



CS545 - Machine Learning for Signal Processing

Detection and Template Matching

13 September 2023

A change in thinking

- So far we did feature extraction
 - “Unsupervised learning”
- Now we’ll do detection and classification
 - “Supervised learning”

Today's lecture

- Detection of elements of interest
- Template matching / matched filters
 - Obtaining invariance/robustness
 - Efficient matching
 - Usage cases
- Probabilistic interpretation

Detection

- Find the presence of a predefined signal
 - Face/pedestrian/car detection
 - Speech, heartbeat, gunshot detection
 - Earthquakes, quasars, aliens, ...
- Many ways to go about it
 - We'll start from the basics

A simple example

- Can we detect a known face in a collection?

Query



Search set



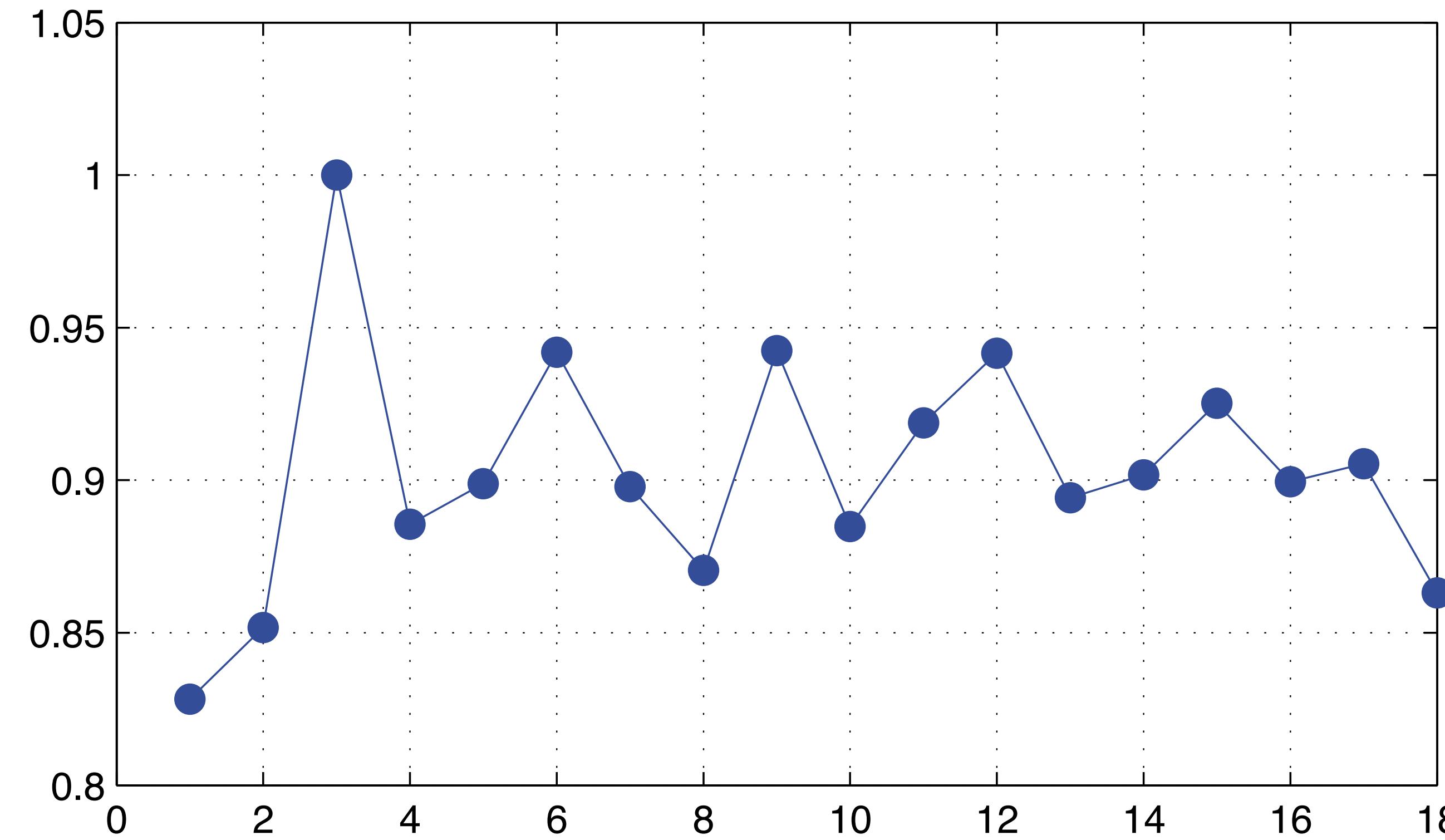
Vectorize the data (and scale to unit norm)

$$\mathbf{x}^\top = \text{vec}(\begin{matrix} \text{Face 1} \\ \text{Face 2} \\ \vdots \\ \text{Face 16} \end{matrix})^\top = \begin{matrix} \text{Pixel 1} & \text{Pixel 2} & \dots & \text{Pixel 1024} \end{matrix}$$

$$\mathbf{Y} = \text{vec}(\begin{matrix} \text{Face 1} & \text{Face 2} & \text{Face 3} & \text{Face 4} \\ \text{Face 5} & \text{Face 6} & \text{Face 7} & \text{Face 8} \\ \text{Face 9} & \text{Face 10} & \text{Face 11} & \text{Face 12} \\ \text{Face 13} & \text{Face 14} & \text{Face 15} & \text{Face 16} \end{matrix}) = \begin{matrix} \text{Pixel 1} & \text{Pixel 2} & \dots & \text{Pixel 1024} \\ \vdots & \vdots & \ddots & \vdots \\ \text{Pixel 1} & \text{Pixel 2} & \dots & \text{Pixel 1024} \end{matrix}$$

And get their dot product

$$\mathbf{X}^\top \cdot \mathbf{Y} =$$



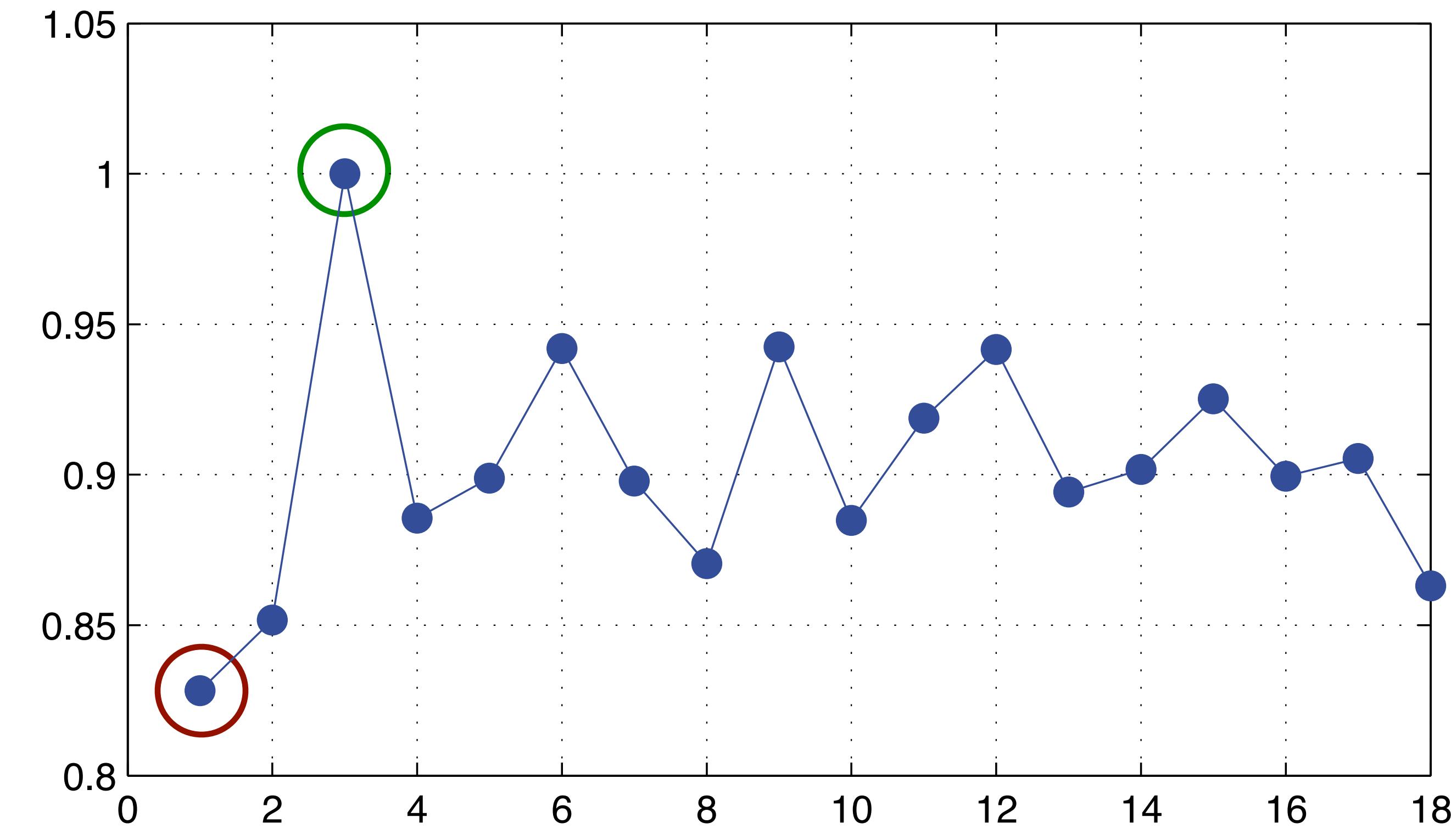
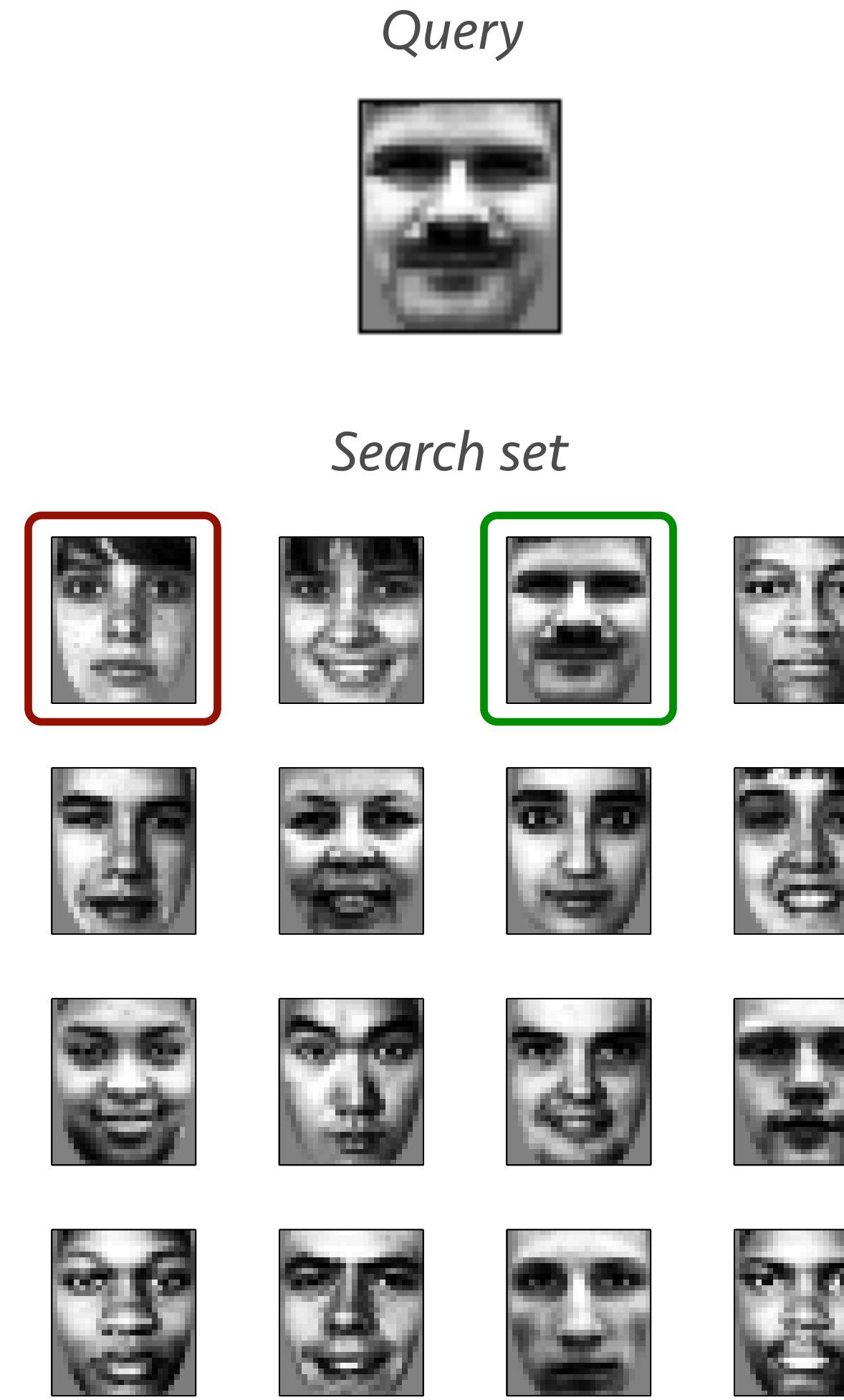
Dot product for matching

