Tracking with Correlation Filters

Lecture for AE4M33MVP

Acknowledgement to João F. Henriques from
Institute of Systems and Robotics
University of Coimbra
for providing materials for this presentation

Lecture Overview

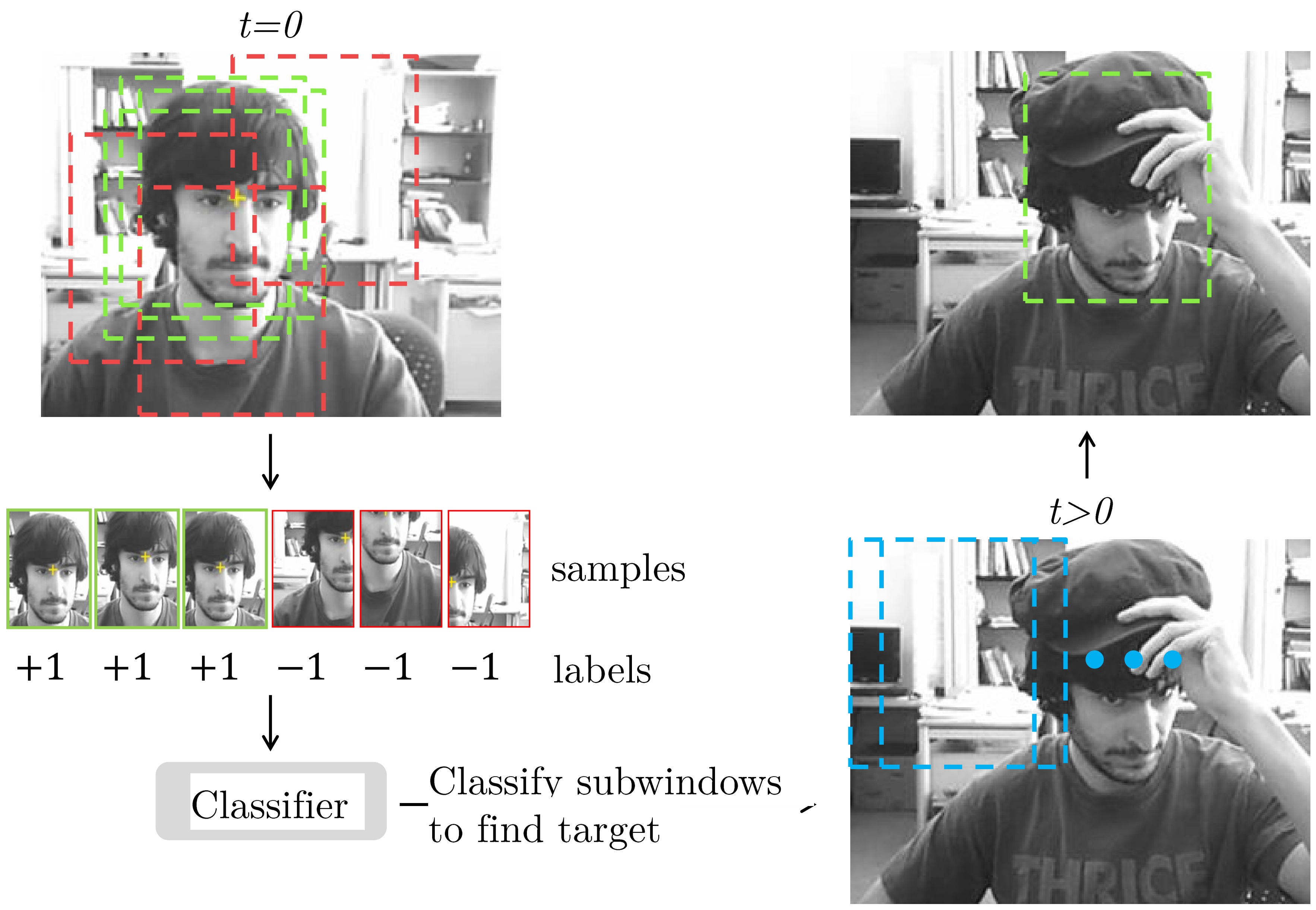
- Discriminative tracking
- Connection of correlation and the discriminative tracking

- Brief history of correlation filters
- Breakthrough by MOSSE tracker

- Why MOSSE works?
  (connection of correlation filters and machine learning)
  - Circulant matrices
  - Ridge Regression

- Kernelized Correlation Filters

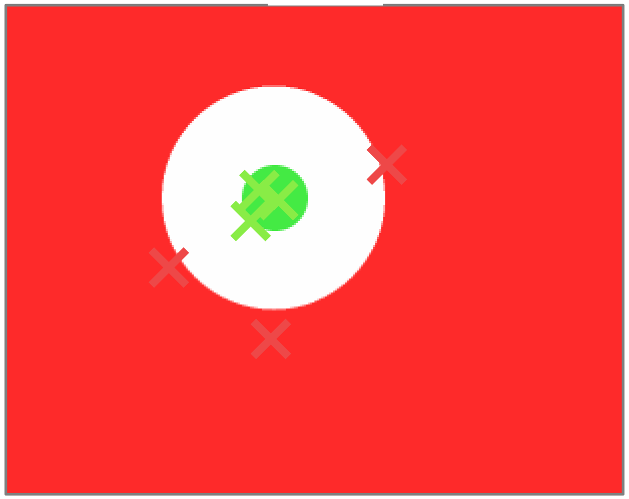
Discriminative Tracking



Discriminative Tracking

- How to get training samples for the classifier?
- Standard approach:
  - bboxes with high overlap with the GT → Pos. samples
  - bboxes far from the GT → Neg. samples

