracking window for each frame. The thick red box shows the updated location of the track window. The red dot shows the center point of the tracking window and helps to determine if the windows has drifted off the original location. In the Taz video, failure detection is enabled and a red X is drawn in the window to indicate that a failure or occlusion has been detected. Across the bottom of the video are images are the input (cropped from the grey rectangle), the updated filter from this frame, and the correlation output.

CSK*



$(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)$ are the training samples and labels, $f(\mathbf{x})$ is the classifier.

$$\min_{w,b} \sum_{i=1}^{m} \left(y_i - f(\mathbf{x}_i)\right)^2 + \lambda \|\boldsymbol{w}\|^2$$

该模型最终要确定的系数矩阵w，对应于MOSSE中最终要确定滤波模板H

$$\boldsymbol{w} = \sum_i \alpha_i \varphi(\mathbf{x}_i)$$

$$\boldsymbol{\alpha} = (K + \lambda I)^{-1} y$$

Kernel trick

Representer theorem

Kernel matrix Identity matrix

CSK*

① $\boldsymbol{\alpha} = (K + \lambda I)^{-1} y$

Permutation matrix

$$\mathbf{x}_m = P^m \mathbf{x}$$

$$K = \begin{bmatrix} \kappa(\mathbf{x}_1, \mathbf{x}_1) & \dots & \kappa(\mathbf{x}_1, \mathbf{x}_j) & \dots & \kappa(\mathbf{x}_1, \mathbf{x}_m) \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ \kappa(\mathbf{x}_i, \mathbf{x}_1) & \dots & \kappa(\mathbf{x}_i, \mathbf{x}_j) & \dots & \kappa(\mathbf{x}_i, \mathbf{x}_m) \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ \kappa(\mathbf{x}_m, \mathbf{x}_1) & \dots & \kappa(\mathbf{x}_m, \mathbf{x}_j) & \dots & \kappa(\mathbf{x}_m, \mathbf{x}_m) \end{bmatrix}$$

$$\kappa^{gauss}(\mathbf{x}_m, \mathbf{x}_j) = \exp(-\frac{1}{\sigma^2}\left(\|\mathbf{x}_m\|^2 + \|\mathbf{x}_j\|^2 - 2\mathcal{F}^{-1}\left(\mathcal{F}(\mathbf{x}_m) \odot \mathcal{F}^*(\mathbf{x}_j)\right)\right))$$

$\odot$ denotes the element-wise product(点积/数量积).

CSK*

训练

$$\textcircled{1}\ \boldsymbol{\alpha} = (K + \lambda I)^{-1} y \longrightarrow \textcircled{2}\ \boldsymbol{\alpha} = \mathcal{F}^{-1}\left(\frac{\mathcal{F}(\boldsymbol{y})}{\mathcal{F}(\boldsymbol{k}) + \lambda}\right)$$

$$K = \begin{bmatrix} \boxed{\kappa(\mathbf{x},\mathbf{x}_1) & \kappa(\mathbf{x},\mathbf{x}_2) & \kappa(\mathbf{x},\mathbf{x}_3) & \ldots & \kappa(\mathbf{x},\mathbf{x}_m)} \\ \kappa(\mathbf{x},\mathbf{x}_m) & \kappa(\mathbf{x},\mathbf{x}_1) & \kappa(\mathbf{x},\mathbf{x}_2) & \ldots & \kappa(\mathbf{x},\mathbf{x}_{m-1}) \\ \kappa(\mathbf{x},\mathbf{x}_{m-1}) & \kappa(\mathbf{x},\mathbf{x}_m) & \kappa(\mathbf{x},\mathbf{x}_1) & \ldots & \kappa(\mathbf{x},\mathbf{x}_{m-2}) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \kappa(\mathbf{x},\mathbf{x}_2) & \kappa(\mathbf{x},\mathbf{x}_3) & \kappa(\mathbf{x},\mathbf{x}_4) & \ldots & \kappa(\mathbf{x},\mathbf{x}_1) \end{bmatrix} = C(\boldsymbol{k})$$

根据Theorem 1，K矩阵为一个循环矩阵，而循环矩阵可以在傅里叶域变成点积运算，减少了计算量

Theorem 1. *The matrix K with elements $K_{ij} = \kappa(P^i \mathrm{x}, P^j \mathrm{x})$ is circulant if κ is a unitarily invariant kernel.*

CSK*

检测

①$\boldsymbol{\alpha} = (K + \lambda I)^{-1} y$ → ②$\boldsymbol{\alpha} = \mathcal{F}^{-1}\left(\frac{\mathcal{F}(\boldsymbol{y})}{\mathcal{F}(\boldsymbol{k}) + \lambda}\right)$ → ③$y' = \sum_i \alpha_i \kappa(x_i, z)$

responses at all positions → ④$\hat{y} = \mathcal{F}^{-1}(\mathcal{F}(\overline{\boldsymbol{k}}) \odot \mathcal{F}(\boldsymbol{\alpha}))$

$$[\kappa(z, \mathbf{x}_1) \quad \kappa(z, \mathbf{x}_2) \quad \kappa(z, \mathbf{x}_3) \quad \ldots \quad \kappa(z, \mathbf{x}_m)] = \overline{\boldsymbol{k}}$$

CSK* 与MOSSE的关系

$$\mathbf{w} = \mathcal{F}^{-1}\left(\frac{\mathcal{F}(\mathbf{x}) \odot \mathcal{F}^*(\mathbf{y})}{\mathcal{F}(\mathbf{x}) \odot \mathcal{F}^*(\mathbf{x}) + \lambda}\right). \quad (17)$$

This is a kind of correlation filter that has been proposed recently, called Minimum Output Sum of Squared Error (MOSSE) [12, 15], with a single training image. It is remarkably powerful despite its simplicity.

Note, however, that correlation filters are obtained with classical signal processing techniques, directly in the Fourier domain. As we have shown, Circulant matrices are the key enabling factor to extend them with the Kernel Trick.

$$w = \mathcal{F}^{-1}\left(\frac{\mathcal{F}(x)\odot\mathcal{F}^*(y)}{\mathcal{F}(x)\odot\mathcal{F}^*(x) + \lambda}\right)$$

$$H^* = \frac{\sum_i G_i \odot F_i^*}{\sum_i F_i \odot F_i^*}$$

MOSSE

The MOSSE is a linear classifier that doesn't make use of kernel trick, but use the simplest kernel function which is just the dot-product $\kappa(x, x') = \langle x, x' \rangle$ in the original space.

CSK* 程序主体部分

```
%extract and pre-process subwindow
x = get_subwindow(im, pos, sz, cos_window);

if frame > 1,
    %calculate response of the classifier at all locations
    k = dense_gauss_kernel(sigma, x, z);
    response = real(ifft2(alphaf .* fft2(k)));   %(Eq. 9)

    %target location is at the maximum response
    [row, col] = find(response == max(response(:)), 1);
    pos = pos - floor(sz/2) + [row, col];
end

%get subwindow at current estimated target position, to train classifer
x = get_subwindow(im, pos, sz, cos_window);

%Kernel Regularized Least-Squares, calculate alphas (in Fourier domain)
k = dense_gauss_kernel(sigma, x);
new_alphaf = yf ./ (fft2(k) + lambda);   %(Eq. 7)
new_z = x;

if frame == 1,  %first frame, train with a single image
    alphaf = new_alphaf;
    z = x;
else
    %subsequent frames, interpolate model
    alphaf = (1 - interp_factor) * alphaf + interp_factor * new_alphaf;
    z = (1 - interp_factor) * z + interp_factor * new_z;
end
```

用训练好的位置
滤波器更新位置

$$\hat{y} = \mathcal{F}^{-1}(\mathcal{F}(\overline{\boldsymbol{k}}) \odot \mathcal{F}(\boldsymbol{\alpha}))$$

$\boldsymbol{k}$

$$\boldsymbol{\alpha} = \mathcal{F}^{-1}\left(\frac{\mathcal{F}(\boldsymbol{y})}{\mathcal{F}(\boldsymbol{k}) + \lambda}\right)$$

CSK* 核函数部分

```
function k = dense_gauss_kernel(sigma, x, y)

    xf = fft2(x);  %x in Fourier domain
    xx = x(:)' * x(:);  %squared norm of x

    if nargin >= 3,  %general case, x and y are different
        yf = fft2(y);
        yy = y(:)' * y(:);
    else
        %auto-correlation of x, avoid repeating a few operations
        yf = xf;
        yy = xx;
    end

    %cross-correlation term in Fourier domain
    xyf = xf .* conj(yf);
    xy = real(circshift(ifft2(xyf), floor(size(x)/2)));  %to spatial domain

    %calculate gaussian response for all positions
    k = exp(-1 / sigma^2 * max(0, (xx + yy - 2 * xy) / numel(x)));

end
```

$$K = \begin{bmatrix} \kappa(\mathbf{x},\mathbf{x}_1) & \kappa(\mathbf{x},\mathbf{x}_2) & \kappa(\mathbf{x},\mathbf{x}_3) & \dots & \kappa(\mathbf{x},\mathbf{x}_m) \\ \kappa(\mathbf{x},\mathbf{x}_m) & \kappa(\mathbf{x},\mathbf{x}_1) & \kappa(\mathbf{x},\mathbf{x}_2) & \dots & \kappa(\mathbf{x},\mathbf{x}_{m-1}) \\ \kappa(\mathbf{x},\mathbf{x}_{m-1}) & \kappa(\mathbf{x},\mathbf{x}_m) & \kappa(\mathbf{x},\mathbf{x}_1) & \dots & \kappa(\mathbf{x},\mathbf{x}_{m-2}) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \kappa(\mathbf{x},\mathbf{x}_2) & \kappa(\mathbf{x},\mathbf{x}_3) & \kappa(\mathbf{x},\mathbf{x}_4) & \dots & \kappa(\mathbf{x},\mathbf{x}_1) \end{bmatrix} = C(\boldsymbol{k})$$

k是一个向量

$$\kappa^{gauss}(\mathbf{x},\mathbf{x}_j) = \exp(-\frac{1}{\sigma^2}\left(\|\mathbf{x}\|^2 + \|\mathbf{x}_j\|^2 - 2\boxed{\mathcal{F}^{-1}\left(\mathcal{F}(\mathbf{x})\odot\mathcal{F}^*(\mathbf{x}_j)\right)}\right))$$

CSK* 程序主要流程

- Get Subwindow
- $\kappa^{gauss} = \exp(-\frac{1}{\sigma^2}\left(\|x\|^2 + \|x\|^2 - 2\mathcal{F}^{-1}\left(\mathcal{F}(x) \odot \mathcal{F}^*(x)\right)\right))$
- new_alphaf($\boldsymbol{\alpha}$) $= \mathcal{F}^{-1}(\frac{\mathcal{F}(y)}{\mathcal{F}(k)+\lambda})$
- new_z←x
- alphaf ← new_alphaf
- z ←x