- For two unit-length vectors x, y

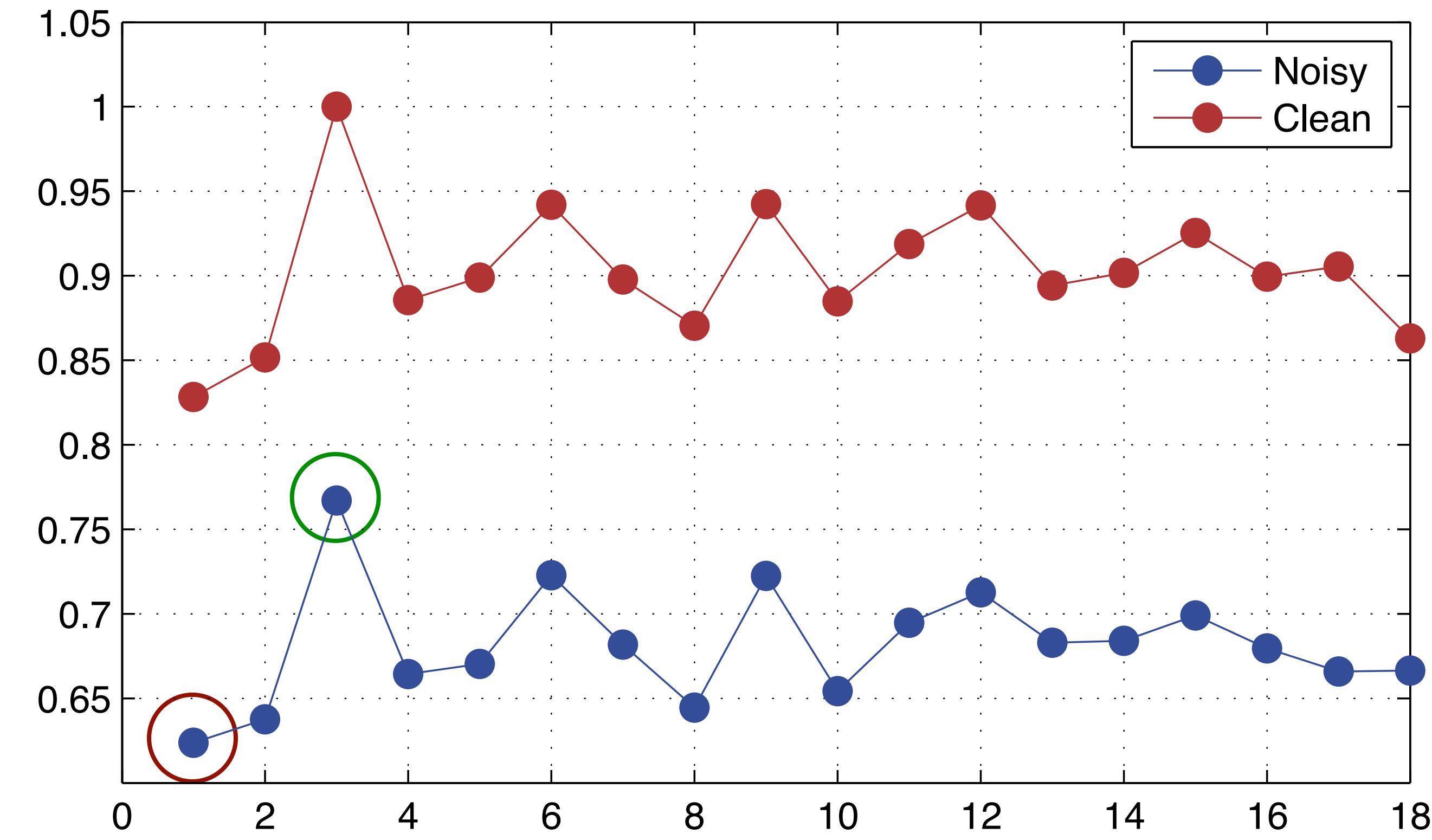
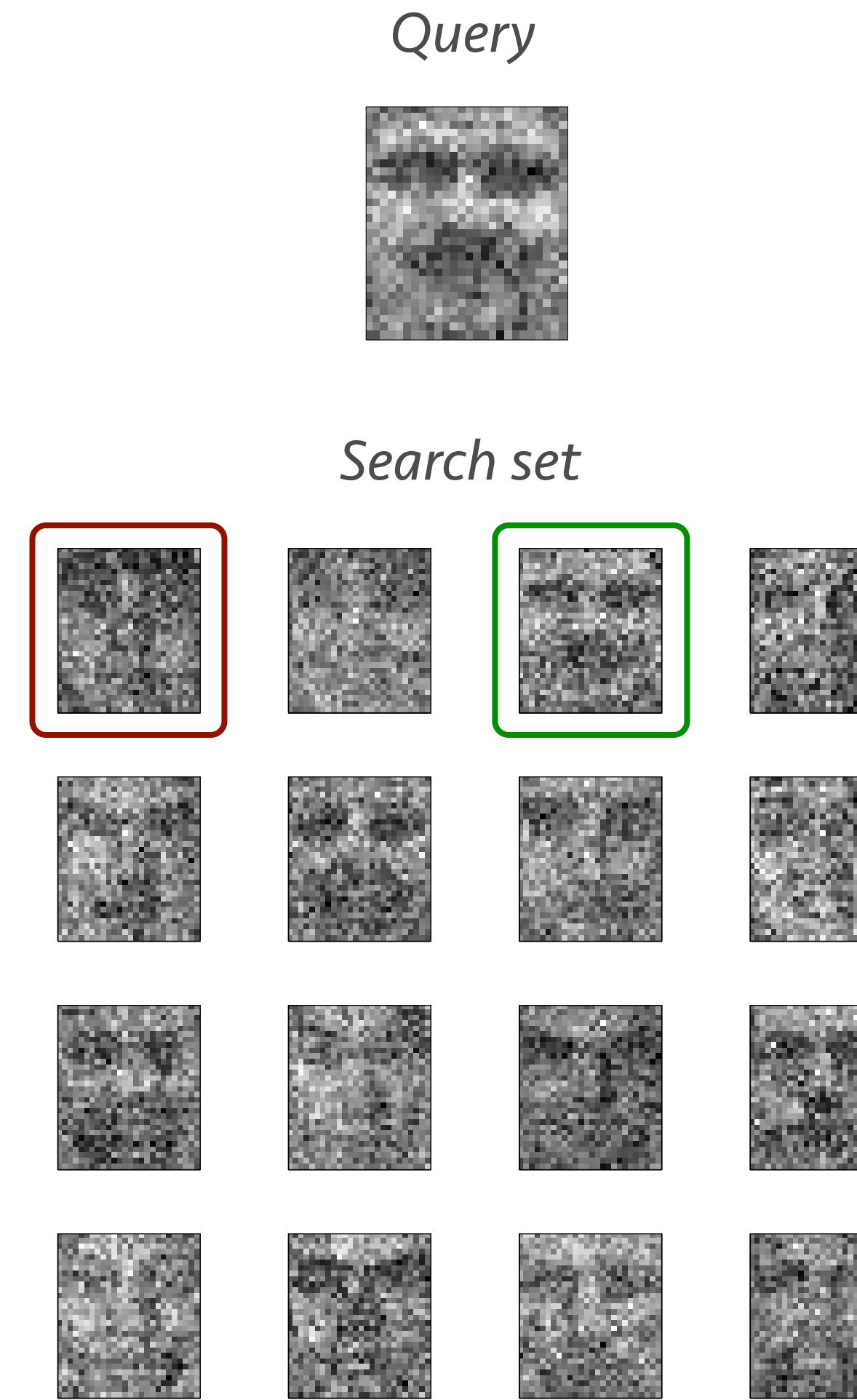
$$\mathbf{x}^\top \cdot \mathbf{y} = 1 - \varepsilon$$

- When x, y are increasingly similar, ε tends to zero
- If the dot product is maximized we have a best match

Interpreting the dot product



And it works ok for noisy inputs too

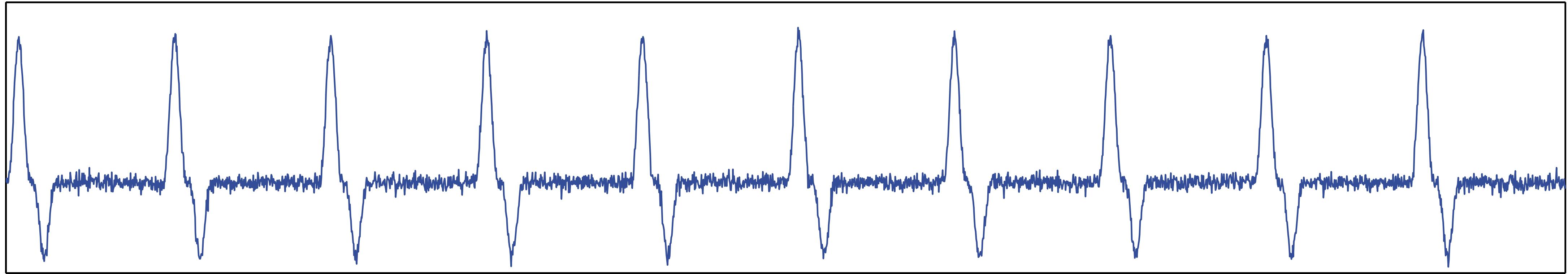


Template matching

- Our query is a template
 - Use dot products to compare it with inputs
- Simple and very fast detector/classifier
 - But it won't get you far as is!
- Let's signal-ize it