t=0



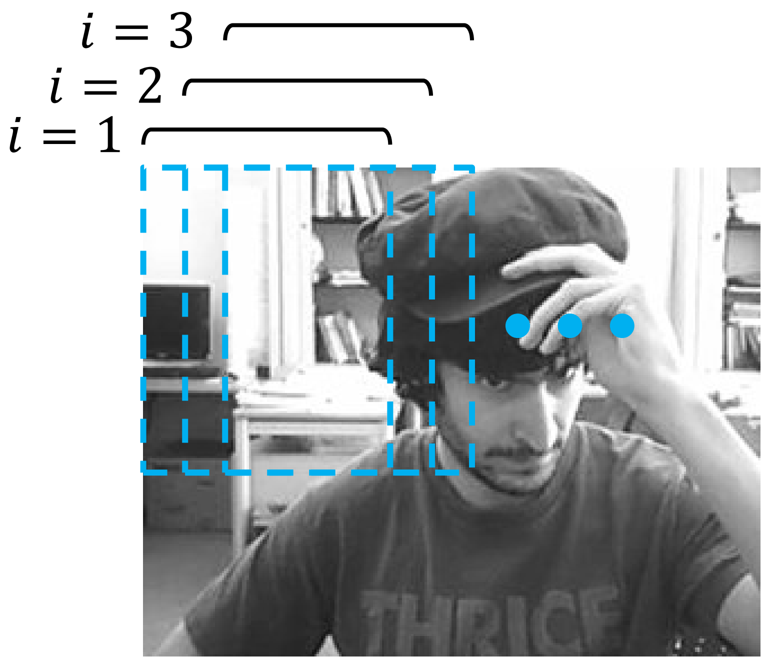
- 🟥 Neg. samples
- 🟩 Pos. samples
- ⬜ Unspecified

- What with the samples in the unspecified area?

Connection to Correlation

- Let's have a linear classifier with weights **w**

$$y = \mathbf{w}^T \mathbf{x}$$



- During tracking we want to evaluate the classifier at subwindows $\mathbf{x}_i$:

$$y_i = \mathbf{w}^T \mathbf{x}_i$$

- Then we can concatenate $\mathbf{y}_\mathrm{i}$ into a vector $\mathbf{y}$ (i.e. response map)

- This is equivalent to **cross-correlation** formulation which can be computed efficiently in Fourier domain

$$\mathbf{y} = \mathbf{x} \circledast \mathbf{w}$$

- Note: Convolution is related; it is the same as cross-correlation, but with the flipped image of **w** (P → d).

Connection to Correlation

The Convolution Theorem

“Cross-correlation is **equivalent** to an **element-wise product** in Fourier domain”

$$\mathbf{y} = \mathbf{x} \circledast \mathbf{w} \quad \Longleftrightarrow \quad \hat{\mathbf{y}} = \hat{\mathbf{x}}^{*} \times \hat{\mathbf{w}}$$

- where:
  - $\hat{\mathbf{y}} = \mathcal{F}(\mathbf{y})$ is the Discrete Fourier Transform (DFT) of $\mathbf{y}$. (likewise for $\hat{\mathbf{x}}$ and $\hat{\mathbf{w}}$)
  - $\times$ is element-wise product
  - $.^{*}$ is complex-conjugate (i.e. negate imaginary part).

- Note that cross-correlation, and the DFT, are **cyclic** (the window wraps at the image edges).

Connection to Correlation

The Convolution Theorem

“Cross-correlation is **equivalent** to an **element-wise product** in Fourier domain”

$$\mathbf{y} = \mathbf{x} \circledast \mathbf{w} \quad \Longleftrightarrow \quad \hat{\mathbf{y}} = \hat{\mathbf{x}}^* \times \hat{\mathbf{w}}$$

- In practice:

$$\begin{tikzcd}
\mathbf{x} \arrow[r] & \boxed{\mathcal{F}} \arrow[r] & \boxed{.^*} \arrow[dr, "\hat{\mathbf{x}}^*"] & & & \\
& & & \boxed{\times} \arrow[r, "\hat{\mathbf{y}}"] & \boxed{\mathcal{F}^{-1}} \arrow[r] & \mathbf{y} \\
\mathbf{w} \arrow[r] & \boxed{\mathcal{F}} \arrow[urr, "\hat{\mathbf{w}}"'] & & & &
\end{tikzcd}$$

- Can be orders of magnitude faster:
  - For $n \times n$ images, cross-correlation is $\mathcal{O}(n^4)$.
  - Fast Fourier Transform (and its inverse) are $\mathcal{O}(n^2 \log n)$.

Connection to Correlation

The Convolution Theorem

“Cross-correlation is **equivalent** to an **element-wise product** in Fourier domain”

$$\mathbf{y} = \mathbf{x} \circledast \mathbf{w} \quad \Longleftrightarrow \quad \hat{\mathbf{y}} = \hat{\mathbf{x}}^{*} \times \hat{\mathbf{w}}$$

- Conclusion:

  The evaluation of any linear classifier can be accelerated with the Convolution Theorem. (Not just for tracking.)