1st frame

- Get Subwindow
- $\kappa^{gauss} = \exp(-\frac{1}{\sigma^2}\left(\|x\|^2 + \|z\|^2 - 2\mathcal{F}^{-1}\left(\mathcal{F}(x) \odot \mathcal{F}^*(z)\right)\right))$
- $response(\hat{y}) = \mathcal{F}^{-1}(\mathcal{F}(\overline{\boldsymbol{k}}) \odot \mathcal{F}(\alpha))$
- Find max response
- Change position(*pos*)
- Get Subwindow(new subwindow according to the *pos*)
- $\kappa^{gauss} = \exp(-\frac{1}{\sigma^2}\left(\|x\|^2 + \|x\|^2 - 2\mathcal{F}^{-1}\left(\mathcal{F}(x) \odot \mathcal{F}^*(x)\right)\right))$
- new_alphaf($\boldsymbol{\alpha}$) $= \mathcal{F}^{-1}(\frac{\mathcal{F}(y)}{\mathcal{F}(k)+\lambda})$
- new_z←x
- alphaf ← (1 - interp_factor) * alphaf + interp_factor * new_alphaf
- z ←(1 - interp_factor) * z + interp_factor * new_z

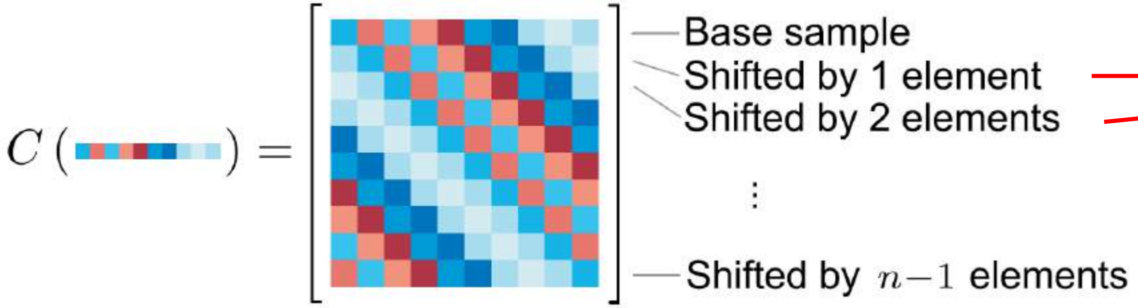
$2^{nd}, 3^{rd}, 4^{th}$ ……frame

CSK* 程序主要流程

- Get Subwindow — 获取子窗口
- $\kappa^{gauss} = \exp(-\frac{1}{\sigma^2}\left(\|x\|^2 + \|x\|^2 - 2\mathcal{F}^{-1}\left(\mathcal{F}(x)\odot\mathcal{F}^*(x)\right)\right))$ — 计算核函数
- new_alphaf($\boldsymbol{\alpha}$) $= \mathcal{F}^{-1}(\frac{\mathcal{F}(y)}{\mathcal{F}(k)+\lambda})$ — 通过核函数和已知响应计算α
- new_z←x — 给new_z赋值
- alphaf ← new_alphaf — 记录alphaf
- z ←x — 给z赋值

- Get Subwindow — 再次获取子窗口
- $\kappa^{gauss} = \exp(-\frac{1}{\sigma^2}\left(\|x\|^2 + \|z\|^2 - 2\mathcal{F}^{-1}\left(\mathcal{F}(x)\odot\mathcal{F}^*(z)\right)\right))$ — 计算核函数（z即为旧的x）
- $response(\hat{y}) = \mathcal{F}^{-1}(\mathcal{F}(\bar{\boldsymbol{k}})\odot\mathcal{F}(\alpha))$ — 通过前一帧训练出来的α计算所有位置的响应y
- Find max response — 找到最大的响应
- Change position(*pos*) — 根据最大响应获取新位置
- Get Subwindow(new subwindow according to the *pos*) — 根据新位置重新获取子窗口
- $\kappa^{gauss} = \exp(-\frac{1}{\sigma^2}\left(\|x\|^2 + \|x\|^2 - 2\mathcal{F}^{-1}\left(\mathcal{F}(x)\odot\mathcal{F}^*(x)\right)\right))$ — 计算核函数
- new_alphaf($\boldsymbol{\alpha}$) $= \mathcal{F}^{-1}(\frac{\mathcal{F}(y)}{\mathcal{F}(k)+\lambda})$ — 更新α
- new_z←x — 记录x的新值
- alphaf ← (1 - interp_factor) * alphaf + interp_factor * new_alphaf — 加权更新alphaf
- z ←(1 - interp_factor) * z + interp_factor * new_z — 加权更新z

CSK*



Every row is a shifted sample based on the first row

1D image

2D image

Appendix A.3: Generalization of circulant forms

For a matter of clarity, all of our derivations have assumed that the images are one-dimensional. The 2D case, despite its usefulness, is also more difficult to analyze. The reason is that the 2D generalization of a circulant matrix, related to the 2D Fourier Transform, is a Block-Circulant Circulant Matrix (BCCM, ie., a matrix that is circulant at the block level, composed of blocks themselves circulant). All of the properties we used for circulant matrices have BCCM equivalents.

We will now generalize Theorem 1. A 1D image $\mathbf{x}$ can be shifted by i with $P^i\mathbf{x}$. With a 2D image X, we can shift both its rows by i and its columns by i' with $P^i X P^{i'}$. Additionally, in an $n^2 \times n^2$ matrix M composed of $n \times n$ blocks, we will index the element $i'j'$ of the block ij as $M_{(ii'),(jj')}$.

Theorem 2. *The block matrix K with elements $K_{(ii'),(jj')} = \kappa(P^i X P^{i'}, P^j X P^{j'})$ is a BCCM if κ is a unitarily invariant kernel.*

+30 +15 Base sample -15 -30

These shifted samples are for training

DCF 线性可分情况下的模型训练

$(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)$ are the training samples and labels, $f(\mathbf{x})$ is the classifier.

① $\min_{w} \sum_{i=1}^{m} \left(y_i - f(\mathbf{x}_i)\right)^2 + \lambda \|\boldsymbol{w}\|^2$

② $\boldsymbol{w} = (X^H X + \lambda I)^{-1} X^H y$ → ③ $\boldsymbol{w} = \mathcal{F}^{-1}\left(\frac{\mathcal{F}(\mathbf{x}^*) \odot \mathcal{F}(y)}{\mathcal{F}(\mathbf{x}^*) \odot \mathcal{F}(\mathbf{x}) + \lambda}\right)$ → $f(\mathbf{z}) = \boldsymbol{w}^T \mathbf{z}$

用第一帧训练出的模型

训练

检测

$$X = \begin{bmatrix} \boxed{x_1 \quad x_2 \quad x_3 \quad \ldots \quad x_n} \\ x_1 & x_1 & x_2 & \ldots & x_{n-1} \\ x_{n-1} & x_n & x_1 & \ldots & x_{n-2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_2 & x_3 & x_4 & \ldots & x_1 \end{bmatrix} = C(\mathbf{x}) = F diag(\mathcal{F}(\mathbf{x})) F^H$$

All circulant matrices are made diagonal by the Discrete Fourier Transform (DFT)

$\odot$ denotes the element-wise product(点积/数量积).

KCF 线性不可分情况下的模型训练

$(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_m, y_m)$ are the training samples and labels, $f(\mathbf{x})$ is the classifier.

① $\min_{w,b} \sum_{i=1}^{m} \left(y_i - f(\mathbf{x}_i)\right)^2 + \lambda \|\boldsymbol{w}\|^2$

② $\boldsymbol{w} = \sum_i \alpha_i \varphi(\mathbf{x}_i)$

Representer theorem

③ $\boldsymbol{\alpha} = (K + \lambda I)^{-1} y$ ← Kernel trick

Kernel matrix Identity matrix

KCF 线性不可分情况下的模型训练

response for a single input z

① $\boldsymbol{\alpha} = (K + \lambda I)^{-1} y$ → ② $\boldsymbol{\alpha} = \mathcal{F}^{-1}\left(\frac{\mathcal{F}(y)}{\mathcal{F}(\boldsymbol{k}^{xx}) + \lambda}\right)$ → ③ $f(\mathbf{z}) = \mathbf{w}^T \mathbf{z} = \sum_i \alpha_i \kappa(z, \mathbf{x}_i)$

用第一帧训练出的模型

循环矩阵对角化 $K = \begin{bmatrix} \boxed{\kappa(\mathbf{x}, \mathbf{x}_1) & \kappa(\mathbf{x}, \mathbf{x}_2) & \kappa(\mathbf{x}, \mathbf{x}_3) & \dots & \kappa(\mathbf{x}, \mathbf{x}_m)} \\ \kappa(\mathbf{x}, \mathbf{x}_m) & \kappa(\mathbf{x}, \mathbf{x}_1) & \kappa(\mathbf{x}, \mathbf{x}_2) & \dots & \kappa(\mathbf{x}, \mathbf{x}_{m-1}) \\ \kappa(\mathbf{x}, \mathbf{x}_{m-1}) & \kappa(\mathbf{x}, \mathbf{x}_m) & \kappa(\mathbf{x}, \mathbf{x}_1) & \dots & \kappa(\mathbf{x}, \mathbf{x}_{m-2}) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \kappa(\mathbf{x}, \mathbf{x}_2) & \kappa(\mathbf{x}, \mathbf{x}_3) & \kappa(\mathbf{x}, \mathbf{x}_4) & \dots & \kappa(\mathbf{x}, \mathbf{x}_1) \end{bmatrix} = C(\boldsymbol{k}^{xx}) = F diag\big(\mathcal{F}(\boldsymbol{k}^{xx})\big) F^H$

训练样本向量$\mathbf{x}_i = P^i \mathbf{x}$

KCF 非线性情况下的模型训练

a single input z

$$①\boldsymbol{\alpha} = (K + \lambda I)^{-1} y \longrightarrow ②\boldsymbol{\alpha} = \mathcal{F}^{-1}\left(\frac{\mathcal{F}(y)}{\mathcal{F}(\boldsymbol{k}^{xx}) + \lambda}\right) \longrightarrow ③ f(\boldsymbol{z}) = \boldsymbol{w}^T \boldsymbol{z} = \sum_i \alpha_i \kappa(z, \mathbf{x}_i)$$

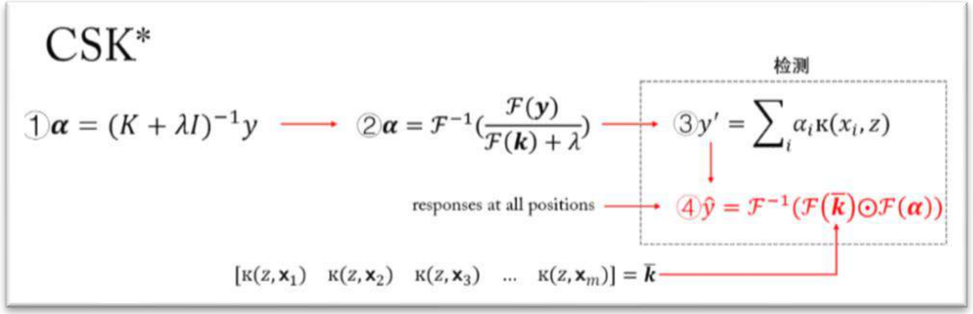
用第一帧训练出的模型

responses at all positions

$$④ f(\boldsymbol{z}) = \boldsymbol{w}^T \boldsymbol{z} = \sum_i \alpha_i \kappa(z, \mathbf{x}_i) = \mathcal{F}^{-1}(\mathcal{F}(\boldsymbol{k}^{xz}) \odot \mathcal{F}(\boldsymbol{\alpha}))$$

$$K^z = \begin{bmatrix} \boxed{\kappa(z,\mathbf{x}_1) & \kappa(z,\mathbf{x}_2) & \kappa(z,\mathbf{x}_3) & \dots & \kappa(z,\mathbf{x}_m)} \\ \kappa(z,\mathbf{x}_m) & \kappa(z,\mathbf{x}_1) & \kappa(z,\mathbf{x}_2) & \dots & \kappa(z,\mathbf{x}_{m-1}) \\ \kappa(z,\mathbf{x}_{m-1}) & \kappa(z,\mathbf{x}_m) & \kappa(z,\mathbf{x}_1) & \dots & \kappa(z,\mathbf{x}_{m-2}) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \kappa(z,\mathbf{x}_2) & \kappa(z,\mathbf{x}_3) & \kappa(z,\mathbf{x}_4) & \dots & \kappa(z,\mathbf{x}_1) \end{bmatrix} = C(\boldsymbol{k}^{xz}) = F diag(\mathcal{F}(\boldsymbol{k}^{xz})) F^H$$