A more complex situation

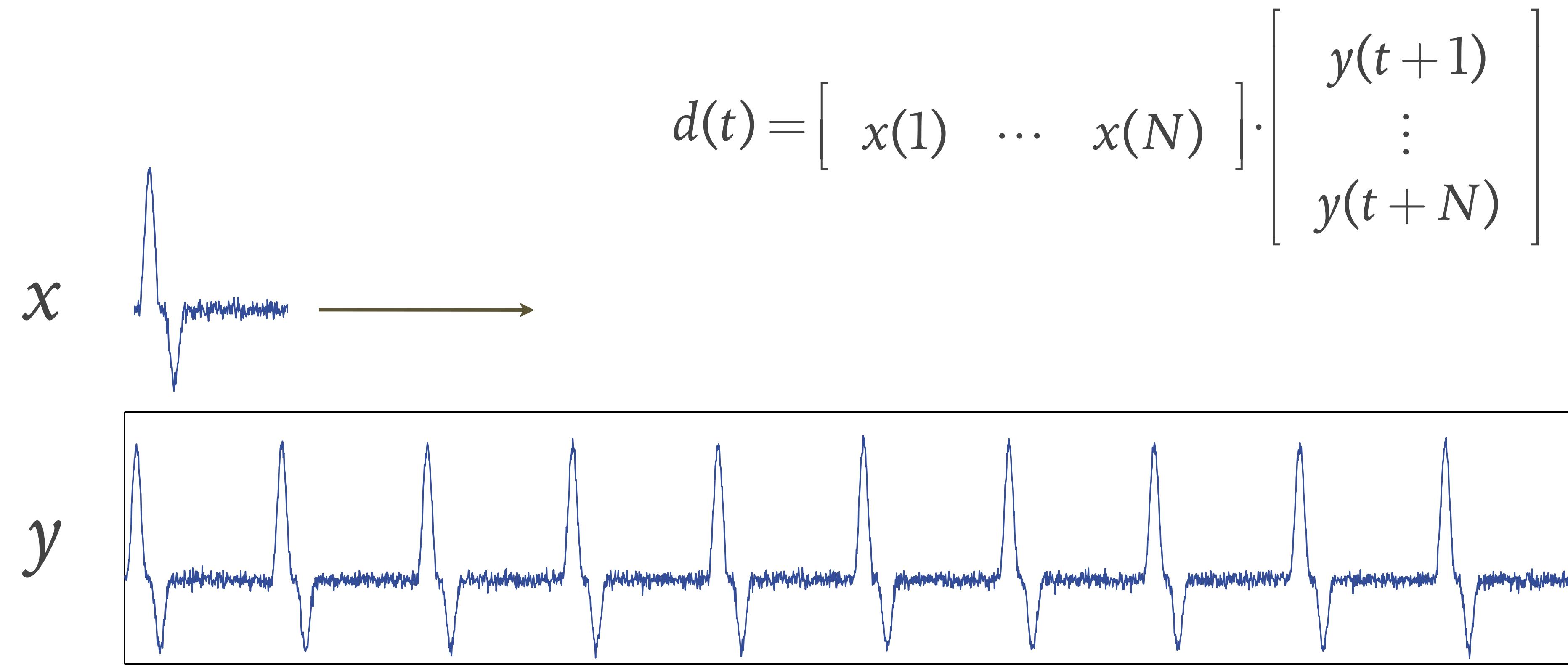
- “Heartbeat” data



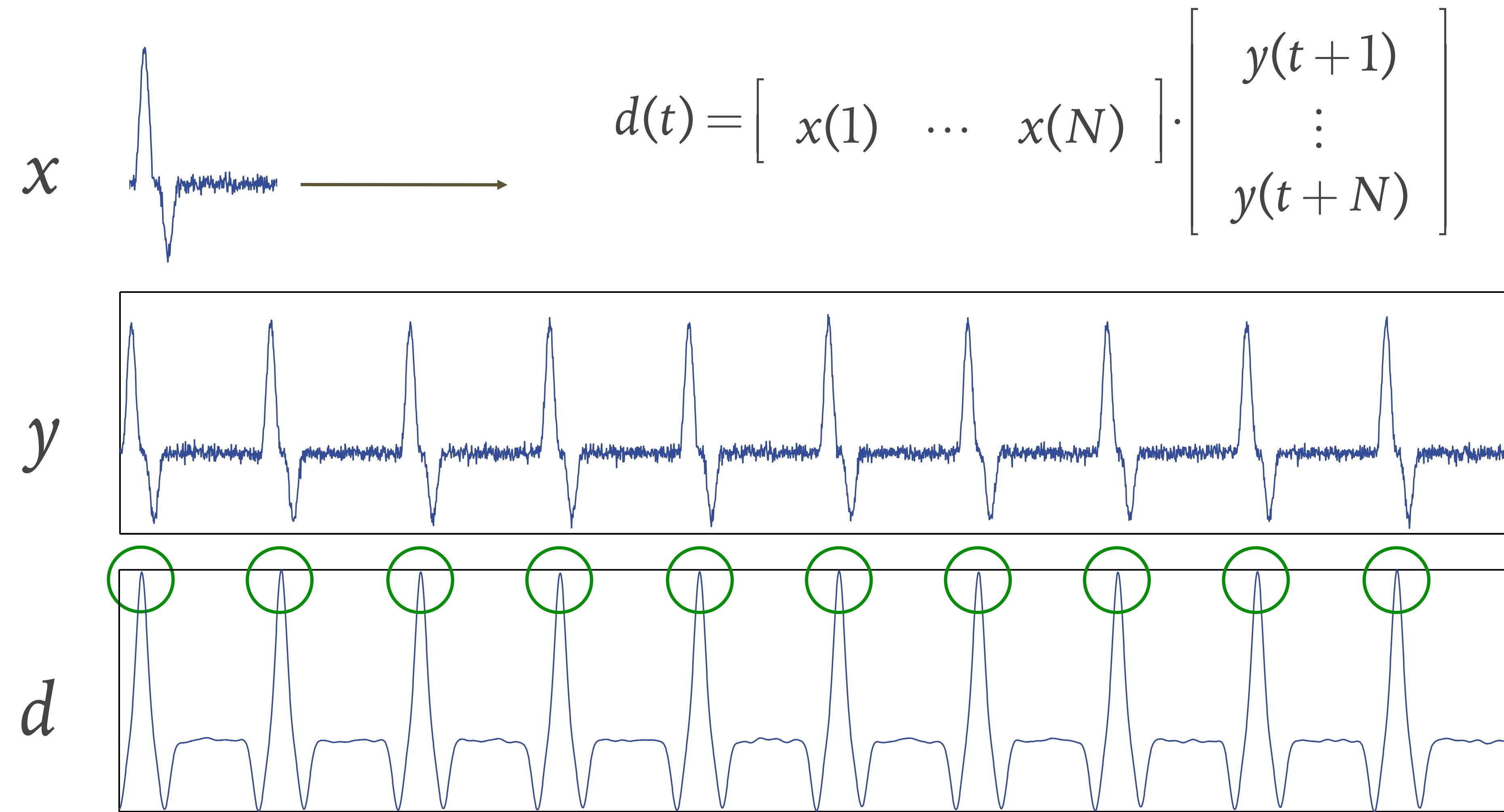
- How do we detect the beats?
 - They can be all over the place
 - How do we do the dot product?

Brute force approach

- Do all possible dot products



“Sliding” dot product



A closer look at this ...

- Writing it as a summation it becomes a convolution
 - But with the template time-reversed

$$\begin{aligned} d(t) &= \begin{bmatrix} x(1) & \dots & x(N) \end{bmatrix} \cdot \begin{bmatrix} y(t+1) \\ \vdots \\ y(t+N) \end{bmatrix} = \\ &= \sum_i x(i)y(t+i) = \hat{x} * y \\ \hat{x}(t) &= x(-t) \end{aligned}$$

Matched filter

- This is known as a *matched filter*

$$d(t) = \sum_k x(k)y(t-k)$$

Template presence at time t

k

Template (flipped in time)

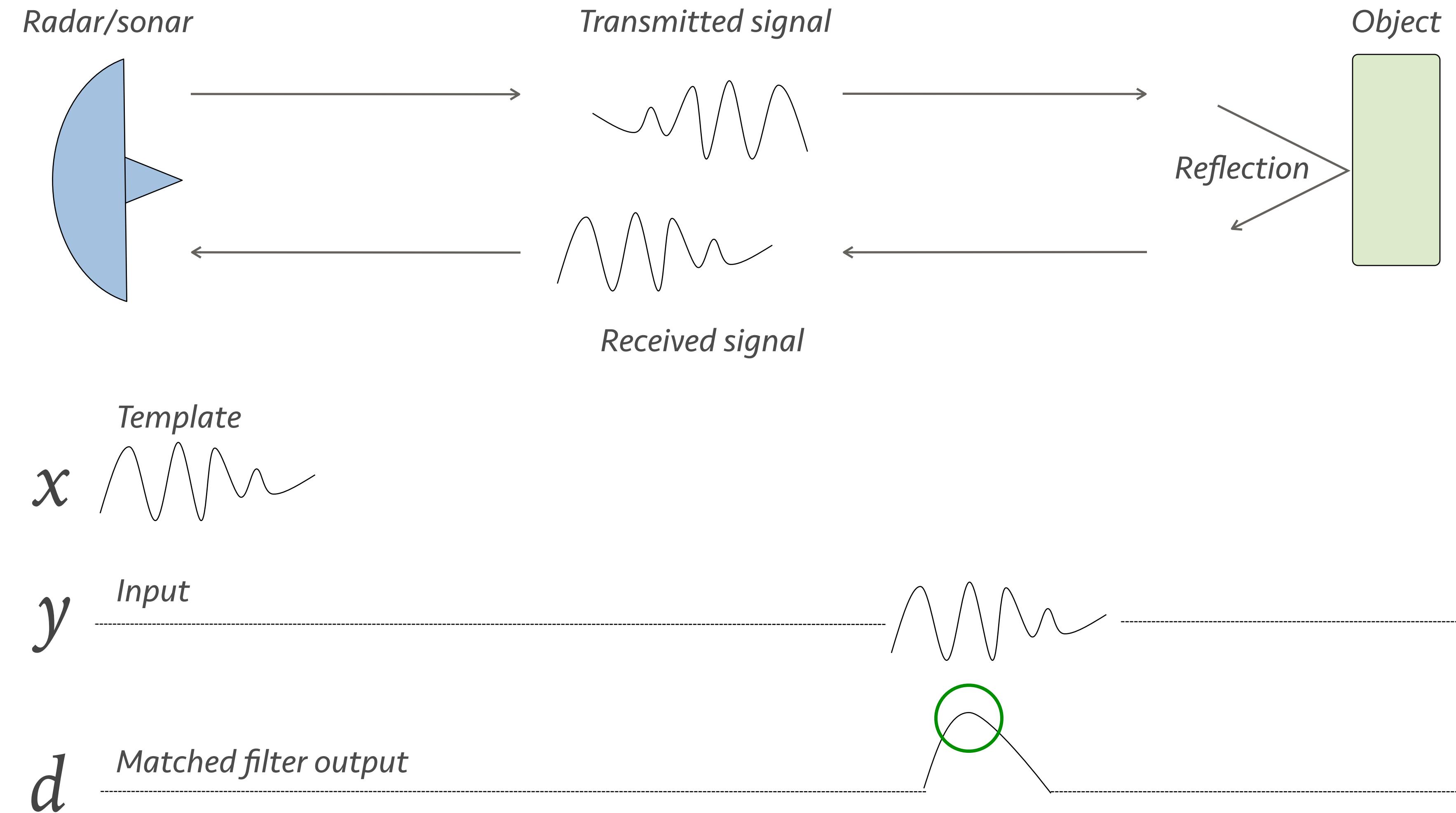
Input

The diagram illustrates the matched filter equation $d(t) = \sum_k x(k)y(t-k)$. It features three annotations with arrows pointing to specific parts of the equation:

- An arrow points from the output term $d(t)$ to the text "Template presence at time t ".
- An arrow points from the summation index k to the text "Template (flipped in time)".
- An arrow points from the input term $y(t-k)$ to the text "Input".

- Why filter? Because we do a convolution
- Also known as *cross-correlation*
- Very common tool in communications, sonar, radar, neural networks, etc ...