- “linear” can become non-linear using kernel trick in some specific cases (will be discussed later)

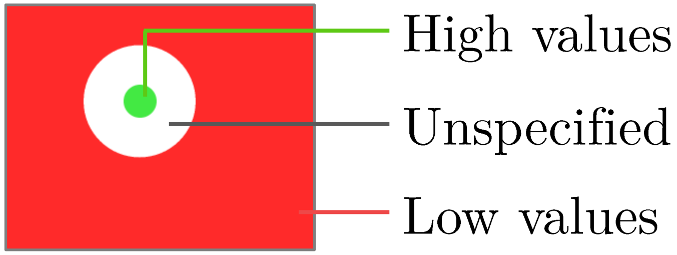
- Q: How the $\mathbf{w}$ for correlation should look like? What about **training**?

Connection to Correlation

- Q: How the **w** for correlation should look like? What about **training**?

Objective

$\circledast$ **w** $=$



- Intuition of requirements of cross-correlation of classifier(filter) **w** and a training image **x**
  - A **high peak** near the true location of the target
  - **Low values** elsewhere (to minimize false positive)

Brief History of Correlation Filters

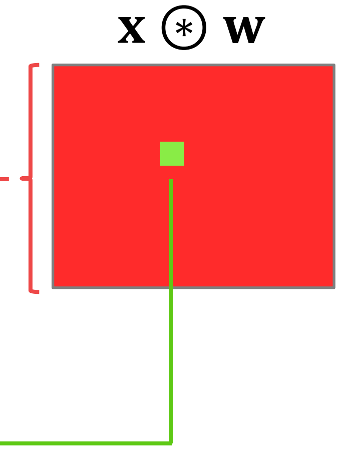
Minimum Average Correlation Energy (MACE) filters, 1980's

- Bring average correlation output towards 0:

$$\min_{\mathbf{w}} \|\mathbf{x} \circledast \mathbf{w}\|^2$$

except for target location, keep the peak value fixed:

subject to: $\mathbf{w}^T \mathbf{x} = 1$



- This produces a **sharp peak** at target location
with closed form solution:

$$\hat{\mathbf{w}} = \frac{\hat{\mathbf{x}}}{\hat{\mathbf{x}}^* \times \hat{\mathbf{x}}}$$

• $\hat{\mathbf{x}}^* \times \hat{\mathbf{x}}$ is called the **spectrum** and is real-valued.
• division and product $(\times)$ are element-wise.

- **Sharp peak = good localization!** Are we done?

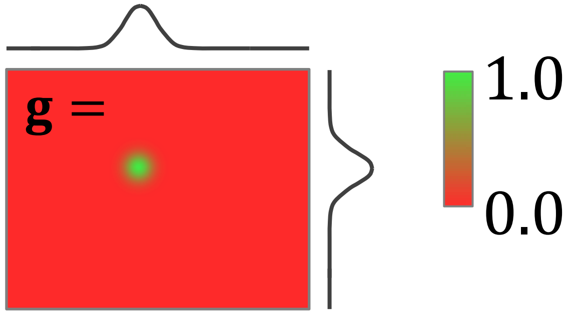
Brief History of Correlation Filters

The MACE filter suffer from 2 main issues:

1. Hard constraints easily lead to overfitting.
   - **UMACE** ("Unconstrained MACE") addresses this by removing the hard constraints and require to produce a high average correlation response on positive samples. However, it still suffer from the 2nd problem.
2. Enforcing a sharp peak is too strong condition; lead to overfitting
   - **Gaussian-MACE / MSE-MACE** – peak to follow a 2D Gaussian shape

$$\min_{\mathbf{w}} \|\mathbf{x} \circledast \mathbf{w} - \mathbf{g}\|^2 ,$$

$$\text{subject to:} \quad \mathbf{w}^T \mathbf{x} = 1$$

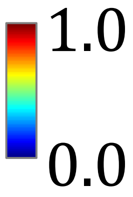


   - In the original method (1990's), the minimization was *still* subject to the MACE hard constraint.
     (*It later turned out to be unnecessary!*)

Brief History of Correlation Filters

Sharp vs. Gaussian peaks

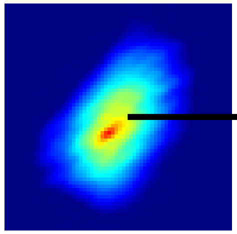
Training image: $\mathbf{x} =$



Naïve filter
$(\mathbf{w} = \mathbf{x})$

Classifier
$(\mathbf{w})$

Output
$(\mathbf{w} * \mathbf{x})$

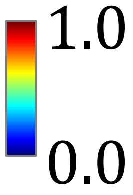


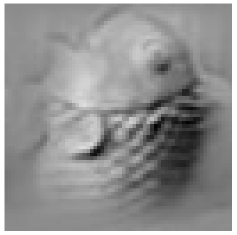
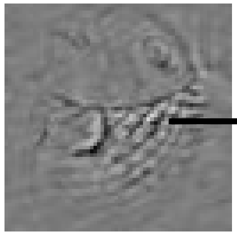
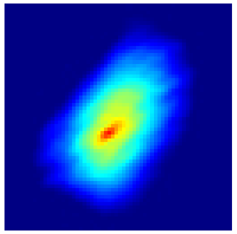
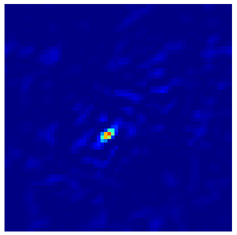
- Very broad peak is hard to localize (especially with clutter).
- State-of-the-art classifiers (e.g. SVM) show **same** behavior!