和第10页CSK的推导是完全相同的，核矩阵也是一致的。



KCF 多通道特征

```
function x = get_features(im, features, cell_size, cos_window)

    if features.hog,
        %HOG features, from Piotr's Toolbox
        x = double(fhog(single(im) / 255, cell_size, features.hog_orientations));
        x(:,:,end) = [];  %remove all-zeros channel ("truncation feature")
    end

    if features.gray,
        %gray-level (scalar feature)
        x = double(im) / 255;

        x = x - mean(x(:));
    end

    %process with cosine window if needed
    if ~isempty(cos_window),
        x = bsxfun(@times, x, cos_window);
    end

end
```

```
function H = fhog( I, binSize, nOrients, clip, crop )

if( nargin<2 ), binSize=8; end
if( nargin<3 ), nOrients=9; end
if( nargin<4 ), clip=.2; end
if( nargin<5 ), crop=0; end

softBin = -1; useHog = 2; b = binSize;

[M,O]=gradientMex('gradientMag',I,0,1);

H = gradientMex('gradientHist',M,O,binSize,nOrients,softBin,useHog,clip);

if( crop ), e=mod(size(I),b)<b/2; H=H(2:end-e(1),2:end-e(2),:); end

end
```

- KCF相比CSK的区别主要在于特征的选取不同：CSK以原始像素为特征；KCF以HOG/Gray为特征。

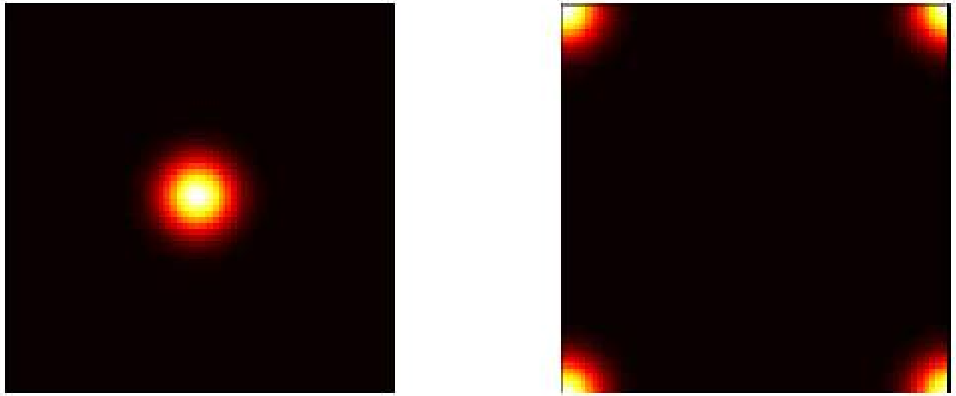
- HOG是3*nOrients+5 =32维的特征，也就是有32个通道。将每个cell的HOG特征并起来，那么一幅图像得到的结果就是一个立体块。

- 假设划分cell的结果是$m \times n$,那么HOG提取结果就是$m \times n \times 31$（去掉了全零的通道）。通过cell的位移来获得样本，这样对应的就是每一通道对应位置的移位，所有样本的第l通道都是由生成图像的第l通道移位获得的。

KCF

```
%store pre-computed cosine window
cos_window = hann(size(yf,1)) * hann(size(yf,2))';
```

①输入的patch（原始像素/提取的特征) 首先会被余弦窗加权处理，消除图像边缘的不连续处
②跟踪区域是目标大小的2.5倍，为了提供额外的负样本
③训练样本由base sample通过循环移位得到
④中心目标的y值为1，其他样本的y值则平缓的衰减到0



(a) (b)

Figure 6: Regression targets y, following a Gaussian function with spatial bandwidth s (white indicates a value of 1, black a value of 0). (a) Placing the peak in the middle will unnecessarily cause the detection output to be shifted by half a window (discussed in Section A.1). (b) Placing the peak at the top-left element (and wrapping around) correctly centers the detection output.

Inputs

- x: training image patch, $m \times n \times c$ 通道数
- y: regression target, Gaussian-shaped, $m \times n$
- z: test image patch, $m \times n \times c$

Output

- responses: detection score for each location, $m \times n$

```
function alphaf = train(x, y, sigma, lambda)
  k = kernel_correlation(x, x, sigma);
  alphaf = fft2(y) ./ (fft2(k) + lambda);
end

function responses = detect(alphaf, x, z, sigma)
  k = kernel_correlation(z, x, sigma);
  responses = real(ifft2(alphaf .* fft2(k)));
end

function k = kernel_correlation(x1, x2, sigma)
  c = ifft2(sum(conj(fft2(x1)) .* fft2(x2), 3));
  d = x1(:)'*x1(:) + x2(:)'*x2(:) - 2 * c;
  k = exp(-1 / sigma^2 * abs(d) / numel(d));
end
```

KCF 程序主体部分

```
for frame = 1:numel(img_files),
    %load image
    im = imread([video_path img_files{frame}]);
    if size(im,3) > 1,
        im = rgb2gray(im);
    end
    if resize_image,
        im = imresize(im, 0.5);
    end

    tic()

    if frame > 1,
        %obtain a subwindow for detection at the position from last
        %frame, and convert to Fourier domain (its size is unchanged)
        patch = get_subwindow(im, pos, window_sz);
        zf = fft2(get_features(patch, features, cell_size, cos_window));

        %calculate response of the classifier at all shifts
        switch kernel.type
        case 'gaussian',
            kzf = gaussian_correlation(zf, model_xf, kernel.sigma);
        case 'polynomial',
            kzf = polynomial_correlation(zf, model_xf, kernel.poly_a, kernel.poly_b);
        case 'linear',
            kzf = linear_correlation(zf, model_xf);
        end
        response = real(ifft2(model_alphaf .* kzf));  %equation for fast detection

        %target location is at the maximum response. we must take into
        %account the fact that, if the target doesn't move, the peak
        %will appear at the top-left corner, not at the center (this is
        %discussed in the paper). the responses wrap around cyclically.
        [vert_delta, horiz_delta] = find(response == max(response(:)), 1);
        if vert_delta > size(zf,1) / 2,  %wrap around to negative half-space of vertical axis
            vert_delta = vert_delta - size(zf,1);
        end
        if horiz_delta > size(zf,2) / 2,  %same for horizontal axis
            horiz_delta = horiz_delta - size(zf,2);
        end
        pos = pos + cell_size * [vert_delta - 1, horiz_delta - 1];
    end
```

论文中响应的表达形式 $\hat{\mathbf{f}}(\mathbf{z}) = \hat{\mathbf{k}}^{\mathbf{xz}} \odot \hat{\alpha}.$

用训练好的滤波器更新位置

$f(z) = \mathcal{F}^{-1}(\mathcal{F}(k^{xz}) \odot \mathcal{F}(\alpha))$

找到响应最大的位置

相比CSK的区别：引入了HOG特征

```
    %obtain a subwindow for training at newly estimated target position
    patch = get_subwindow(im, pos, window_sz);
    xf = fft2(get_features(patch, features, cell_size, cos_window));

    %Kernel Ridge Regression, calculate alphas (in Fourier domain)
    switch kernel.type
    case 'gaussian',
        kf = gaussian_correlation(xf, xf, kernel.sigma);
    case 'polynomial',
        kf = polynomial_correlation(xf, xf, kernel.poly_a, kernel.poly_b);
    case 'linear',
        kf = linear_correlation(xf, xf);
    end
    alphaf = yf ./ (kf + lambda);   %equation for fast training
```

通过第一帧训练滤波器

$$\hat{\alpha} = \frac{\hat{\mathbf{y}}}{\hat{\mathbf{k}}^{\mathbf{xx}} + \lambda}$$

```
    if frame == 1,  %first frame, train with a single image
        model_alphaf = alphaf;
        model_xf = xf;
    else
        %subsequent frames, interpolate model
        model_alphaf = (1 - interp_factor) * model_alphaf + interp_factor * alphaf;
        model_xf = (1 - interp_factor) * model_xf + interp_factor * xf;
    end
```

滤波器更新

```
    %save position and timing
    positions(frame,:) = pos;
    time = time + toc();

    %visualization
    if show_visualization,
        box = [pos([2,1]) - target_sz([2,1])/2, target_sz([2,1])];
        stop = update_visualization(frame, box);
        if stop, break, end  %user pressed Esc, stop early

        drawnow
        pause(0.05)  %uncomment to run slower
    end

end
```

KCF 程序主要流程

- Get Subwindow
- Get features(hog/gray)
- κ^{gauss}/ $\kappa^{polynomial}$/ κ^{linear}
- alphaf($\hat{\boldsymbol{\alpha}}$) $= \frac{\mathcal{F}(y)}{\mathcal{F}(k^{xx})+\lambda}$
- model_alphaf ← alphaf;
- model_xf ← xf;

1^{st} frame

- Get Subwindow
- Get features(hog/gray)
- κ^{gauss}/ $\kappa^{polynomial}$/ κ^{linear}
- $response(\hat{y}) = \mathcal{F}(k^{xz}) \odot \mathcal{F}(\alpha)$
- Find max response
- Change position(*pos*)
- Get Subwindow(new subwindow according to the *pos*)
- Get features(hog/gray)
- κ^{gauss}/ $\kappa^{polynomial}$/ κ^{linear}
- alphaf($\hat{\boldsymbol{\alpha}}$) $= \frac{\mathcal{F}(y)}{\mathcal{F}(k^{xx})+\lambda}$
- model_alphaf←(1 - interp_factor) * model_alphaf + interp_factor * alphaf
- model_xf←(1 - interp_factor) * model_xf + interp_factor *xf