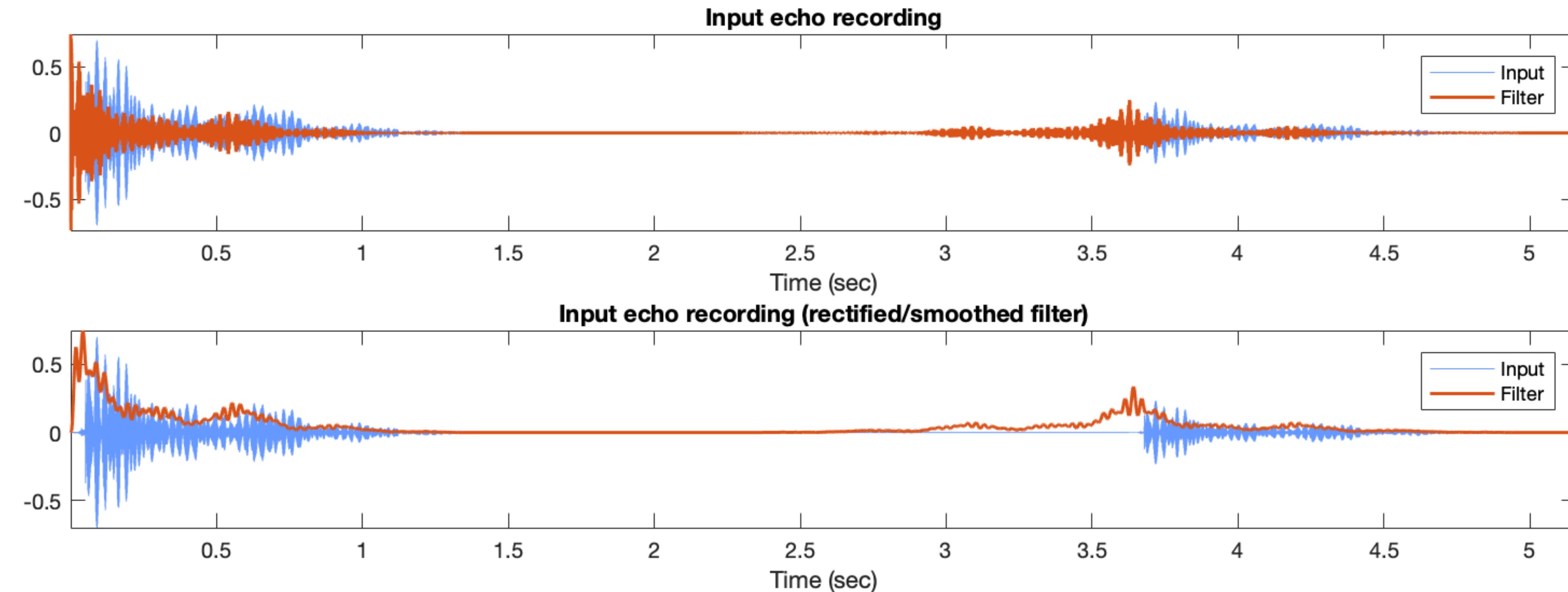
Matched filters in sonar/radar



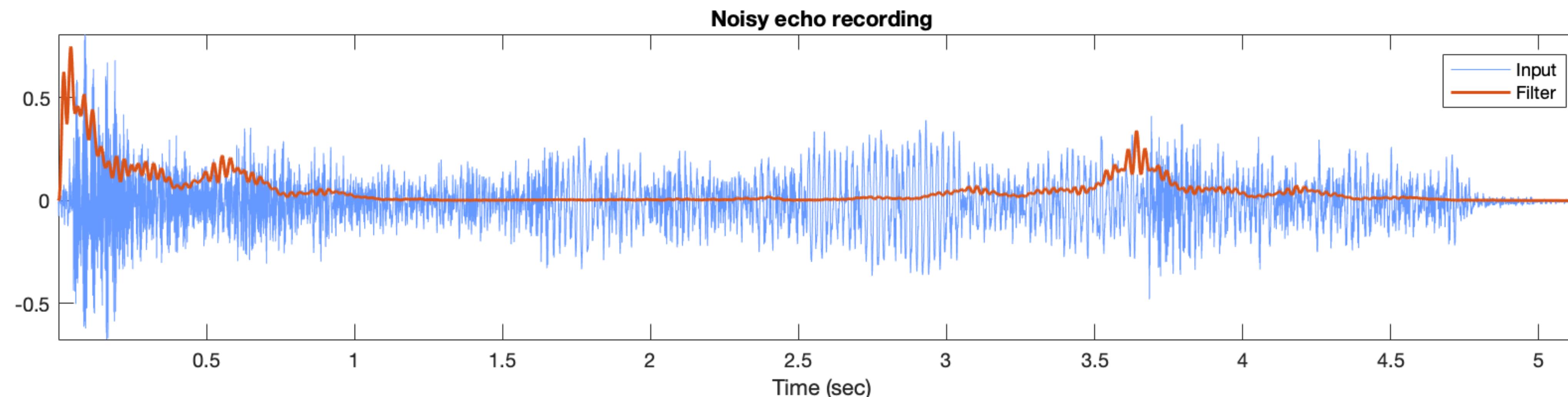
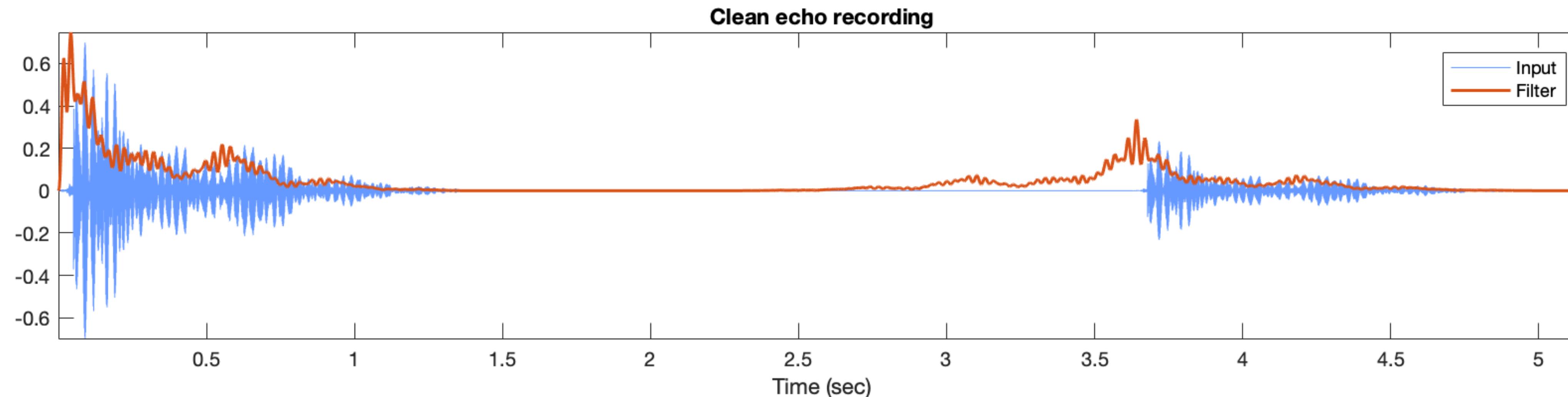
Some post-processing



- The matched filter output is a little messy
 - We mostly care about rough peaks, and its energy
 - So we rectify and lowpass filter

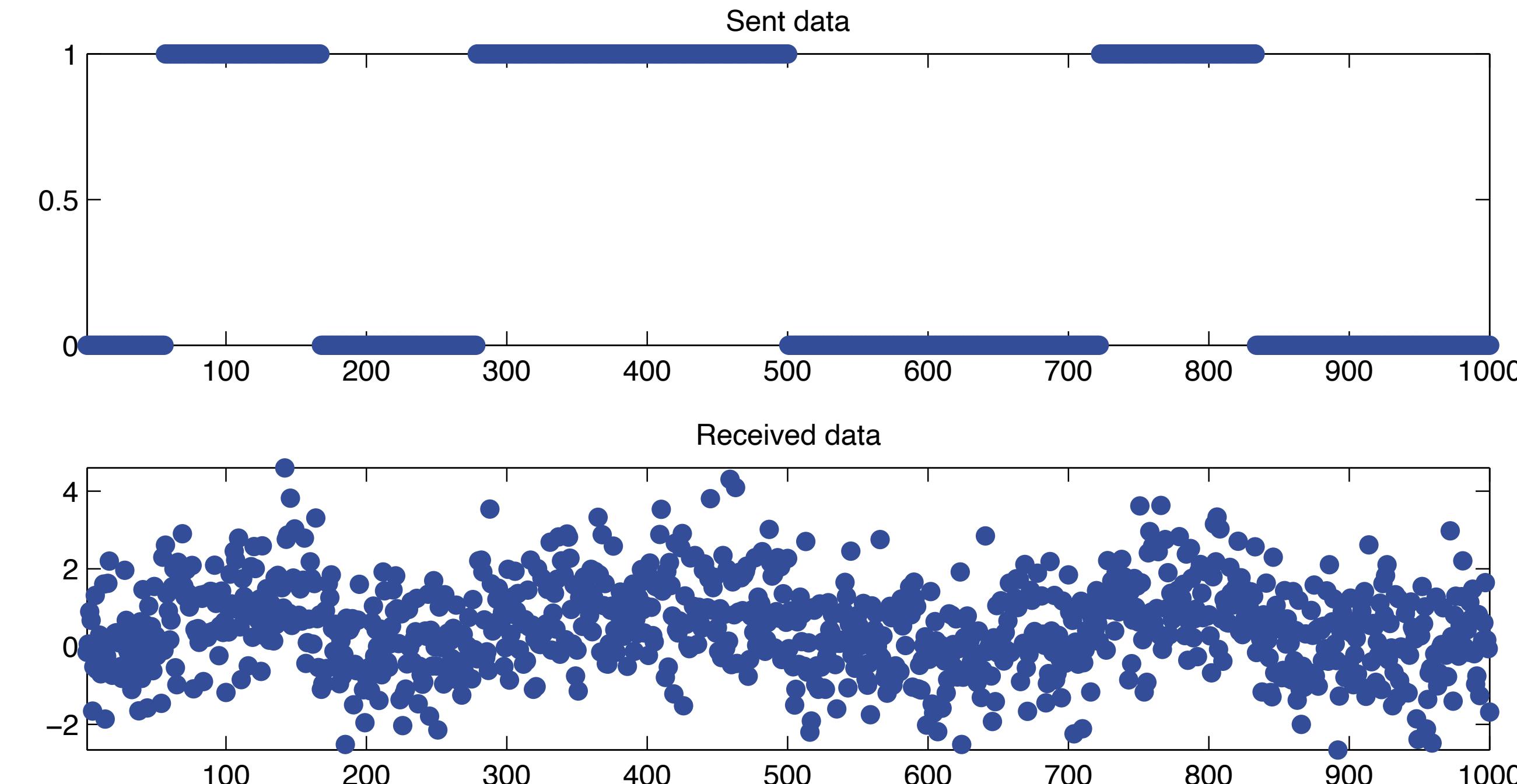


Most useful in noisy conditions



Simple communications example

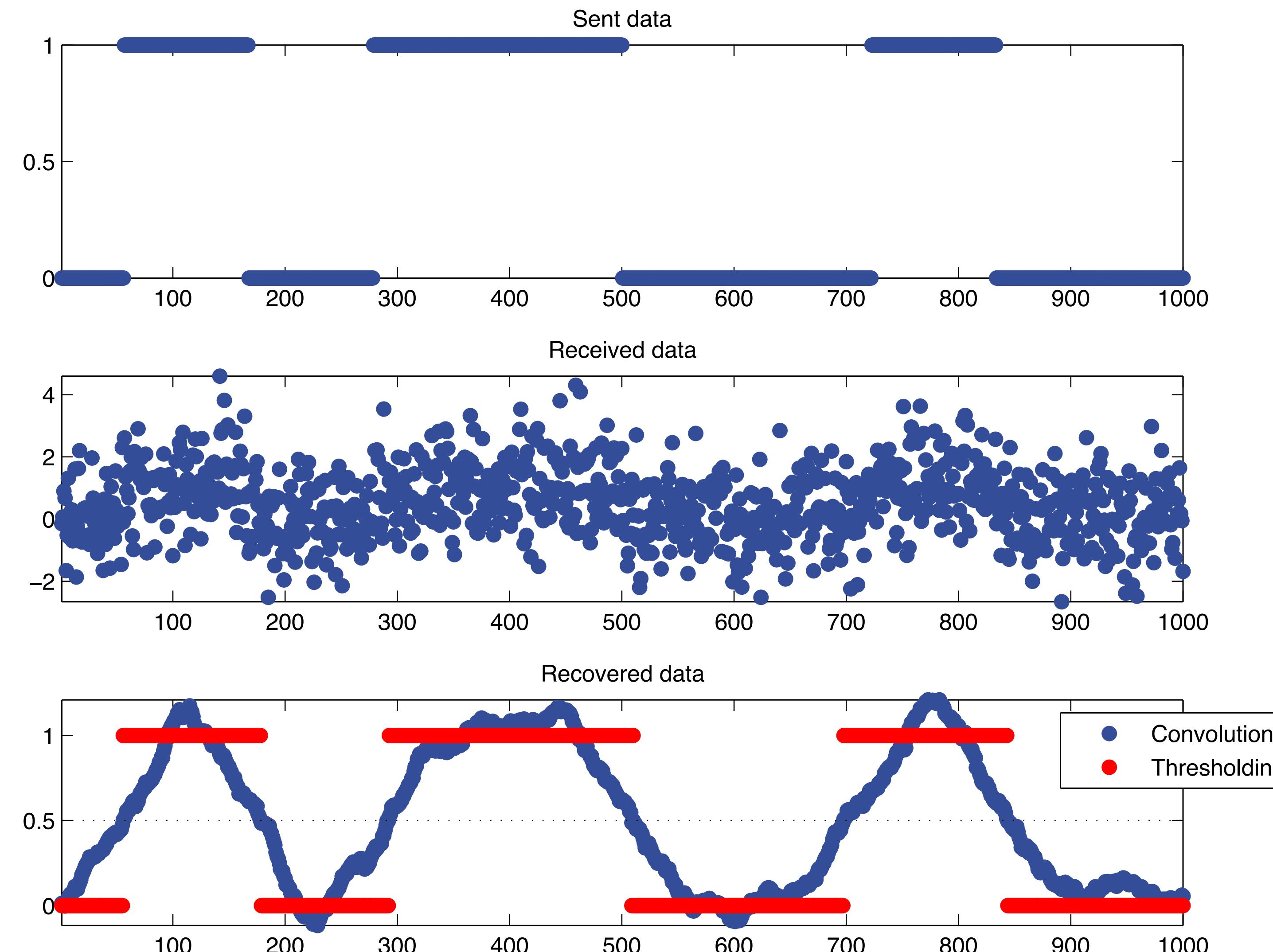
- You send 0/1's over the air
 - But you pick up some noise
- How do you recover the original data?



Detecting the 1's

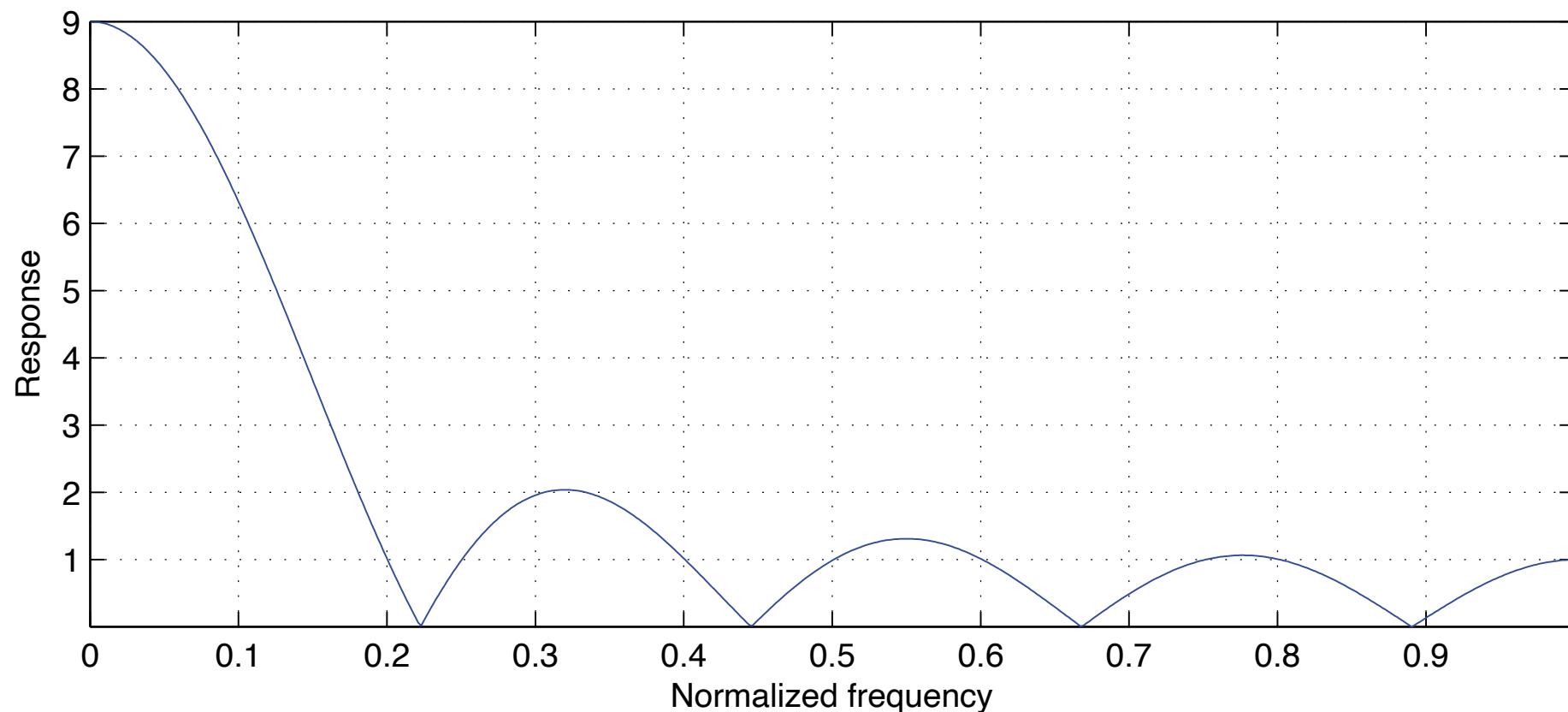
- Make a template for them:
 - $x = [1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1]$
 - This will detect a series of “1” symbols
- Convolve template with the input
 - And measure when the result is above threshold
 - E.g. greater than 0.5
 - Where it is greater than 0.5 we have a series of “1”'s