Brief History of Correlation Filters

Sharp vs. Gaussian peaks

Training image: $\mathbf{x} =$



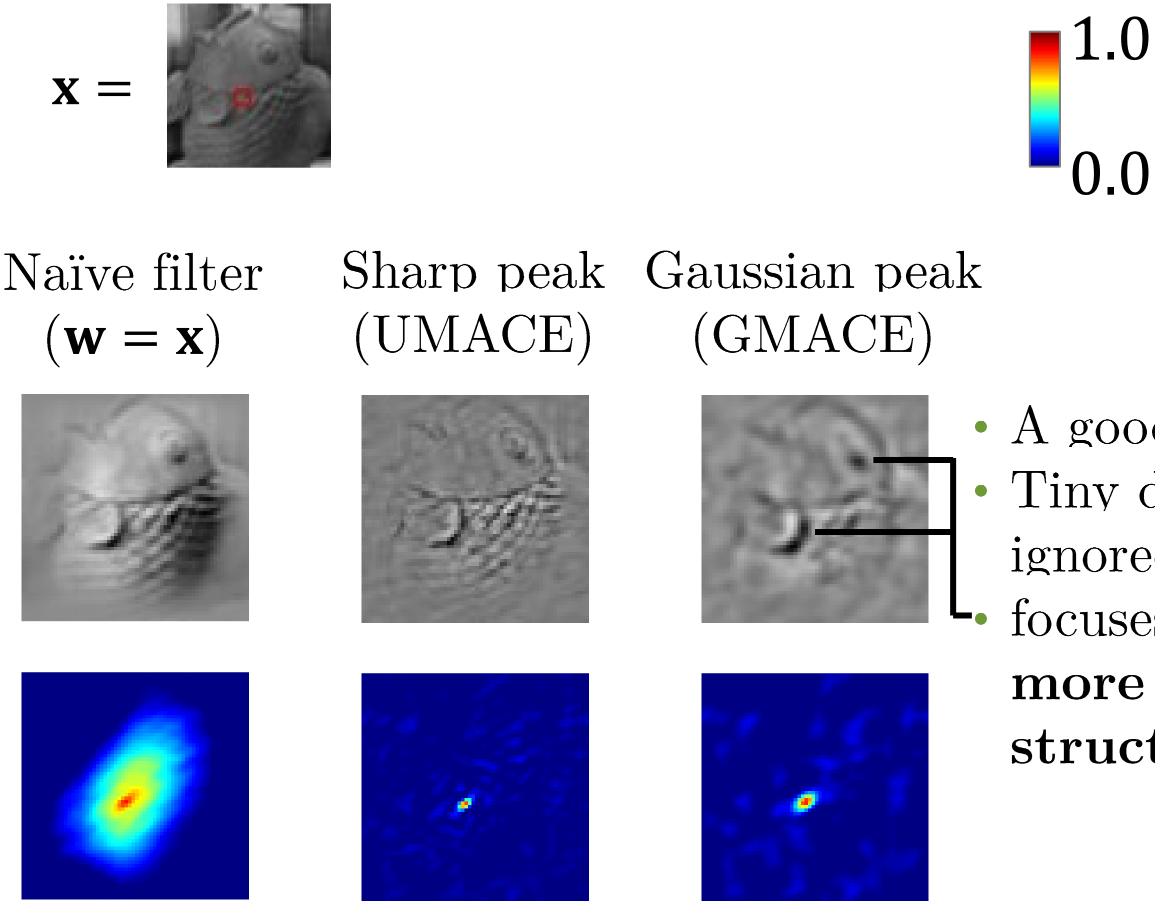
	Naïve filter ($\mathbf{w} = \mathbf{x}$)	Sharp peak (UMACE)
Classifier ($\mathbf{w}$)		
Output ($\mathbf{w} * \mathbf{x}$)		

- A very sharp peak is obtained by emphasizing **small image details** (like the fish's scales here).
- **generalizes poorly**; fine scale details that are usually not robust

Brief History of Correlation Filters

Sharp vs. Gaussian peaks

Training image: $\mathbf{x} =$


1.0
0.0
Naïve filter ($\mathbf{w} = \mathbf{x}$)
Sharp peak (UMACE)
Gaussian peak (GMACE)
Classifier ($\mathbf{w}$)
Output ($\mathbf{w} * \mathbf{x}$)

- A good compromise.
- Tiny details are ignored.
- focuses on **larger, more robust structures**.