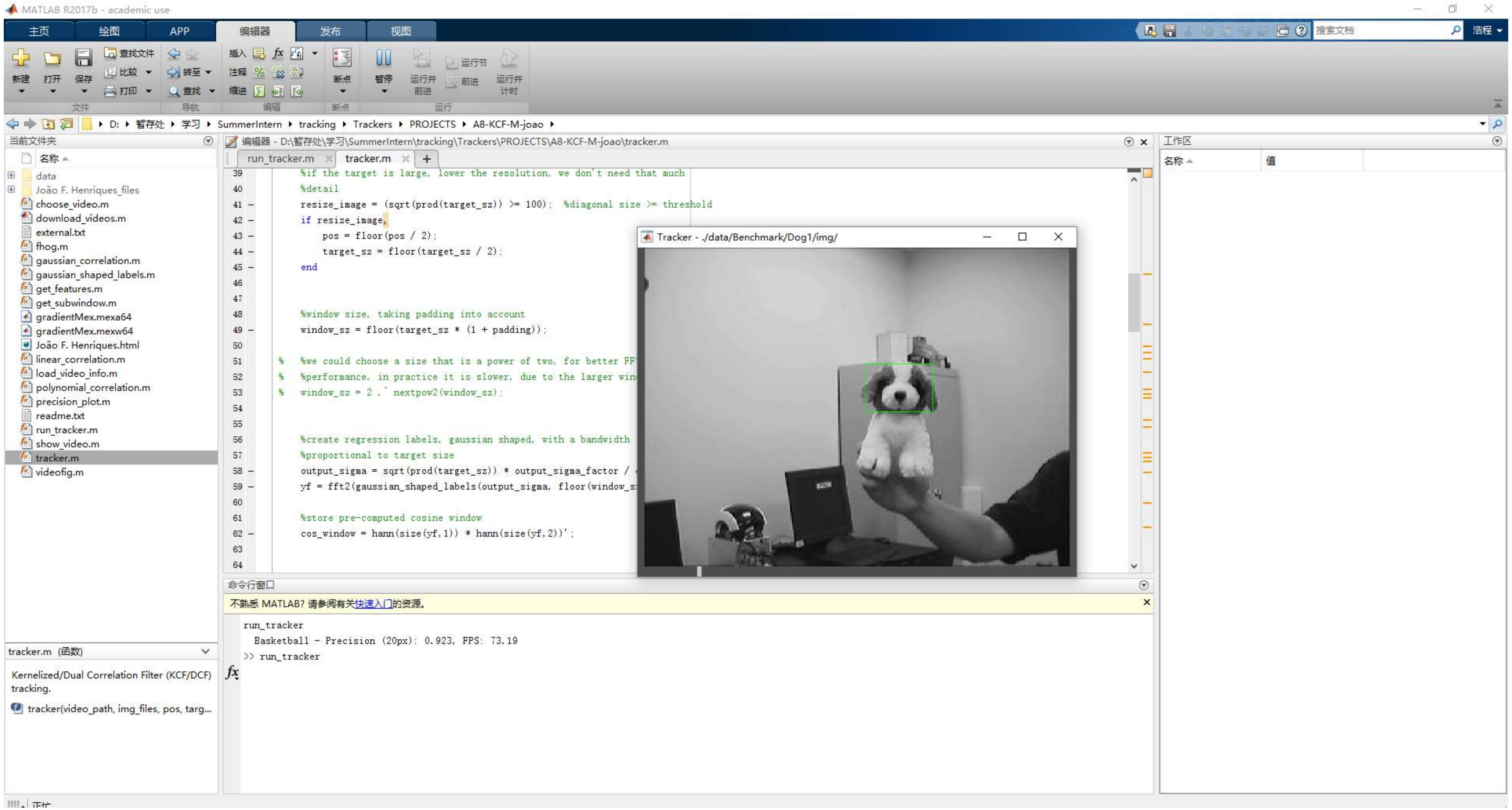
2^{nd},3^{rd},4^{th} ……frame

KCF 程序主要流程

- Get Subwindow — 获取子窗口
- Get features(hog/gray) — 获取子窗口特征
- κ^{gauss}/ $\kappa^{polynomial}$/ κ^{linear} — 计算核函数
- alphaf($\hat{\boldsymbol{\alpha}}$) $= \frac{\mathcal{F}(y)}{\mathcal{F}(k^{xx})+\lambda}$ — 通过核函数和已知响应计算α
- model_alphaf ← alphaf; — 给model_alphaf赋值
- model_xf ← xf; — 给model_xf赋值

- Get Subwindow — 再次获取子窗口
- Get features(hog/gray) — 获取子窗口特征
- κ^{gauss}/ $\kappa^{polynomial}$/ κ^{linear} — 计算核函数
- $response(\hat{y}) = \mathcal{F}(k^{xz}) \odot \mathcal{F}(\alpha)$ — 通过前一帧训练出来的α计算所有位置的响应y
- Find max response — 找到最大的响应
- Change position(*pos*) — 根据最大响应获取新位置
- Get Subwindow(new subwindow according to the *pos*) — 根据新位置重新获取子窗口
- Get features(hog/gray) — 获取子窗口特征
- κ^{gauss}/ $\kappa^{polynomial}$/ κ^{linear} — 计算核函数
- alphaf($\hat{\boldsymbol{\alpha}}$) $= \frac{\mathcal{F}(y)}{\mathcal{F}(k^{xx})+\lambda}$ — 更新α
- model_alphaf←(1 - interp_factor) * model_alphaf+ interp_factor * alphaf — 加权更新model_alphaf
- model_xf←(1 - interp_factor) * model_xf + interp_factor *xf — 加权更新model_xf

KCF 运行情况



DSST 多通道特征

①$F = \mathcal{F}(f)$ ← Input image ②$H = \mathcal{F}(h)$ ← filter ③$g = f\ h$

④$\varepsilon = \left|\sum_{l=1}^{d} f^l h^l - g\right|^2 + \lambda \sum_{l=1}^{d} \left|h^l\right|^2$ l表示特征的第l维（通道）

⑤$H^l = \dfrac{\boxed{G^* F^l}}{\boxed{\sum_{k=1}^{d} (F^k)^* F^k} + \lambda}$

A^l ⑥$A_t^l = (1-\eta)A_{t-1}^l + \eta G^* F_t^l$

B ⑦$B_t = (1-\eta)B_{t-1} + \eta \sum_{k=1}^{d} (F^k)^* F^k$

⑧$Y_t = \dfrac{\sum_{l=1}^{d} (A_{t-1}^l)^* Z_t^l}{B_{t-1} + \lambda}$ 位置估计和尺度估计都是利用⑧

DSST 程序主体部分

```
for frame = 1:num_frames,
    %load image
    im = imread([video_path img_files{frame}]);

    tic;

    if frame > 1

        % extract the test sample feature map for the translation filter
        xt = get_translation_sample(im, pos, sz, currentScaleFactor, cos_window);

        % calculate the correlation response of the translation filter
        xtf = fft2(xt);
        response = real(ifft2(sum(hf_num .* xtf, 3) ./ (hf_den + lambda)));

        % find the maximum translation response
        [row, col] = find(response == max(response(:)), 1);

        % update the position
        pos = pos + round((-sz/2 + [row, col]) * currentScaleFactor);
```

用训练好的位置滤波器更新位置

```
        % extract the test sample feature map for the scale filter
        xs = get_scale_sample(im, pos, base_target_sz, currentScaleFactor * scaleFactors, scale_window, scale_model_sz);

        % calculate the correlation response of the scale filter
        xsf = fft(xs,[],2);
        scale_response = real(ifft(sum(sf_num .* xsf, 1) ./ (sf_den + lambda)));

        % find the maximum scale response
        recovered_scale = find(scale_response == max(scale_response(:)), 1);

        % update the scale
        currentScaleFactor = currentScaleFactor * scaleFactors(recovered_scale);
        if currentScaleFactor < min_scale_factor
            currentScaleFactor = min_scale_factor;
        elseif currentScaleFactor > max_scale_factor
            currentScaleFactor = max_scale_factor;
        end
    end
```

用训练好的尺度滤波器更新尺度

```
    % extract the training sample feature map for the translation filter
    xl = get_translation_sample(im, pos, sz, currentScaleFactor, cos_window);

    % calculate the translation filter update
    xlf = fft2(xl);
    new_hf_num = bsxfun(@times, yf, conj(xlf));
    new_hf_den = sum(xlf .* conj(xlf), 3);

    % extract the training sample feature map for the scale filter
    xs = get_scale_sample(im, pos, base_target_sz, currentScaleFactor * scaleFactors, scale_window, scale_model_sz);
```

$\overline{G}$ F_t^l $\overline{G}F_t^l$

F_t^k $\overline{F_t^k}$ $\sum_{k=1}^{d}\overline{F_t^k}F_t^k$

通过第一帧训练位置滤波器

```
    % calculate the scale filter update
    xsf = fft(xs,[],2);
    new_sf_num = bsxfun(@times, ysf, conj(xsf));
    new_sf_den = sum(xsf .* conj(xsf), 1);

    if frame == 1
        % first frame, train with a single image
        hf_den = new_hf_den;
        hf_num = new_hf_num;

        sf_den = new_sf_den;
        sf_num = new_sf_num;
    else
        % subsequent frames, update the model
        hf_den = (1 - learning_rate) * hf_den + learning_rate * new_hf_den;
        hf_num = (1 - learning_rate) * hf_num + learning_rate * new_hf_num;
        sf_den = (1 - learning_rate) * sf_den + learning_rate * new_sf_den;
        sf_num = (1 - learning_rate) * sf_num + learning_rate * new_sf_num;
    end
```

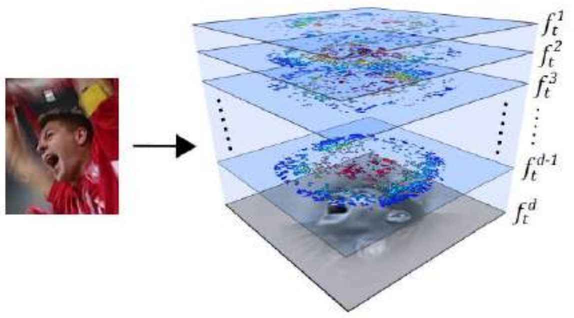
$\overline{G}$ F_t^l $\overline{G}F_t^l$

F_t^k $\overline{F_t^k}$ $\sum_{k=1}^{d}\overline{F_t^k}F_t^k$

滤波器更新

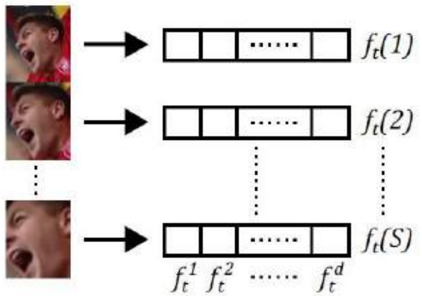
```
    % calculate the new target size
    target_sz = floor(base_target_sz * currentScaleFactor);

    %save position
    positions(frame,:) = [pos target_sz];

    time = time + toc;


    %visualization
    if visualization == 1
        rect_position = [pos([2,1]) - target_sz([2,1])/2, target_sz([2,1])];
        if frame == 1,  %first frame, create GUI
            figure('Name',['Tracker - ' video_path]);
            im_handle = imshow(uint8(im), 'Border','tight', 'InitialMag', 100 + 100 * (length(im) < 500));
            rect_handle = rectangle('Position',rect_position, 'EdgeColor','g');
            text_handle = text(10, 10, int2str(frame));
            set(text_handle, 'color', [0 1 1]);
        else
            try  %subsequent frames, update GUI
                set(im_handle, 'CData', im)
                set(rect_handle, 'Position', rect_position)
                set(text_handle, 'string', int2str(frame));
            catch
                return
            end
        end

        drawnow
        pause
```

DSST



(a) Translation filter sample.

位置过滤样本（d维）



(b) Scale filter sample.