Result



An interesting connection

- What is this filter?
 - $\mathbf{x} = [1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1]$
- It's a lowpass filter
 - We effectively remove the high frequencies (i.e. the noise)
- Two ways to interpret this operation
 - A template to match our signal
 - Or a denoising filter



Computational considerations

- The sliding dot product is slow
 - N multiplications/additions per input sample
 - No fun with large templates
- Fortunately we can use the Fast Fourier Transform!
 - Convolutions can be performed efficiently with the FFT

FFT-based matched filtering

- Convolution in the time domain maps to element-wise multiplication in the frequency domain

$$\mathbf{d} = \mathbf{x} * \mathbf{y} \Rightarrow \mathbf{F} \cdot \mathbf{d} = (\mathbf{F} \cdot \mathbf{x}) \odot (\mathbf{F} \cdot \mathbf{y})$$

- But we have three sizes here: template, input, output
 - template is length N , input is M , output is $M+N-1$
 - What do we use?

Approach 1

- Zero pad everything to the largest size ($M+N-1$)
 - Ensures that we have enough space to store the output
 - Doesn't change the result since we convolve with more zeros
 - Slightly redundant though
- Example: $N = 1,000, M = 10,000,000$
 - Time domain takes about 10 GFLOPs
 - FFT convolution takes about 200 MFLOPs

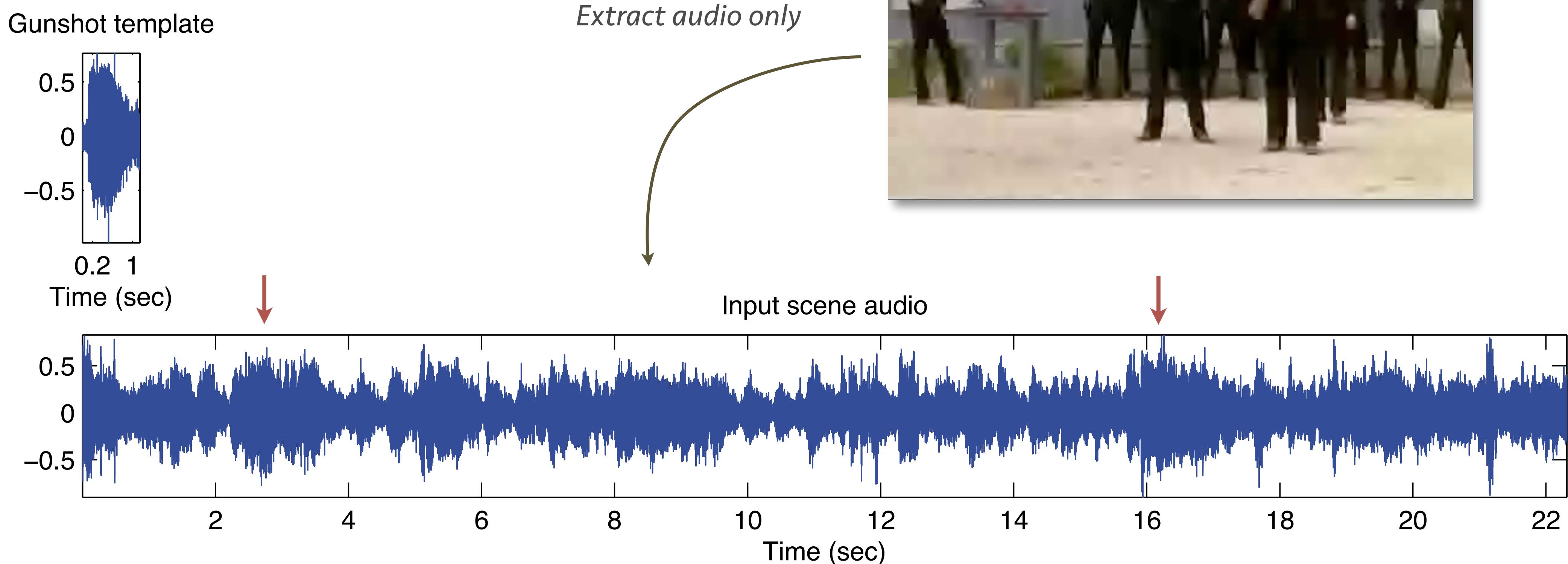
Approach 2

- Take small snippets of the input, convolve them with the template and add them together (convolution is linear!)
- How small?
 - Same size as template (N) to avoid excessive zero padding
 - But, the output will be $2N-1$, so we need to zero pad as much
- Example speedup, $N = 1,000, M = 10,000,000$:
 - Time domain: 10 GFLOPS
 - FFT: 200 MFLOPs
 - "Fast convolution": 60 MFLOPs

Matched filtering for event detection

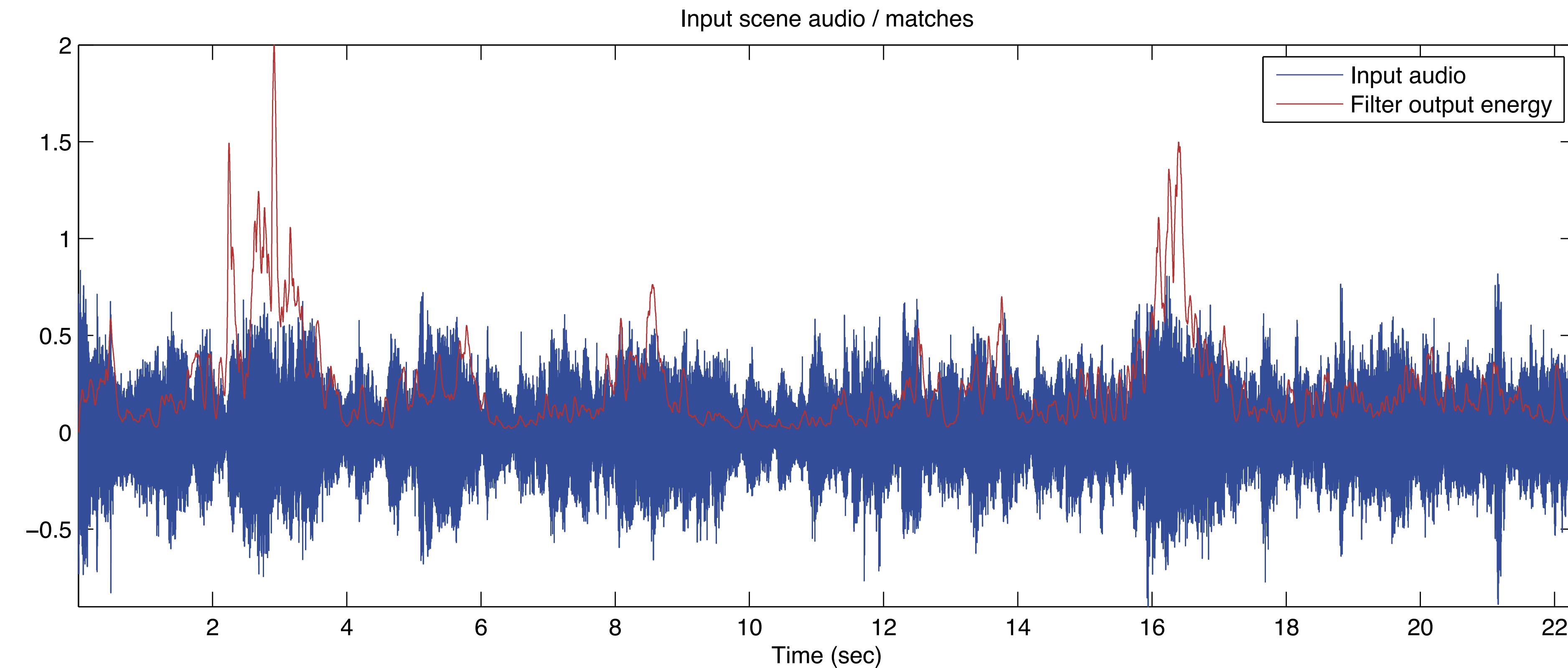


- Let's try to detect gunshot sounds in a video



Matched filter result

- Strong peaks where gunshots took place
 - Processing: Time domain: 1.8 sec, Fast Conv: 0.06 sec

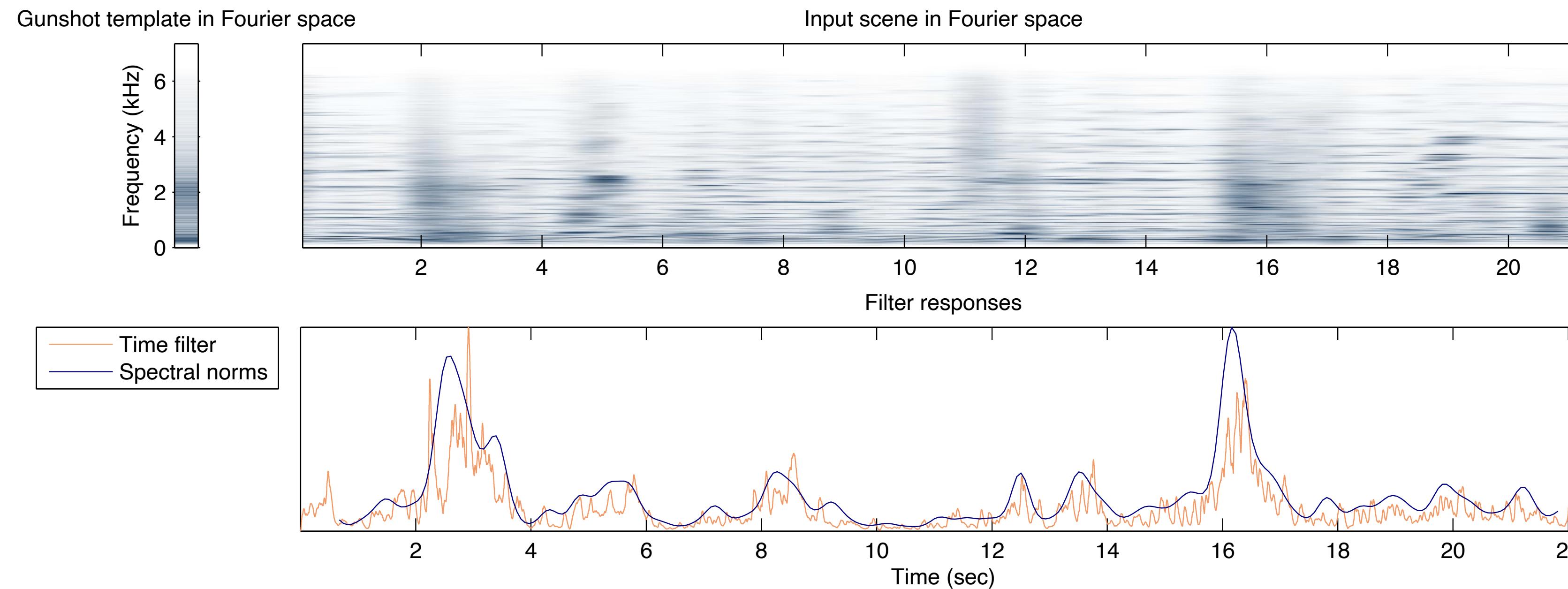


Going even faster

- The template and each time frame is a vector

$$\mathbf{t} * \mathbf{s} = \mathbf{F}^H \cdot [(\mathbf{F} \cdot \mathbf{t}) \odot (\mathbf{F} \cdot \mathbf{s})] = \mathbf{F}^H \cdot (\mathbf{f}_t \odot \mathbf{f}_s)$$

$$\|\|\mathbf{t} * \mathbf{s}\|\|^2 = [\mathbf{F}^H \cdot (\mathbf{f}_t \odot \mathbf{f}_s)]^H \cdot [\mathbf{F}^H \cdot (\mathbf{f}_t \odot \mathbf{f}_s)] = (\mathbf{f}_t \odot \mathbf{f}_s)^H \cdot (\mathbf{f}_t \odot \mathbf{f}_s) = \|\|\mathbf{f}_t \odot \mathbf{f}_s\|\|^2$$