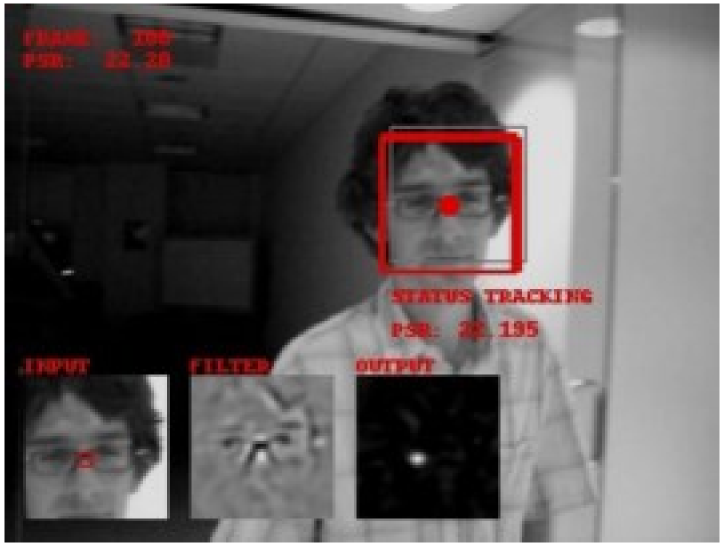
Breakthrough by MOSSE tracker

Min. Output Sum of Sq. Errors (MOSSE)

- Presented by David Bolme and colleagues at CVPR 2010

- Tracker run at speed over a
  600 frames per second

- **very simple** to implement
  - no complex features only
    raw pixel values
  - only FFT and element-wise operation



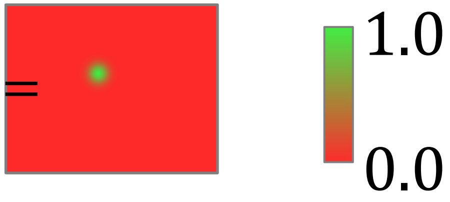
- performance similar to the most sophisticated tracker (at that time)

Breakthrough by MOSSE tracker

How does it work?

- Use only the “Gaussian peak” objective (no hard constraints)

$$\min_{\mathbf{w}} \|\mathbf{x} \circledast \mathbf{w} - \mathbf{g}\|^2 ,$$

$\mathbf{g} =$ 

- Found the following solution using the Convolution Theorem:

$$\hat{\mathbf{w}} = \frac{\hat{\mathbf{g}}^* \times \hat{\mathbf{x}}}{\hat{\mathbf{x}}^* \times \hat{\mathbf{x}} + \lambda}$$

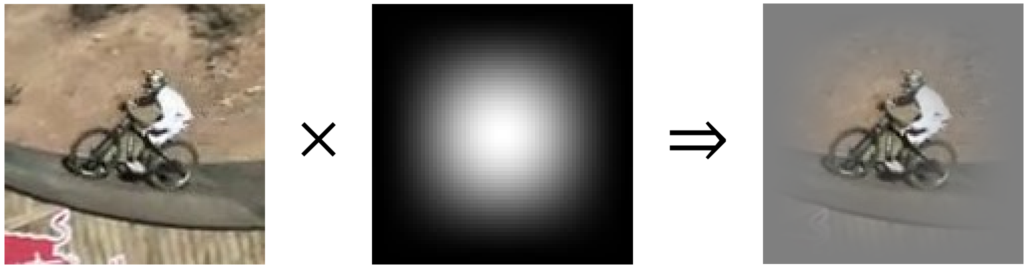
($\lambda = 10^{-4}$ is artificially added to prevent divisions by 0)

- **No expensive matrix operations!** $\Rightarrow$ only FFT and element-wise op.

Breakthrough by MOSSE tracker

Implementation aspects

- Cosine (or sine) window preprocessing



  - image edges smooth to zero
    ⇀ the filter sees an image as a "cyclic" (important for the FFT)
  - gives more importance to the target center.

- Simple update

$$\hat{\mathbf{w}}_{\text{new}} = \frac{\hat{\mathbf{g}}^* \times \hat{\mathbf{x}}}{\hat{\mathbf{x}}^* \times \hat{\mathbf{x}} + \lambda}$$

Train a MOSSE filter $\hat{\mathbf{w}}_{\text{new}}$ using the new image $\hat{\mathbf{x}}$.