尺度过滤样本（d维）

初始化目标的预期高斯输出G

以目标旧位置为中心，采集一个尺寸为目标大小2倍大小的样本z

样本中每个像素点计算28维融合特征（1维原始灰度特征+27维fhog特征），乘以余弦窗作为输入z

$$Y_t = \frac{\sum_{l=1}^{d}(A_{t-1}^{l})^{*}Z_t^{l}}{B_{t-1}+\lambda}$$

初始化目标的预期高斯输出G

以目标新位置为中心，提取33中不同尺度下的样本z

把每个样本resize成固定尺寸，分别提取31维fhog特征，每个样本的所有fhog特征再串联成一个特征向量构成33层金字塔特征，乘以1维hann窗后作为测试输出z

$$Y_t = \frac{\sum_{l=1}^{d}(A_{t-1}^{l})^{*}Z_t^{l}}{B_{t-1}+\lambda}$$

求max(y)得到目标新位置/新尺度

Input:

Image I_t.
Previous target position p_{t-1} and scale s_{t-1}.
Translation model $A_{t-1,\text{trans}}$, $B_{t-1,\text{trans}}$.
Scale model $A_{t-1,\text{scale}}$, $B_{t-1,\text{scale}}$.

Output:

Estimated target position p_t and scale s_t.
Updated translation model $A_{t,\text{trans}}$, $B_{t,\text{trans}}$.
Updated scale model $A_{t,\text{scale}}$, $B_{t,\text{scale}}$.

Translation estimation:

1: Extract sample $z_{t,\text{trans}}$ from I_t at p_{t-1} and s_{t-1}.
2: Compute correlation scores $y_{t,\text{trans}}$ using (4).
3: Set p_t to the target position that maximizes $y_{t,\text{trans}}$.

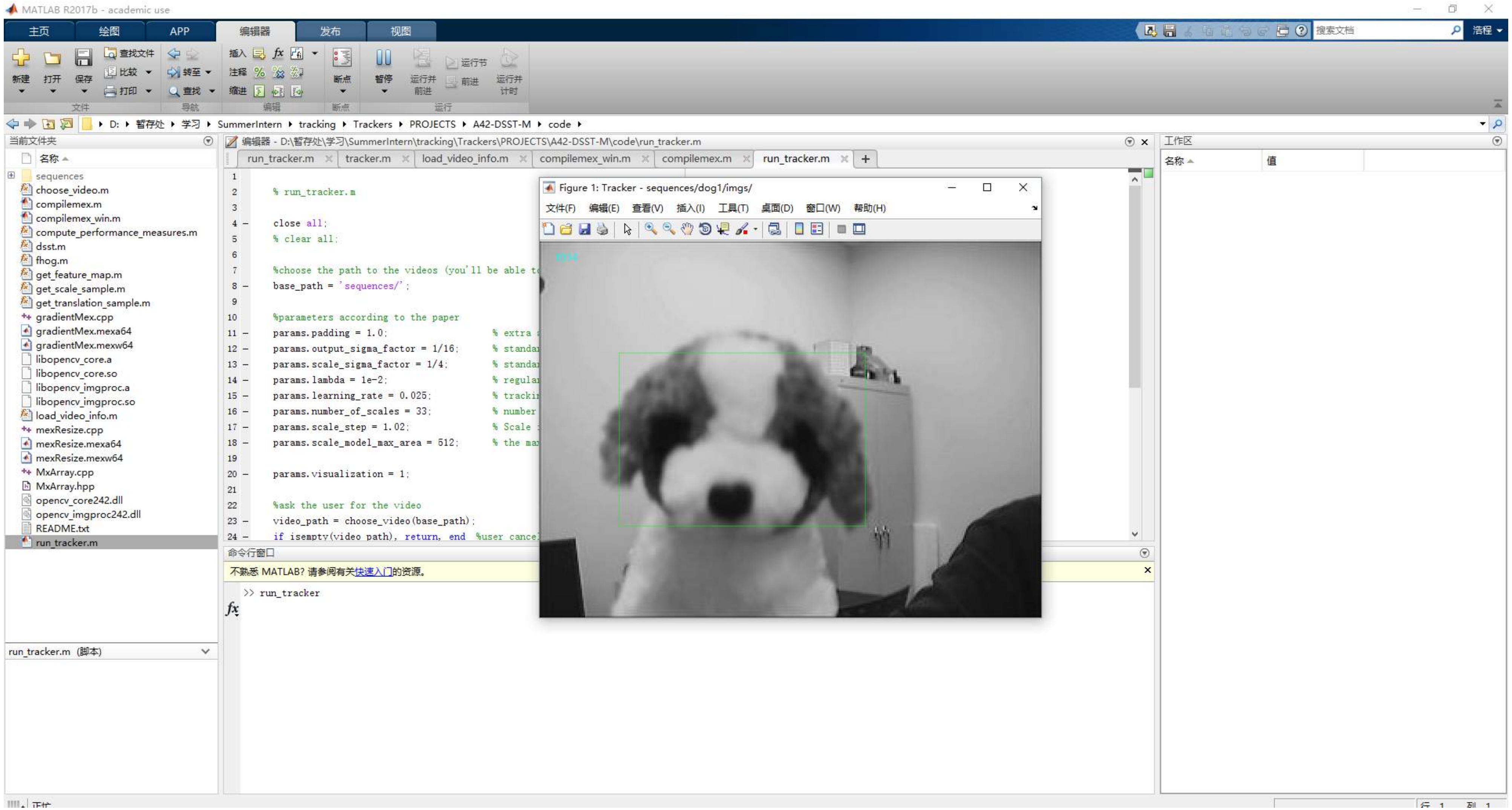
Scale estimation:

4: Extract sample $z_{t,\text{scale}}$ from I_t at p_t and s_{t-1}.
5: Compute correlation scores $y_{t,\text{scale}}$ using (4).
6: Set s_t to the target scale that maximizes $y_{t,\text{scale}}$.

Model update:

7: Extract samples $f_{t,\text{trans}}$ and $f_{t,\text{scale}}$ from I_t at p_t and s_t.
8: Update the translation model $A_{t,\text{trans}}$, $B_{t,\text{trans}}$ using (3).
9: Update the scale model $A_{t,\text{scale}}$, $B_{t,\text{scale}}$ using (3).

DSST运行情况



MATLAB R2017b - academic use
主页
绘图
APP
编辑器
发布
视图
编辑器 - D:\暂存处\学习\SummerIntern\tracking\Trackers\PROJECTS\A42-DSST-M\code\run_tracker.m
Figure 1: Tracker - sequences/dog1/imgs/
% run_tracker.m
close all;
% clear all;
%choose the path to the videos (you'll be able t
base_path = 'sequences/';
%parameters according to the paper
params.padding = 1.0;
params.output_sigma_factor = 1/16;
params.scale_sigma_factor = 1/4;
params.lambda = 1e-2;
params.learning_rate = 0.025;
params.number_of_scales = 33;
params.scale_step = 1.02;
params.scale_model_max_area = 512;
params.visualization = 1;
%ask the user for the video
video_path = choose_video(base_path);
if isempty(video_path), return, end %user cance
命令行窗口
不熟悉 MATLAB? 请参阅有关快速入门的资源。
>> run_tracker
工作区

Correlation Filter 特点

MOSSE	=	相关滤波	+	单通道灰度特征		
CSK	=	MOSSE	+	RBF/高斯核函数	+	循环移位稠密采样
DCF	=	CSK(线性核函数)	+	多通道HOG特征		
KCF	=	CSK	+	多通道HOG特征		
DSST	=	MOSSE	+	尺度更新		

MDNet

“Training CNNs is even more difficult since the same kind of objects can be considered as a target in a sequence and as a background object in another. Due to such variations and inconsistencies across sequences, we believe that the ordinary learning methods based on the standard classification task are not appropriate, and another approach to capture sequence-independent information should be incorporated for better representations for tracking.”

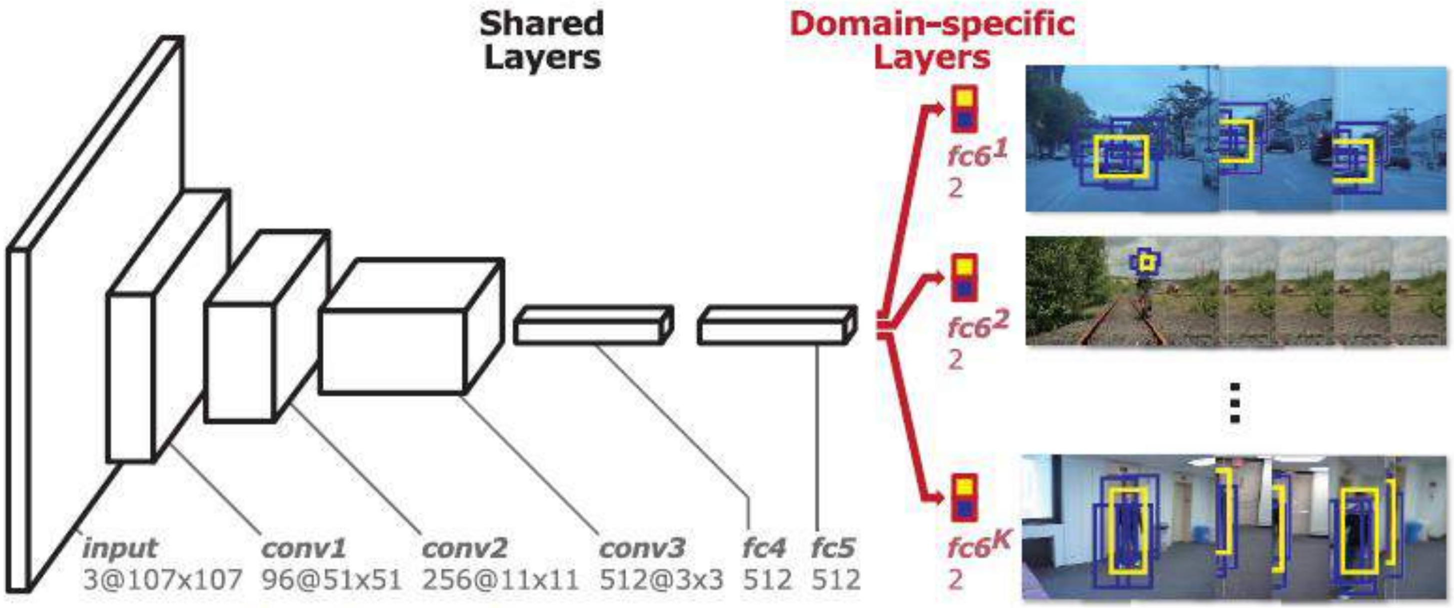


Figure 1: The architecture of our Multi-Domain Network, which consists of shared layers and K branches of domain-specific layers. Yellow and blue bounding boxes denote the positive and negative samples in each domain, respectively.

MDNet

Bounding box regression

N target candidates $\mathbf{x}^1, \dots, \mathbf{x}^N$ are sampled around the previous target state. The optimal target state $\mathbf{x}^*$ is given by finding the example with the maximum positive score.

$$\mathbf{x}^* = \arg\max_{\mathbf{x}^i} f^+(\mathbf{x}^i)$$

Positive score

If the quantity of samples exceeds capacity, then discard the old ones.