Generalizing to two dimensions

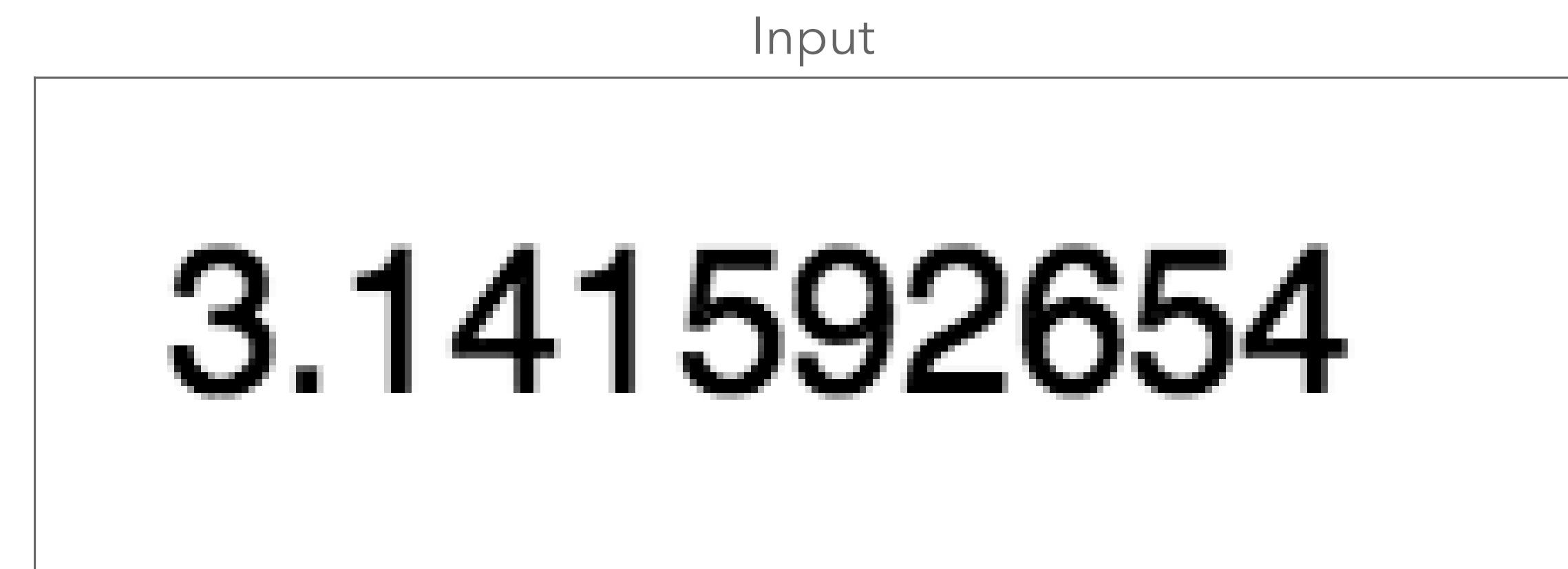
- In 1D the template was time-reversed
 - In 2D we flip both dimensions

$$d(i, j) = \sum_k \sum_l x(k, l) y(i + k, j + l)$$

- We now slide across both dimensions
 - E.g. left-to-right for all vertical offsets
 - FFT speedup still holds
 - But we now use 2D FFTs
 - Same caveats as before apply here as well (zero padding, etc.)

Digit recognition example

- Given the following input



- Find the digits by analyzing the image

Collect templates

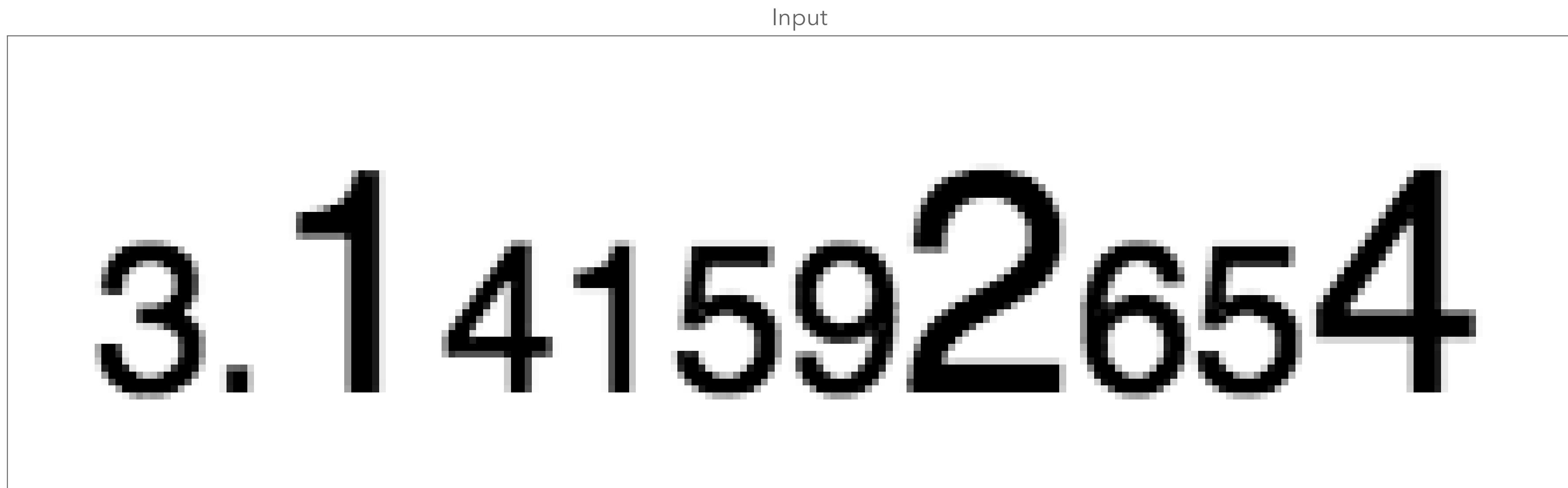
- Create a set of templates for all digits

$$\mathbf{x}_1 = \boxed{1} \quad \mathbf{x}_2 = \boxed{2} \quad \dots$$

- Filter flipped versions with input image
 - Each template will peak at digit's position
- Live demo!

A new problem

- What if we have a change in size?



Broadening the search

- Resize the the templates to more sizes and run template matching again
 - How much resizing?
- This is N times more computation
 - When considering N scales
- Live demo

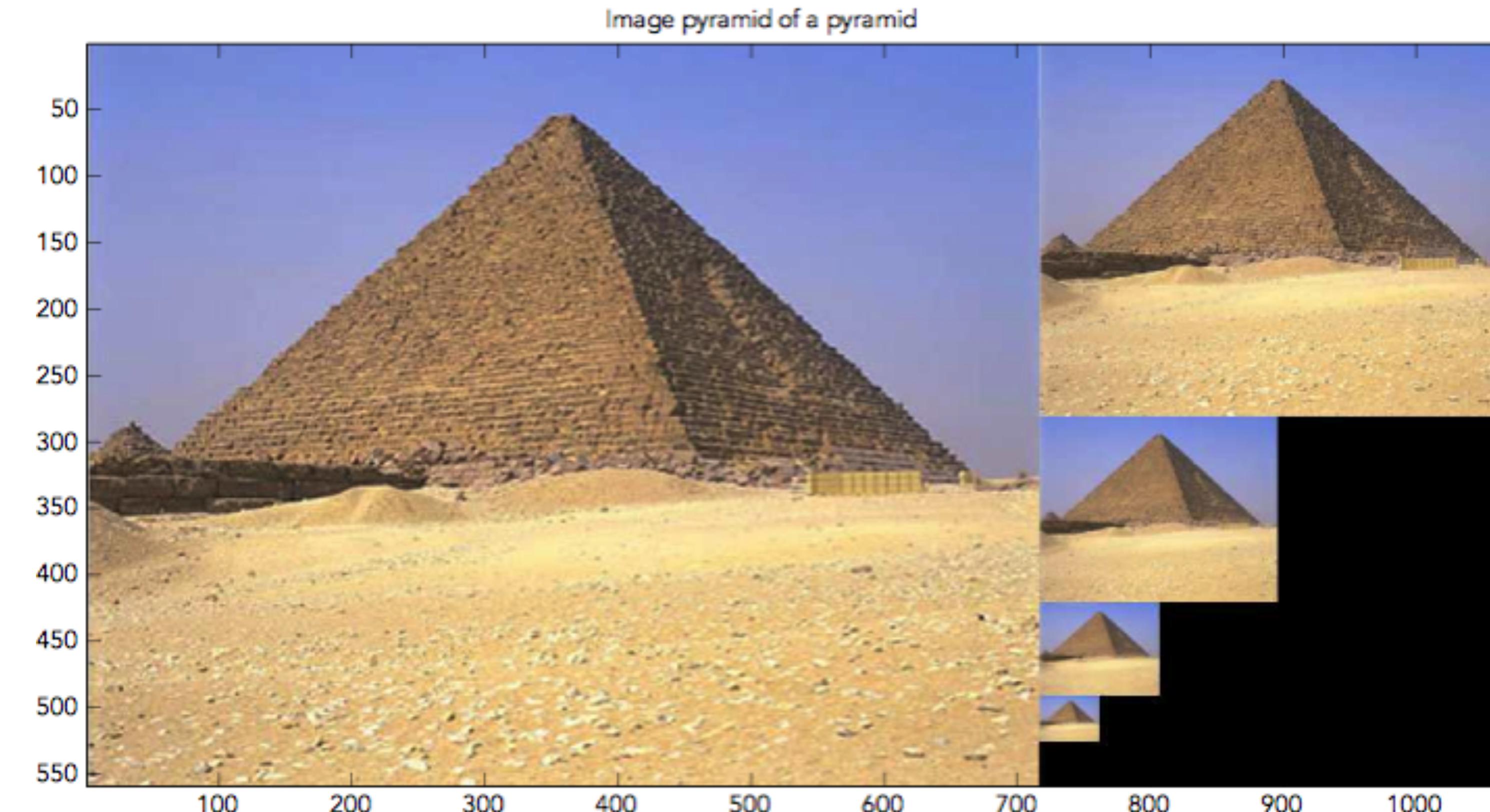
$$\mathbf{x}_1^L = \begin{matrix} 1 \end{matrix} \quad \mathbf{x}_2^L = \begin{matrix} 2 \end{matrix} \dots$$

$$\mathbf{x}_1^M = \begin{matrix} 1 \end{matrix} \quad \mathbf{x}_2^M = \begin{matrix} 2 \end{matrix} \dots$$

$$\mathbf{x}_1^S = \begin{matrix} 1 \end{matrix} \quad \mathbf{x}_2^S = \begin{matrix} 2 \end{matrix} \dots$$

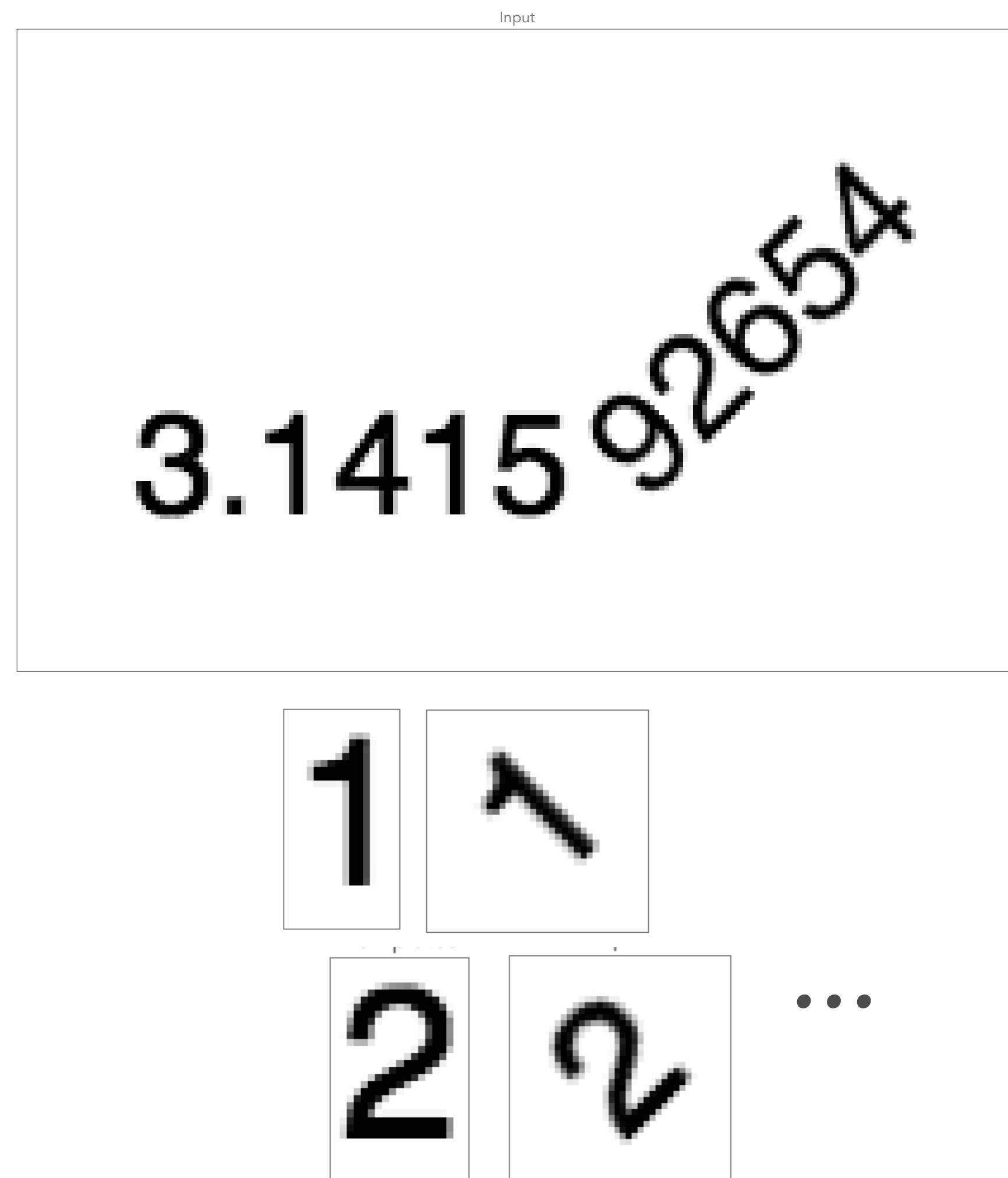
Image pyramids

- Scaling the templates is not too efficient
 - Instead resample the input at multiple scales, not the template



Rotation?