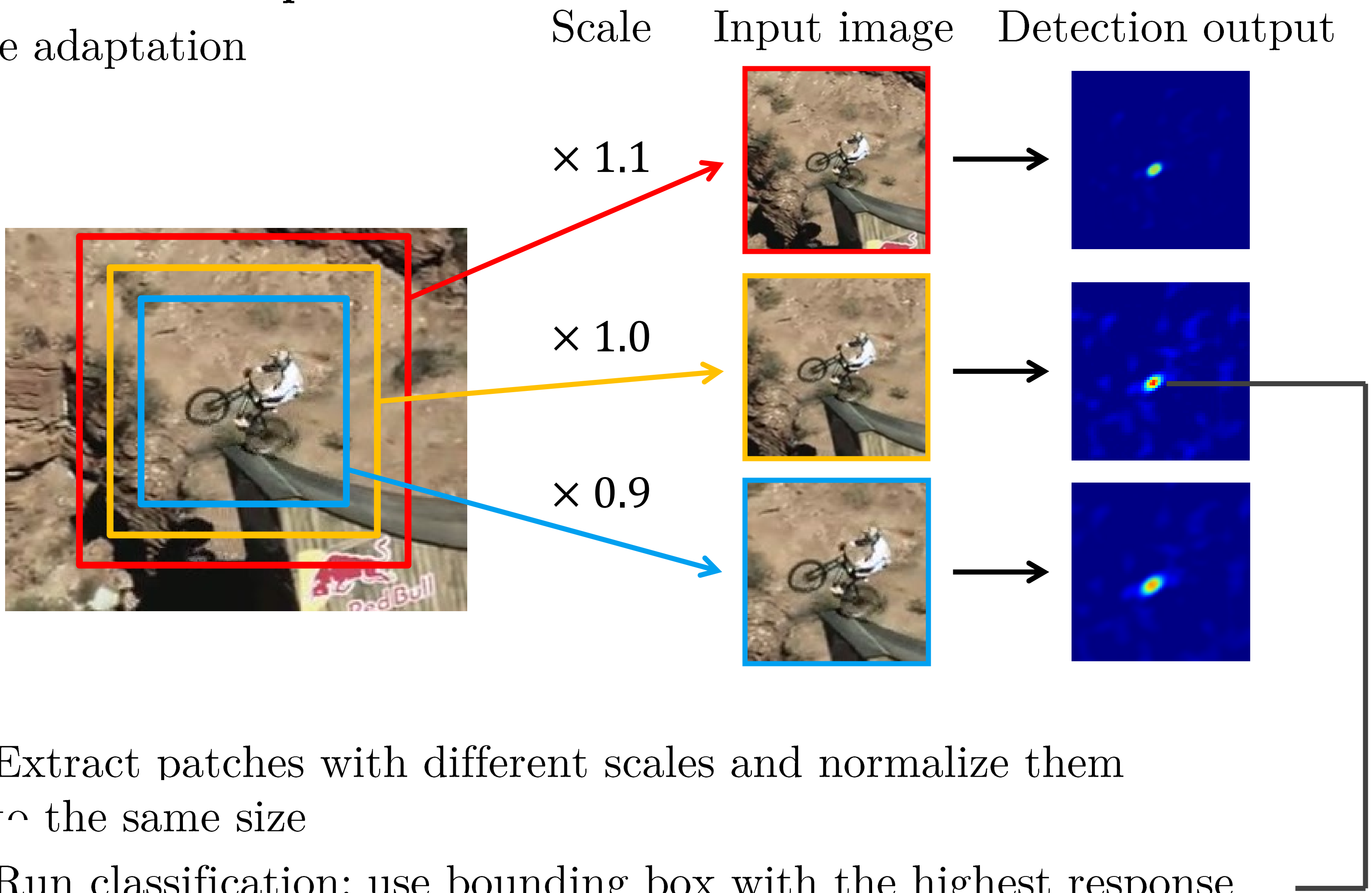
$$\hat{\mathbf{w}}_t = (1-\eta)\hat{\mathbf{w}}_{t-1} + \eta\hat{\mathbf{w}}_{\text{new}}$$

Update previous solution $\hat{\mathbf{w}}_{t-1}$ with $\hat{\mathbf{w}}_{\text{new}}$ by linear interpolation.

Breakthrough by MOSSE tracker

Implementation aspects

- Scale adaptation



- Extract patches with different scales and normalize them to the same size
- Run classification; use bounding box with the highest response

Why MOSSE works?

Circulant matrices

is a tool that connects **correlation filters** with **machine learning**

$$\min_{\mathbf{w}} \|\mathbf{x} \circledast \mathbf{w} - \mathbf{g}\|^2 \xrightarrow[\text{a special matrix } C(\mathbf{x})]{\text{replace correlation with}} \boxed{\min_{\mathbf{w}} \|C(\mathbf{x})\mathbf{w} - \mathbf{g}\|^2}$$

- $C(\mathbf{x})$ is a **circulant matrix:**

$$C(\mathbf{u}) = \begin{bmatrix} u_0 & u_1 & u_2 & \cdots & u_{n-1} \\ u_{n-1} & u_0 & u_1 & \cdots & u_{n-2} \\ u_{n-2} & u_{n-1} & u_0 & \cdots & u_{n-3} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ u_1 & u_2 & u_3 & \cdots & u_0 \end{bmatrix}$$

Why MOSSE works?

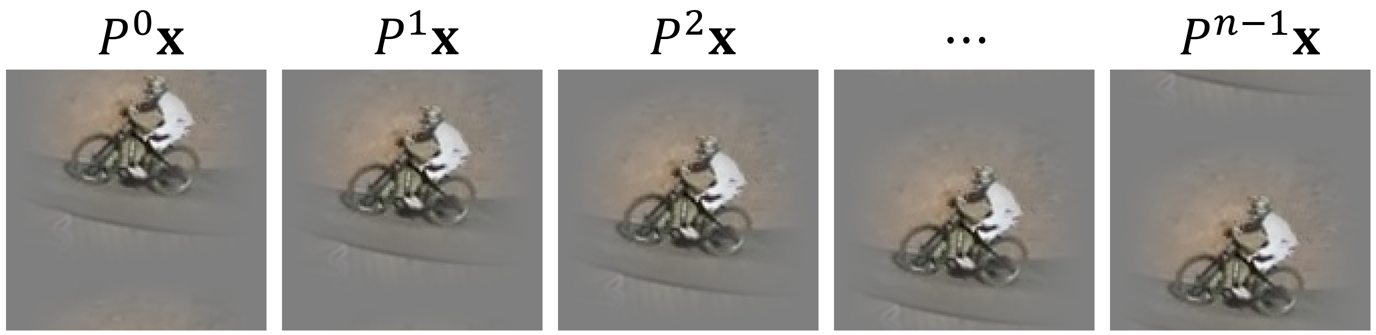
Circulant matrices

is a tool that connects **correlation filters** with **machine learning**

- We can see $\mathrm{X} = C(\mathbf{x})$ as a **dataset** with **cyclically shifted** versions of the image $\mathbf{x}$

$$X = \begin{bmatrix} (P^0\mathbf{x})^T \\ (P^1\mathbf{x})^T \\ \vdots \\ (P^{n-1}\mathbf{x})^T \end{bmatrix}$$

- P is a permutation matrix that shifts the pixels in vertical/horizontal direction by 1 element.
- Arbitrary shift i obtained with power $P^i\mathbf{x}$.
- Cyclic: $P^n\mathbf{x} = P^0\mathbf{x} = \mathbf{x}$.