Full connected layers

Algorithm 1 Online tracking algorithm

Input : Pretrained CNN filters $\{\mathbf{w}_1, \dots, \mathbf{w}_5\}$
Initial target state $\mathbf{x}_1$
Output: Estimated target states $\mathbf{x}_t^*$

```
Randomly initialize the last layer w_6.
Train a bounding box regression model.
Draw positive samples S_1^+ and negative samples S_1^-.
Update {w_4, w_5, w_6} using S_1^+ and S_1^-;
T_s ← {1} and T_l ← {1}.
repeat
    Draw target candidate samples x_t^i.
    Find the optimal target state x_t^* by Eq. (1).
    if f^+(x_t^*) > 0.5 then
        Draw training samples S_t^+ and S_t^-.
        T_s ← T_s ∪ {t}, T_l ← T_l ∪ {t}.
        if |T_s| > τ_s then T_s ← T_s \ {min_{v∈T_s} v}.
        if |T_l| > τ_l then T_l ← T_l \ {min_{v∈T_l} v}.
        Adjust x_t^* using bounding box regression.
    if f^+(x_t^*) < 0.5 then
        Update {w_4, w_5, w_6} using S_{v∈T_s}^+ and S_{v∈T_s}^-.
    else if t mod 10 = 0 then
        Update {w_4, w_5, w_6} using S_{v∈T_l}^+ and S_{v∈T_s}^-.
until end of sequence
```

MDNet 程序主体部分

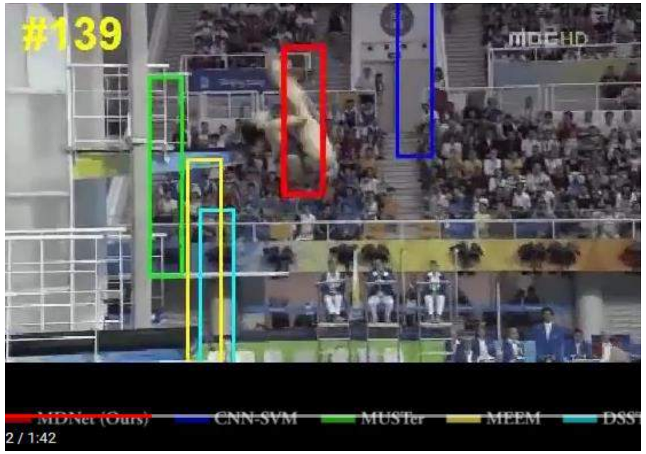
```
%% Main loop
for To = 2:nFrames;
    fprintf('Processing frame %d/%d... ', To, nFrames);

    img = imread(images{To});
    if(size(img,3)==1), img = cat(3,img,img,img); end

    spf = tic;
    %% Estimation
    % draw target candidates
    samples = gen_samples('gaussian', targetLoc, opts.nSamples, opts, trans_f, scale_f);
    feat_conv = mdnet_features_convX(net_conv, img, samples, opts);

    % evaluate the candidates
    feat_fc = mdnet_features_fcX(net_fc, feat_conv, opts);
    feat_fc = squeeze(feat_fc)';
    [scores,idx] = sort(feat_fc(:,2),'descend');
    target_score = mean(scores(1:5));
    targetLoc = round(mean(samples(idx(1:5),:)));

    % final target
    result(To,:) = targetLoc;

    % extend search space in case of failure
    if(target_score<0)
        trans_f = min(1.5, 1.1*trans_f);
    else
        trans_f = opts.trans_f;
    end

    % bbox regression
    if(opts.bbreg && target_score>0)
        X_ = permute(gather(feat_conv(:,:,:,idx(1:5))),[4,3,1,2]);
        X_ = X_(:,:);
        bbox_ = samples(idx(1:5),:);
        pred_boxes = predict_bbox_regressor(bbox_reg.model, X_, bbox_);
        result(To,:) = round(mean(pred_boxes,1));
    end

    %% Prepare training data
    if(target_score>0)
        pos_examples = gen_samples('gaussian', targetLoc, opts.nPos_update*2, opts, 0.1, 5);
        r = overlap_ratio(pos_examples,targetLoc);
        pos_examples = pos_examples(r>opts.posThr_update,:);
        pos_examples = pos_examples(randsample(end,min(opts.nPos_update,end)),:);

        neg_examples = gen_samples('uniform', targetLoc, opts.nNeg_update*2, opts, 2, 5);
        r = overlap_ratio(neg_examples,targetLoc);
        neg_examples = neg_examples(r<opts.negThr_update,:);
        neg_examples = neg_examples(randsample(end,min(opts.nNeg_update,end)),:);
```

Bounding box regression

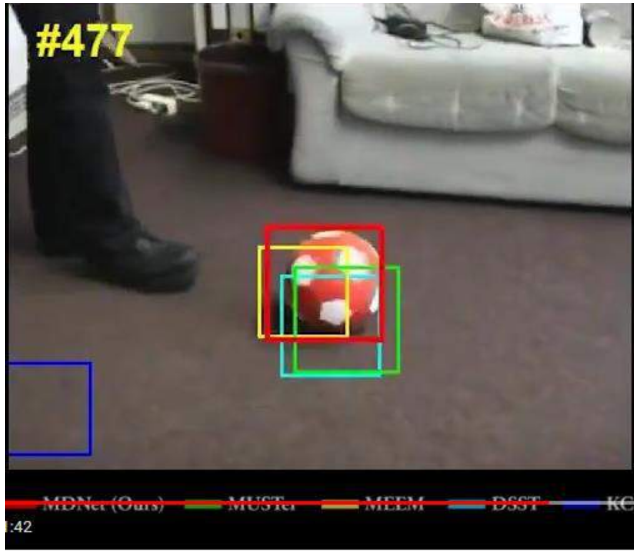
MDNet 程序主体部分

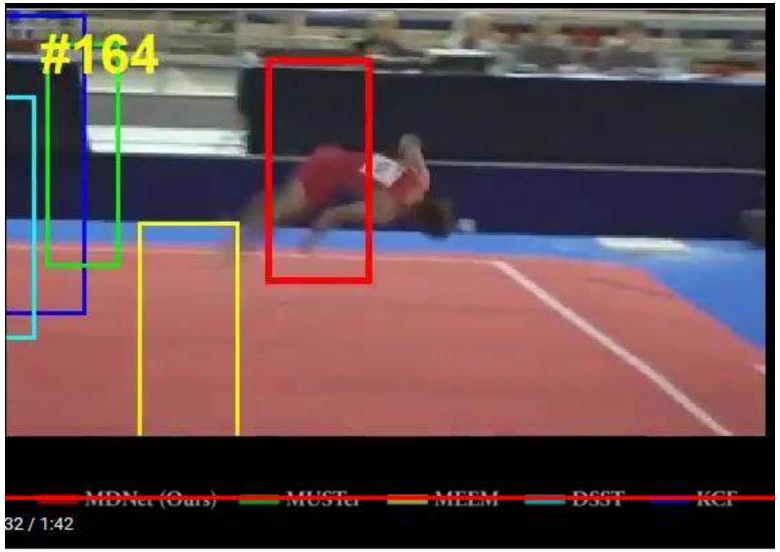
```
        examples = [pos_examples; neg_examples];
        pos_idx = 1:size(pos_examples,1);
        neg_idx = (1:size(neg_examples,1)) + size(pos_examples,1);

        feat_conv = mdnet_features_convX(net_conv, img, examples, opts);
        total_pos_data{To} = feat_conv(:,:,:,pos_idx);
        total_neg_data{To} = feat_conv(:,:,:,neg_idx);

        success_frames = [success_frames, To];
        if(numel(success_frames)>opts.nFrames_long)
            total_pos_data{success_frames(end-opts.nFrames_long)} = single([]);
        end
        if(numel(success_frames)>opts.nFrames_short)
            total_neg_data{success_frames(end-opts.nFrames_short)} = single([]);
        end
    else
        total_pos_data{To} = single([]);
        total_neg_data{To} = single([]);
    end

    %% Network update
    if((mod(To,opts.update_interval)==0 || target_score<0) && To~=nFrames)
        if (target_score<0) % short-term update
            pos_data = cell2mat(total_pos_data(success_frames(max(1,end-opts.nFrames_short+1):end)));
        else % long-term update
            pos_data = cell2mat(total_pos_data(success_frames(max(1,end-opts.nFrames_long+1):end)));
        end
        neg_data = cell2mat(total_neg_data(success_frames(max(1,end-opts.nFrames_short+1):end)));

%         fprintf('\n');
        [net_fc] = mdnet_finetune_hnm(net_fc,pos_data,neg_data,opts,...
            'maxiter',opts.maxiter_update,'learningRate',opts.learningRate_update);
    end

    spf = toc(spf);
    fprintf('%f seconds\n',spf);

    %% Display
    if display
        hc = get(gca, 'Children'); delete(hc(1:end-1));
        set(hd,'cdata',img); hold on;

        rectangle('Position', result(To,:), 'EdgeColor', [1 0 0], 'Linewidth', 3);
        set(gca,'position',[0 0 1 1]);

        text(10,10,num2str(To),'Color','y', 'HorizontalAlignment', 'left', 'FontWeight','bold', 'FontSize', 30);
        hold off;
        drawnow;
    end
end
```

MDNet 运行情况

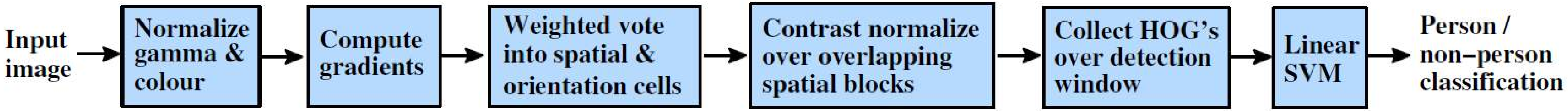


1. KCF和DSST对于形变程度大的目标——跳水、体操运动员（或者说帧与帧之间变化较大），跟踪效果不是很好；
2. KCF和DSST等跟踪器在跟丢目标之后就无法再继续跟踪了；
3. DSST在human5 sequence中也发生了漂移，跟丢后也没能恢复跟踪；
4. 左上角bolt sequence中除了MDNet的其他跟踪器都跟错了目标，其原因很有可能是有其他与target非常相似的对象对跟踪器造成了干扰；
5. MDNet跟踪框可以根据目标尺度的变化改变大小





HOG single channel



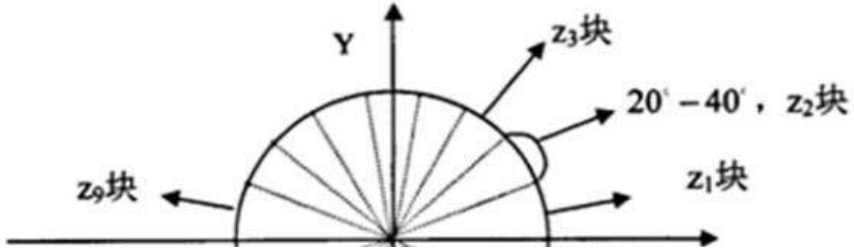
	$(x, y-1)$	
$(x-1, y)$	(x, y)	$(x+1, y)$
	$(x, y+1)$	

$$G_x(x,y) = H(x+1,y) - H(x-1,y)$$

$$G_y(x,y) = H(x,y+1) - H(x,y-1)$$

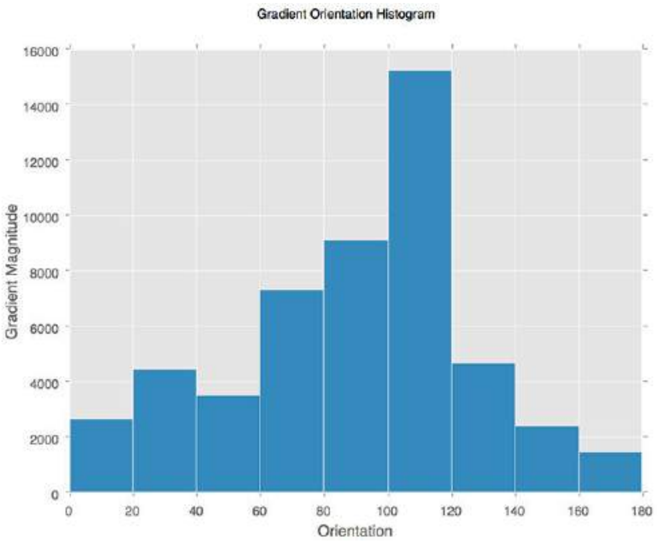
$$magnitude: G(x,y) = \sqrt{G_x(x,y)^2 + G_y(x,y)^2}$$

$$direction: \theta(x,y) = \tan^{-1}\frac{G_y(x,y)}{G_x(x,y)}$$

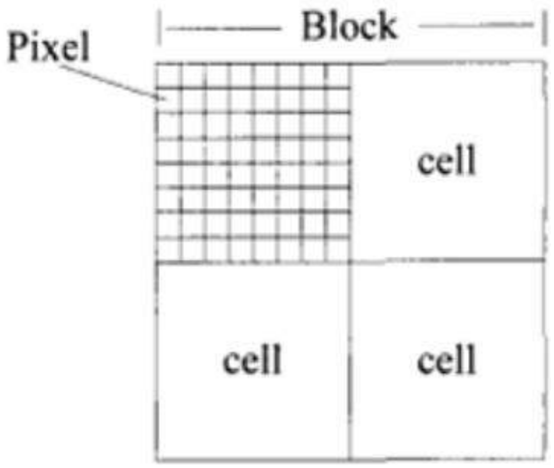


e.g. magnitude = 20, direction= 65°, then give 60° 20*15/20=15 votes, give 80° 20*5/20=5 votes