Why MOSSE works?

Circulant matrices

is a tool that connects **correlation filters** with **machine learning**

- Similar role to the Convolution Theorem

$$\underbrace{X = \begin{bmatrix} (P^0\mathbf{x})^T \\ (P^1\mathbf{x})^T \\ \vdots \\ (P^{n-1}\mathbf{x})^T \end{bmatrix}}_{\text{Data matrix is } \textbf{circulant}} \Rightarrow \underbrace{\mathcal{F}(X) = \begin{bmatrix} \hat{\mathbf{x}}_1 & 0 & \cdots & 0 \\ 0 & \hat{\mathbf{x}}_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \hat{\mathbf{x}}_n \end{bmatrix}}_{\text{Becomes } \textbf{diagonal} \text{ in Fourier domain}}$$

- **Most of the "data" is 0 and can be ignored!** ⇒ Massive speed-up

Why MOSSE works?

Ridge Regression Formulation

= Least-Squares with regularization (avoids overfitting!)

- Consider simple Ridge Regression (RR) problem:

$$\min_{\mathbf{w}} \|X\mathbf{w} - \mathbf{y}\|^2 + \lambda\|\mathbf{w}\|^2$$

has closed-form solution: $\mathbf{w} = (X^T X + \lambda I)^{-1} X^T \mathbf{y}$

We can replace X $= C(\mathbf{x})$ (circulant data), and $\mathbf{y} = \mathbf{g}$ (Gaussian targets).

- **Diagonalizing** the involved circulant matrices with the DFT yields:

$$\hat{\mathbf{w}} = \frac{\hat{\mathbf{x}}^* \times \hat{\mathbf{y}}}{\hat{\mathbf{x}}^* \times \hat{\mathbf{x}} + \lambda} \quad \Rightarrow$$

- Exactly the MOSSE solution!
- **good learning algorithm** (RR) with **lots of data** (circulant/shifted samples).

Kernelized Correlation Filters

- Circulant matrices are a **very general tool** which allows to replace standard operations with fast Fourier operations.
- The same idea can by applied e.g. to the **Kernel Ridge Regression**:

  with K kernel matrix $K_{ij} = \kappa(\mathbf{x}_i, \mathbf{x}_j)$ and dual space representation

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

- For many kernels, circulant data $\Rightarrow$ circulant K matrix

$$K = C(\mathbf{k}^{\mathbf{xx}}),$$

where $\mathbf{k}^{\mathbf{xx}}$ is kernel auto-correlaton and the first row of K (small, and easy to compute)

- Diagonalizing with the DFT for learning the classifier yields:

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

Fast solution in $\mathcal{O}(n \log n)$.
Typical kernel algorithms are $\mathcal{O}(n^2)$ or higher!

Kernelized Correlation Filters

- The $\mathbf{k}^{\mathbf{x}\mathbf{x}'}$ is kernel correlation of two vectors **x** and **x'**

$$k_i^{\mathbf{x}\mathbf{x}'} = \kappa(\mathbf{x}',\ P^{i-1}\mathbf{x})$$

- For Gaussian kernel it yields:

multiple channels can be concatenated to the vector **x** and then sum over in this term

$$\mathbf{k}^{\mathbf{x}\mathbf{x}'} = \exp\left(-\frac{1}{\sigma^2}\left(\|\mathbf{x}\|^2 + \|\mathbf{x}'\|^2 - 2\mathcal{F}^{-1}(\hat{\mathbf{x}}^* \odot \hat{\mathbf{x}}')\right)\right)$$

- Evaluation on subwindows of image **z** with classifier $\boldsymbol{\alpha}$ and model **x**:

1. $K^{\mathbf{z}} = C(\mathbf{k}^{\mathbf{x}\mathbf{z}})$
2. $\mathbf{f}(\mathbf{z}) = \mathcal{F}^{-1}\left(\hat{\mathbf{k}}^{\mathbf{x}\mathbf{z}} \odot \hat{\boldsymbol{\alpha}}\right)$

- Update classifier $\boldsymbol{\alpha}$ and model **x** by linear interpolation from the location of maximum response $\mathbf{f}(\mathbf{z})$

- Kernel allows integration of more complex and multi-channel features

Kernelized Correlation Filters

KCF Tracker

- very few hyperparameters
- code fits on one slide of the presentation!
- Use HoG features (32 channels)
- ~300 FPS
- Open-Source (Matlab/Python/Java/C)

Training and detection (Matlab)