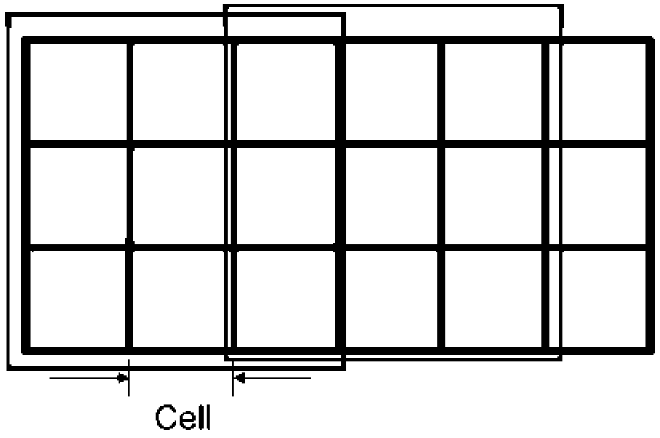
HOG single channel



every cell has a histogram with 9 bins
so we can get a 9-d vector for every cell

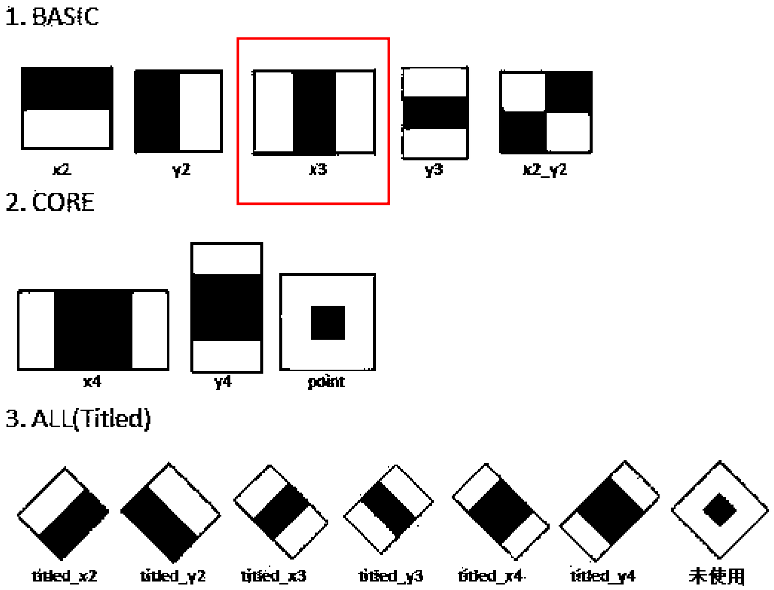


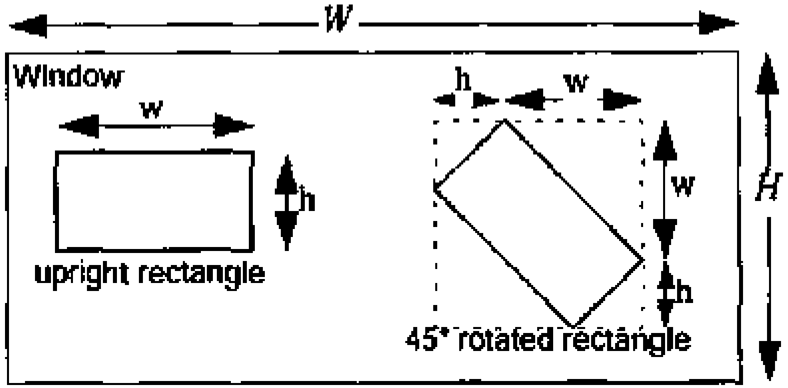
every block has 4 cells
so we can get a 4*9-d vector for every block

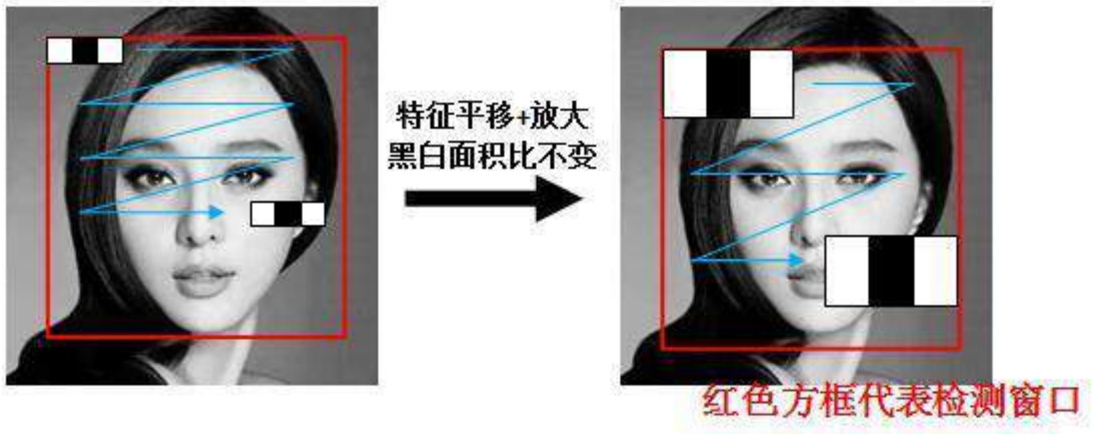


Assume the size of an image, a cell and a block are 64*128(pixel), 8*8(pixel) and 2*2 (cell) respectively. The step of block is 1 (cell). So the image has a feature vector with (64/8-1)*(128/8-1)*4*9=3780 dimensions.

Haar







The dimension of X3 feature in vertical direction

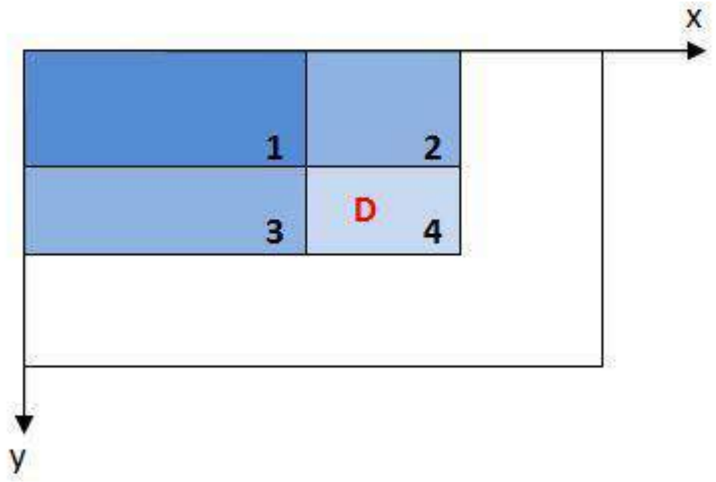
$$(H-h+1)+(H-2h+1)+(H-3h+1)+\ldots\ldots+(H-Y*h+1)=Y[H+1-h(1+Y)/2]$$

Haar

$$featureValue(x) = weight_{all} \times \sum_{pixel \in all} pixel + weight_{black} \times \sum_{pixel \in black} pixel$$

$$sum(D) = sum(x_4, y_4) - sum(x_3, y_3) - sum(x_2, y_2) + sum(x_1, y_1)$$

Integral image

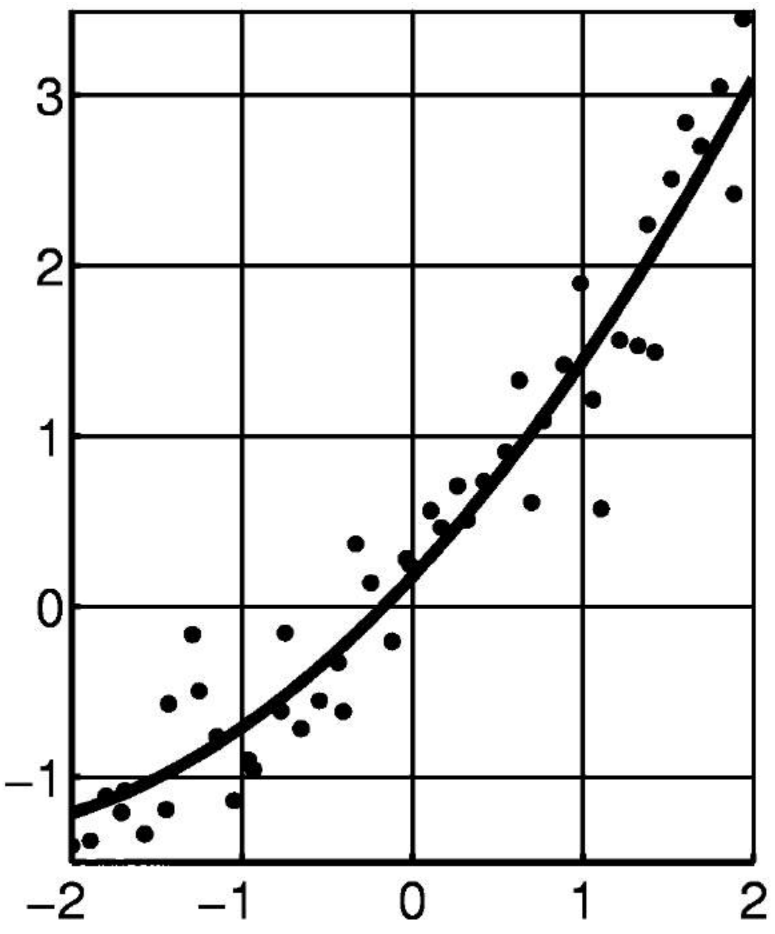


Least squares method

$$①y_i = \sum_{j=1}^{n} X_{ij} w_j \ (i = 1,2,3, \dots, m)$$

$$②\boldsymbol{y} = X \cdot \boldsymbol{w}$$

$$X = \begin{bmatrix} X_{11} & X_{12} & \dots & X_{1n} \\ X_{21} & X_{22} & \dots & X_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ X_{m1} & X_{m1} & \dots & X_{mn} \end{bmatrix}, \qquad w = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix}, \qquad y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$