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

function y = detect(alphaf, x, z, sigma)
  k = kernel_correlation(z, x, sigma);
  y = 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
```

Sum over channel dimension in kernel computation

Variations of KCF trackers

Basic

- Henriques et al. – CSK
  - raw grayscale pixel values as features
- Henriques et al. – KCF
  - HoG multi-channel features

Further work

- Danelljan et al. – DSST:
  - PCA-HoG + grayscale pixels features
  - filters for translation and for scale (in the scale-space pyramid)
- Li et al. – SAMF:
  - HoG, color-naming and grayscale pixels features
  - quantize scale space and normalize each scale to one size by bilinear inter. $\rightarrow$ only one filter on normalized size

Variations of KCF trackers

Further work

- Danelljan et al. –SRDCF:
  - spatial regularization in the learning process
    → limits boundary effect
    → penalize filter coefficients depending on their spatial location
  - allows to use much larger search region
  - more discriminative to background (more training data)

CNN-based Correlation Trackers

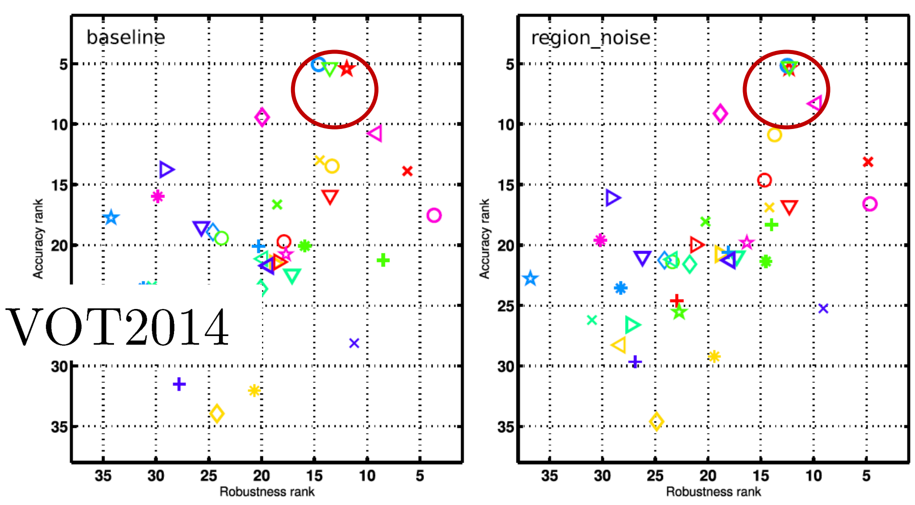
- Ma et al.
  - features : VGG-Net pretrained on ImageNet dataset extracted from third, fourth and fifth convolution layer
  - for each feature learn a linear correlation filter
  - coarse-to-fine approach from 5→3 layer
- Nam et al. – MDNet:
  - CNN classification (3 convolution layers and 2 fully connected layers) learn on tracking sequences with bbox regression

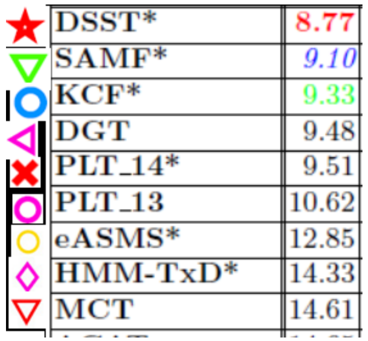
Results of KCF-based trackers

Result on recent standard evaluation benchmarks

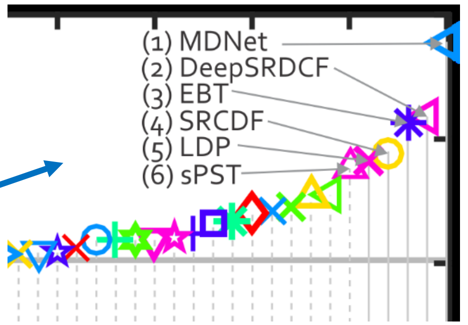
DSST, SAMF, KCF, DGT, PLT_{14}, PLT_{13}

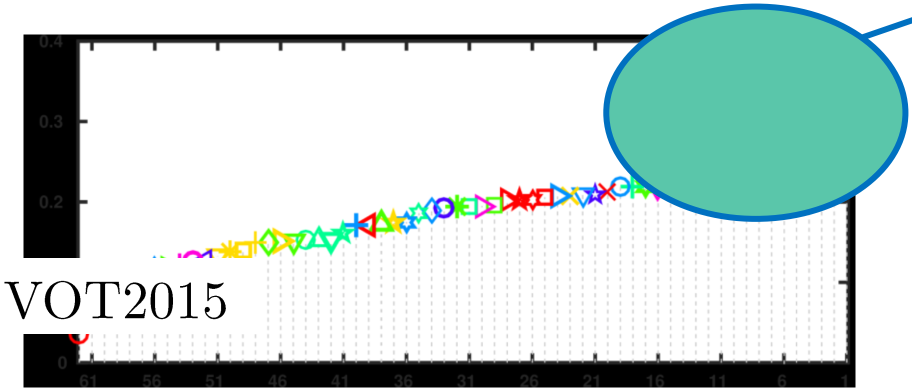
★ ▽ ○ ▷ ✖ ○





★	DSST*	8.77
▽	SAMF*	9.10
○	KCF*	9.33
◁	DGT	9.48
✖	PLT_14*	9.51
○	PLT_13	10.62
○	eASMS*	12.85
◇	HMM-TxD*	14.33
▽	MCT	14.61





Tracker		Type
MDNet*	◁	CNN learned on video sequences
DeepSRDCF	◁	Corr. Filter + CNN feats
EBT	✱	Edgebox features+SSVM+color hist.
SRDCF	○	Corr. Filter + color names + HoG
LDP	✕	Part-based Corr. Filter
sPST	△	Flow + Edgebox feats + SVM