$$③\hat{\boldsymbol{w}} = \arg\min_{w} \|X \cdot \boldsymbol{w} - \boldsymbol{y}\|^2$$

$$④\hat{\boldsymbol{w}} = (X^T X)^{-1} X^T \boldsymbol{y}$$

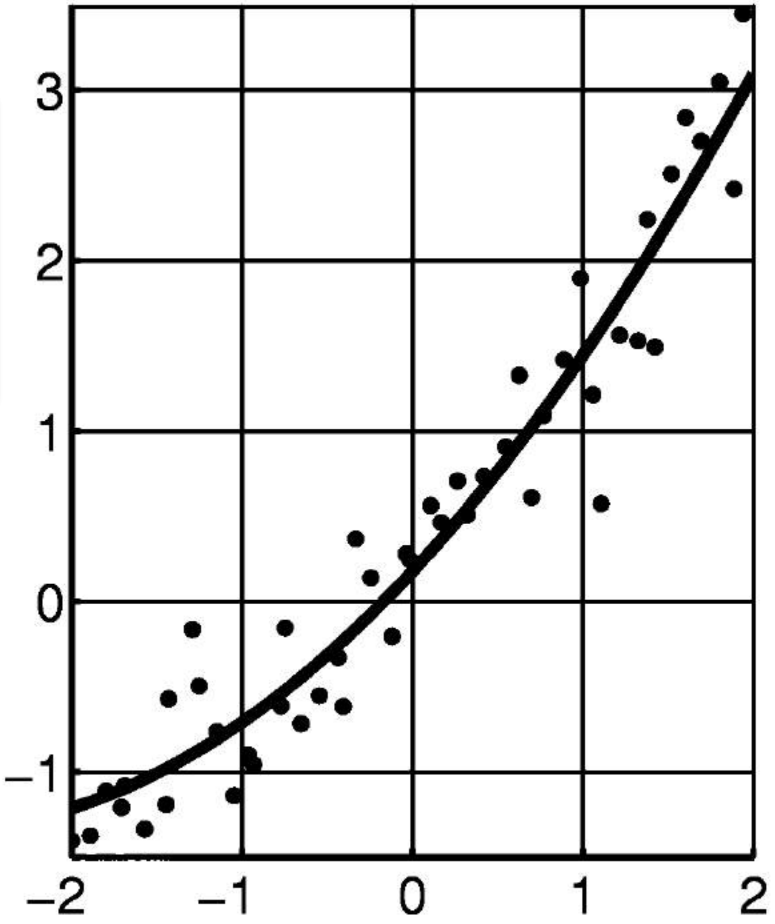
当$X^T X$为满秩矩阵或正定矩阵时，可得④式

Ridge Regression

然而, 现实任务中 $\mathbf{X}^{\mathrm{T}}\mathbf{X}$ 往往不是满秩矩阵. 例如在许多任务中我们会遇到大量的变量, 其数目甚至超过样例数, 导致 $\mathbf{X}$ 的列数多于行数, $\mathbf{X}^{\mathrm{T}}\mathbf{X}$ 显然不满秩. 此时可解出多个 $\hat{\boldsymbol{w}}$, 它们都能使均方误差最小化. 选择哪一个解作为输出, 将由学习算法的归纳偏好决定, 常见的做法是引入正则化 (regularization)项.

$$\hat{\boldsymbol{w}} = (X^T X)^{-1} X^T \boldsymbol{y}$$

$$S(\boldsymbol{w}) = \|X \cdot \boldsymbol{w} - y\|^2 \rightarrow S(\boldsymbol{w}) = \|X \cdot \boldsymbol{w} - y\|^2 + \lambda \|\boldsymbol{w}\|^2$$



Kernel trick

例如：

$$\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}, \mathbf{x}\text{的高次变换为}\varphi(\mathbf{x}) = \begin{pmatrix} x_1x_1 \\ x_1x_2 \\ x_1x_3 \\ x_2x_1 \\ x_2x_2 \\ x_2x_3 \\ x_3x_1 \\ x_3x_2 \\ x_3x_3 \end{pmatrix};\ \boldsymbol{z} = \begin{pmatrix} z_1 \\ z_2 \\ z_3 \end{pmatrix}, \boldsymbol{z}\text{的高次变换为}\varphi(\boldsymbol{z}) = \begin{pmatrix} z_1z_1 \\ z_1z_2 \\ z_1z_3 \\ z_2z_1 \\ z_2z_2 \\ z_2z_3 \\ z_3z_1 \\ z_3z_2 \\ z_3z_3 \end{pmatrix}$$

由于特征空间维数可能很高，直接计算$\langle\varphi(\mathbf{x}), \varphi(\boldsymbol{z})\rangle$通常是困难的；
而有了核函数$\kappa(\mathbf{x}, \boldsymbol{z}) = (\langle\mathbf{x}, \boldsymbol{z}\rangle)^2$，在特征空间的内积等于他们在原始样本空间中通过核函数计算的结果，
跳过了复杂的中间步骤，计算量则大大减少，二者最终结果相同。

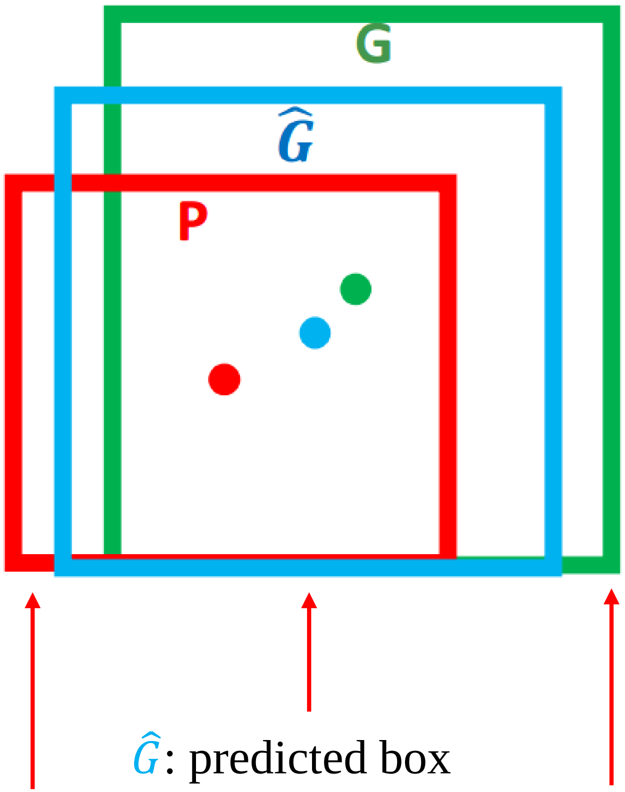
核矩阵　　样本

$$K = \begin{bmatrix} \kappa(\mathbf{x}_1, \mathbf{x}_1) & \cdots & \kappa(\mathbf{x}_1, \mathbf{x}_j) & \cdots & \kappa(\mathbf{x}_1, \mathbf{x}_m) \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ \kappa(\mathbf{x}_i, \mathbf{x}_1) & \cdots & \kappa(\mathbf{x}_i, \mathbf{x}_j) & \cdots & \kappa(\mathbf{x}_i, \mathbf{x}_m) \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ \kappa(\mathbf{x}_m, \mathbf{x}_1) & \cdots & \kappa(\mathbf{x}_m, \mathbf{x}_j) & \cdots & \kappa(\mathbf{x}_m, \mathbf{x}_m) \end{bmatrix}$$

表 6.1　常用核函数

名称	表达式	参数
线性核	$\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) = \boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{x}_j$	
多项式核	$\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) = (\boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{x}_j)^d$	$d \geqslant 1$ 为多项式的次数
高斯核	$\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp\left(-\frac{\lVert\boldsymbol{x}_i - \boldsymbol{x}_j\rVert^2}{2\sigma^2}\right)$	$\sigma > 0$ 为高斯核的带宽(width)
拉普拉斯核	$\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp\left(-\frac{\lVert\boldsymbol{x}_i - \boldsymbol{x}_j\rVert}{\sigma}\right)$	$\sigma > 0$
Sigmoid 核	$\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) = \tanh(\beta\boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{x}_j + \theta)$	tanh 为双曲正切函数，$\beta > 0$, $\theta < 0$

Bounding Box regression



$\widehat{G}$: predicted box

P: original proposal

G: ground truth

给定(P_x,P_y,P_w,P_h)，寻找一个映射f使得$f(P_x,P_y,P_w,P_h)=(\widehat{G_x},\widehat{G_y},\widehat{G_g},\widehat{G_h})$,
并且有$(G_x,G_y,G_w,G_h)\approx(\widehat{G_x},\widehat{G_y},\widehat{G_g},\widehat{G_h})$

平移 $(\Delta x,\Delta y),\Delta x=P_w d_x(P),\Delta y=P_h d_y(P)\rightarrow\widehat{G_x}=P_x+\Delta x,\widehat{G_y}=P_y+\Delta y,$

尺度缩放 $(S_w,S_h),S_w=\exp\left(d_w(P)\right),S_h=\exp\left(d_h(P)\right)\rightarrow\widehat{G_w}=P_w S_w,\widehat{G_h}=P_h S_h.$

推导出

$$t_x=\frac{G_x-P_x}{P_w},\ t_y=\frac{G_y-P_y}{P_h},t_w=\log\left(\frac{G_w}{P_w}\right),t_h=\log\left(\frac{G_h}{P_h}\right)$$

代入

$$w_*=\underset{w_*}{\arg\min}\sum_i^N(t_*^{(i)}-\hat{w}_*^T\varphi_5(P^{(i)}))^2+\lambda\|\hat{w}_*\|^2$$

第i个样本真实的平移或缩放量

w_*中的*可以是x, y, w, h；$\varphi_5(P^i)$是从P^i中提取出的特征；总共有N个训练样本；
$d_*(P)$与t_*分别代表P到$\hat{G}$和P到G的平移或者缩放量

References

[1] D. S. Bolme, J. R. Beveridge, B. Draper, Y. M. Lui, et al. Visual object tracking using adaptive correlation filters. In CVPR, 2010.
[2] J. Henriques, R. Caseiro, P. Martins, and J. Batista. Exploiting the Circulant Structure of Tracking-by-detection with Kernels. In ECCV,2012
[3] J. Henriques, R. Caseiro, P. Martins, and J. Batista. High-speed tracking with kernelized correlation filters. PAMI,37(3):583–596, 2015.
[4] M. Danelljan, G. Hager, F. S. Khan, and M. Felsberg. Accurate scale estimation for robust visual tracking. In Proceedings of the British Machine Vision Conference BMVC, 2014.
[5] http://blog.csdn.net/liulina603/article/details/8291093
[6] http://blog.csdn.net/jbddygb/article/details/62894692
[7] http://blog.csdn.net/v_july_v/article/details/7624837
[8] https://www.jianshu.com/p/fe428f0b32c1
[9] https://zhuanlan.zhihu.com/p/31427728
[10] https://www.cnblogs.com/wangxiaocvpr/p/5598608.html
[11] https://www.zhihu.com/question/26493945
[12] 周志华.机器学习,北京:清华大学出版社,2016.
[13] https://www.youtube.com/watch?v=zYM7G5qd090&t=8s