- Same as before
 - Brute force search over all potential rotations
- One more multiplicative factor in computation

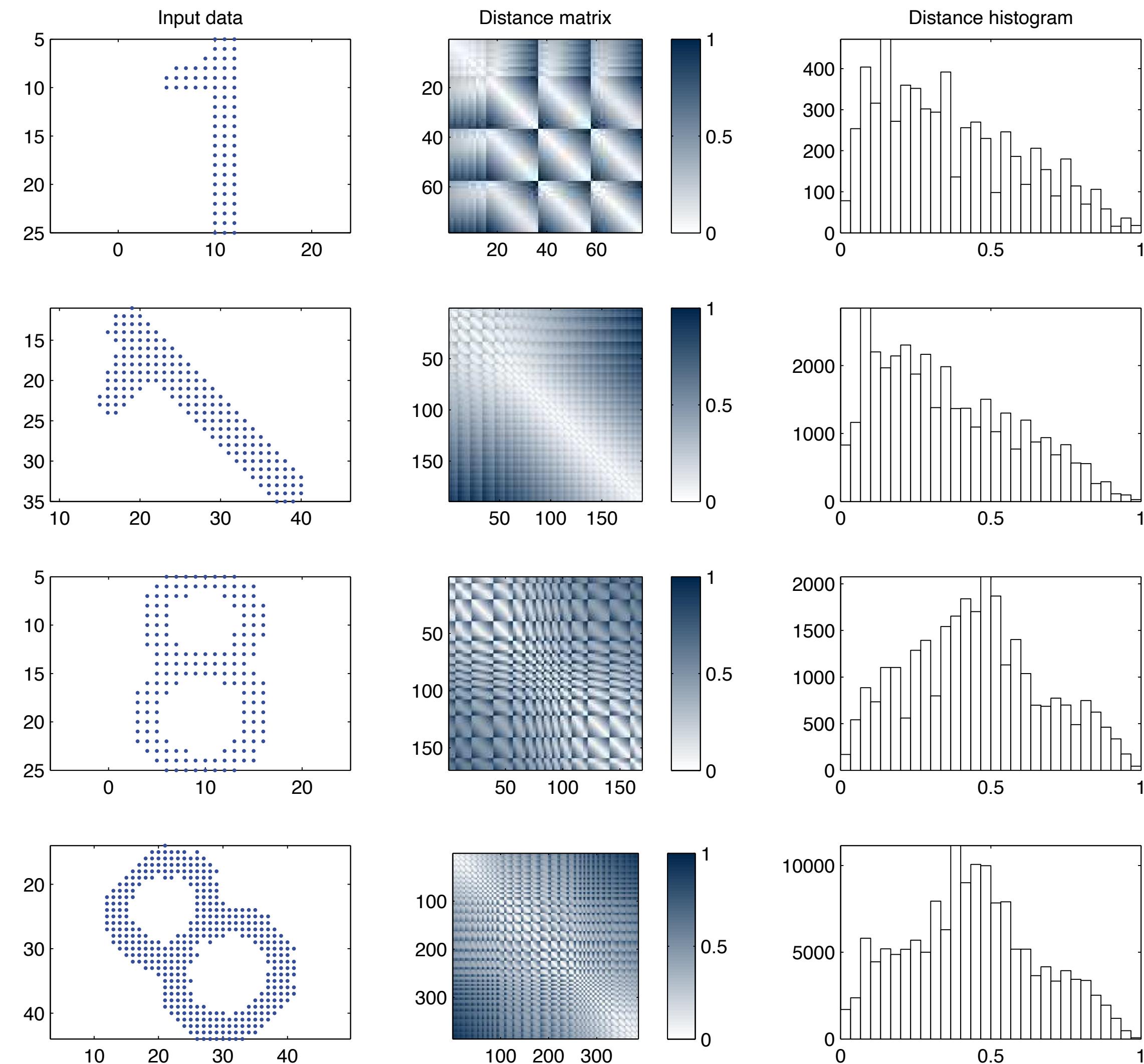


Some relief

- Rotation and scale and ..., that's a lot of searching that needs to be done!
- There are some tricks we can pull
 - e.g. radial filters for rotation
- Better features
 - Choose, e.g., a rotation invariant feature
- Lots and lots of papers ...
 - But some searching will be there

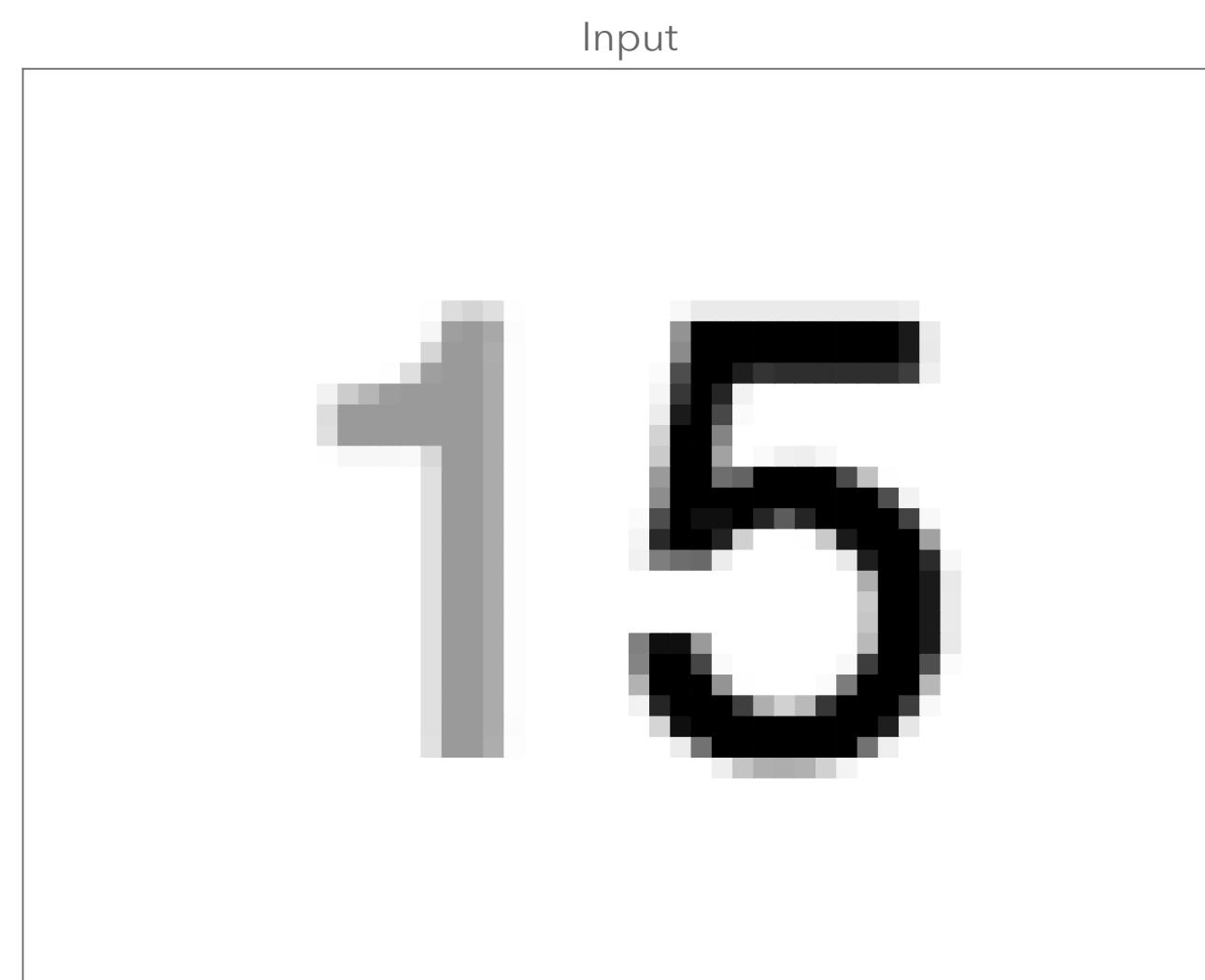
For example

- Treat pixels as points
 - Get their cross-distances
 - And normalize them
- The distances histogram is rotation- and scale-invariant
 - Do matching on that instead



One more problem ...

- What happened to the normalization?!
 - Remember that vectors were unit-length?
 - We have not enforced that!
- Here's a potential problem case

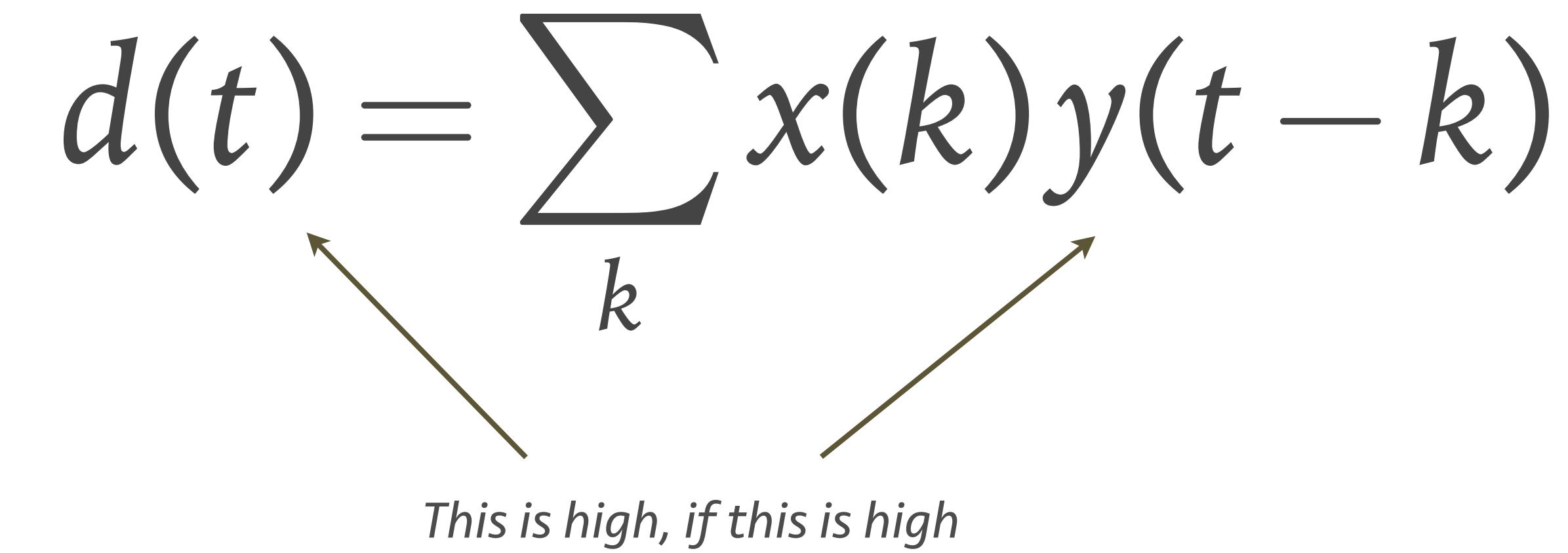


Problems with scaling

- Regular cross-correlation is:

$$d(t) = \sum_k x(k)y(t - k)$$

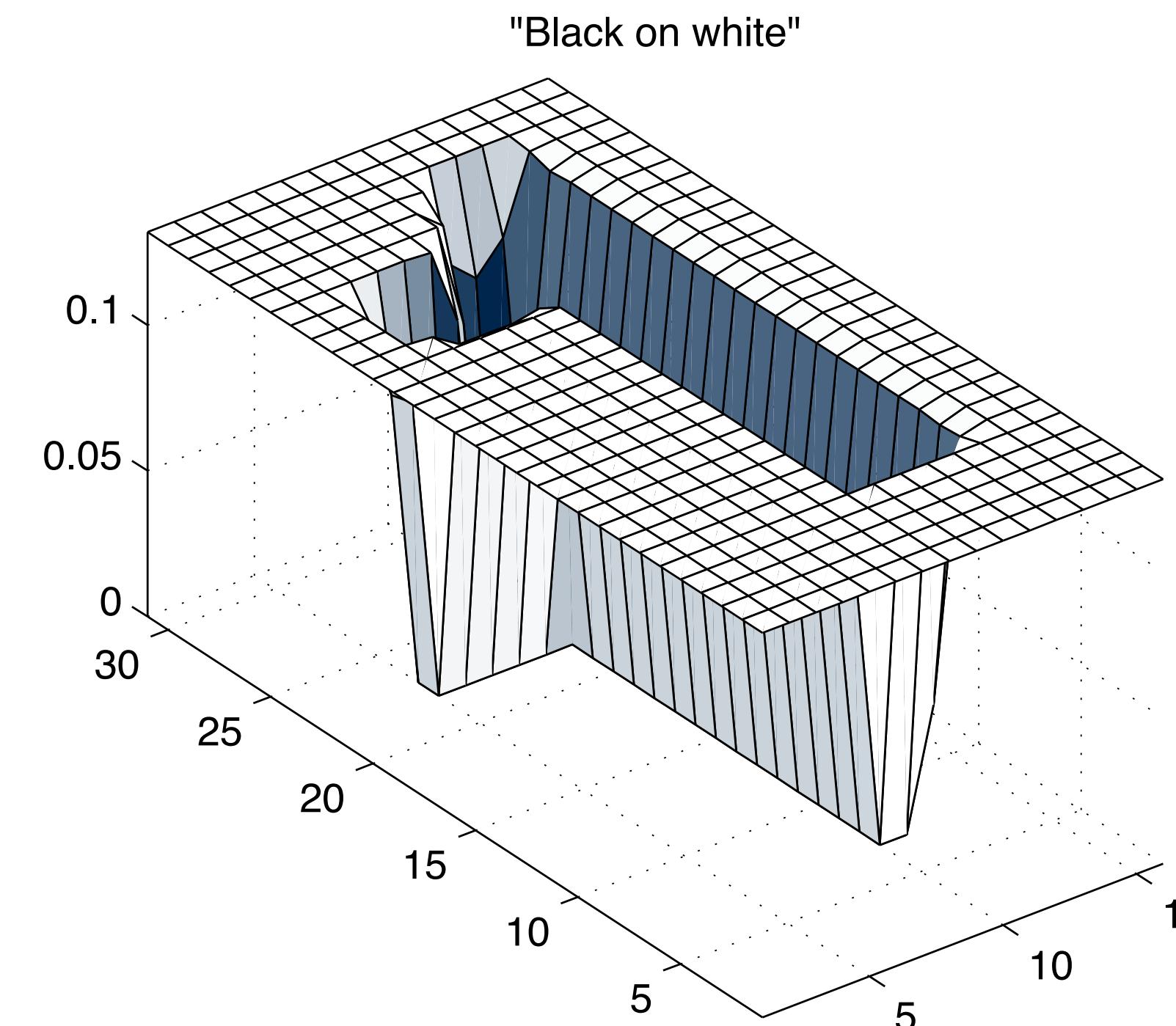
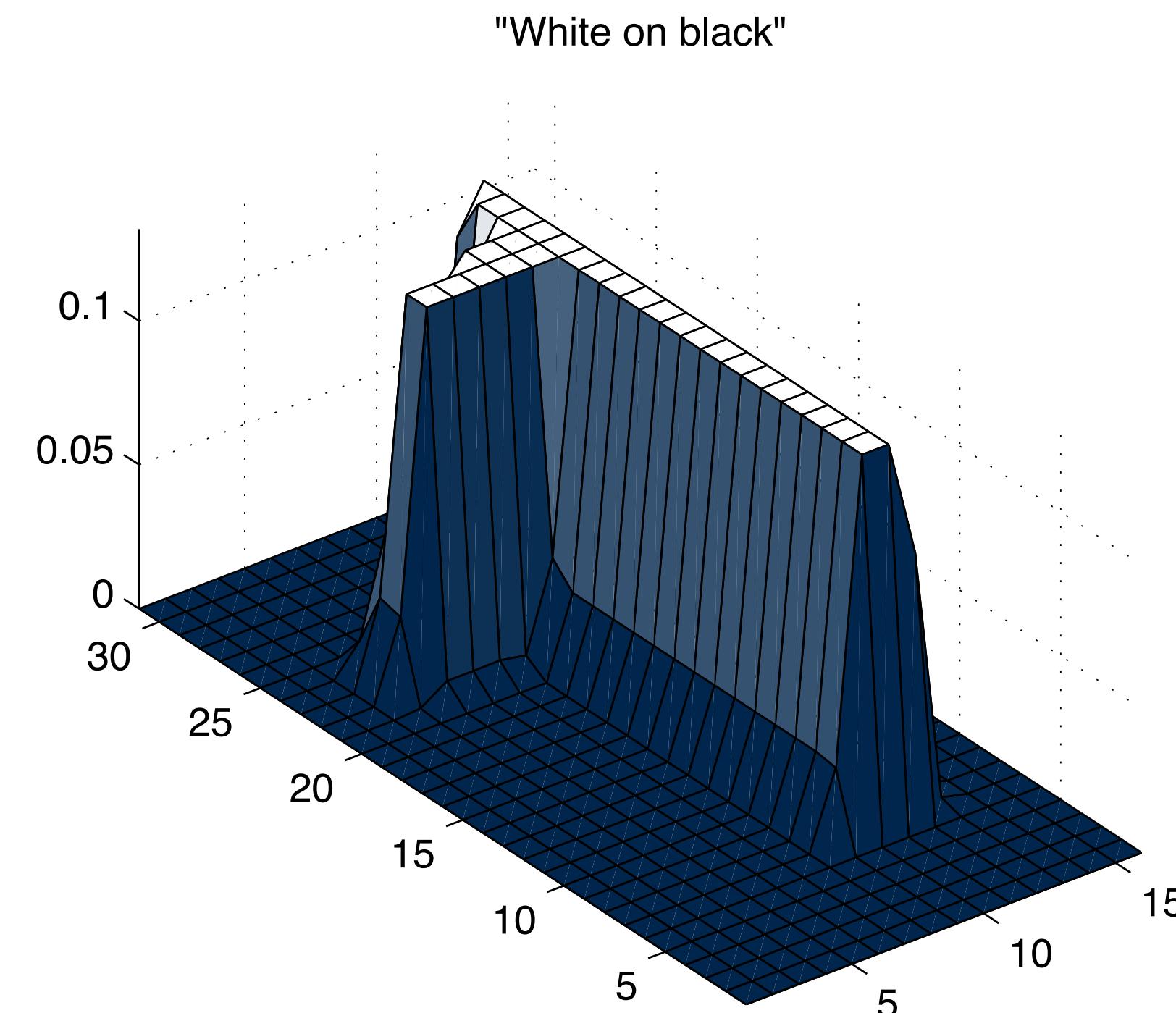
This is high, if this is high



- Output is highly dependent on the local magnitude of the input

In fact, I've been cheating all this time

- I used an inverted colormap (white:0, black:1)
 - I made sure that larger values were parts of interest
 - Do you see a new problem?



Solution 1: Normalize

- We really want:

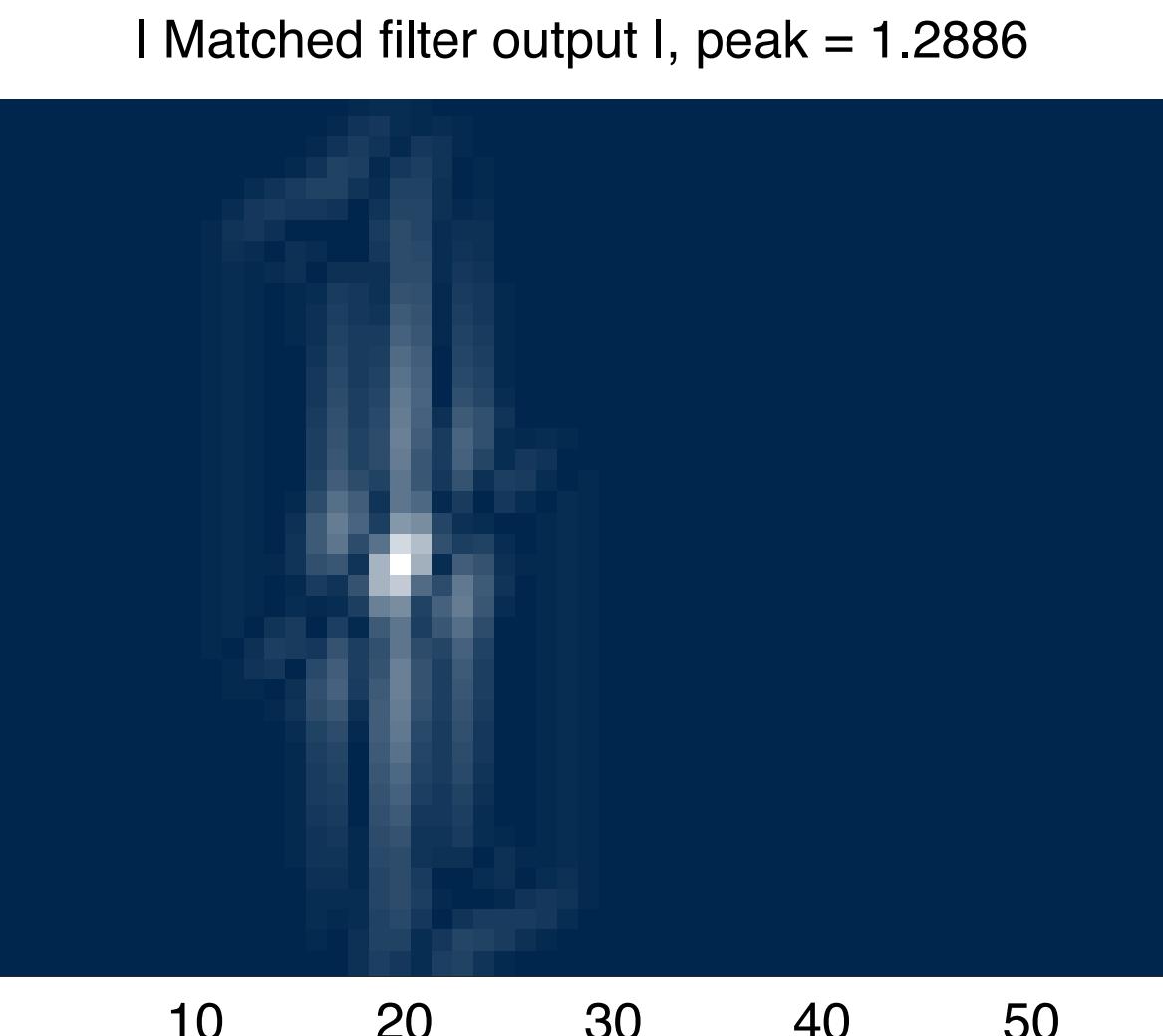
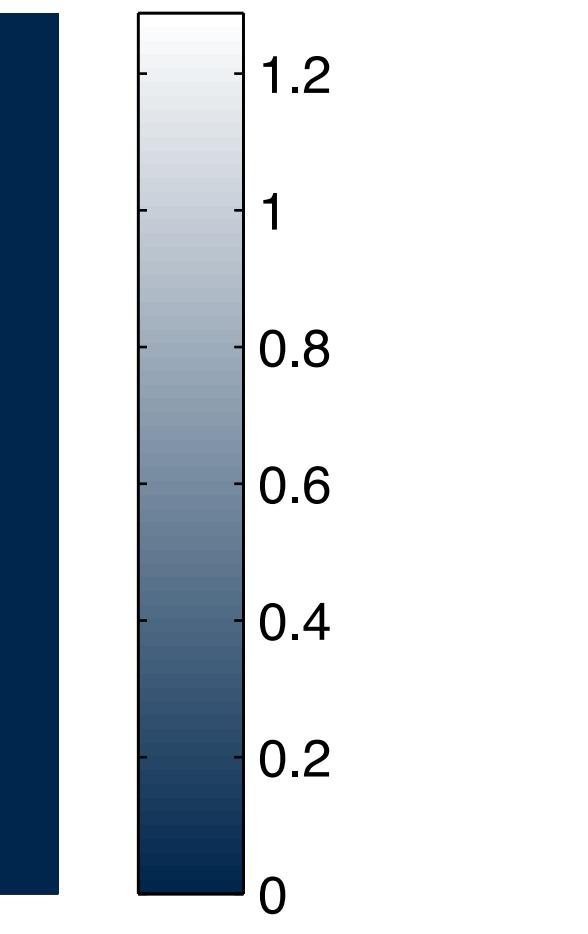
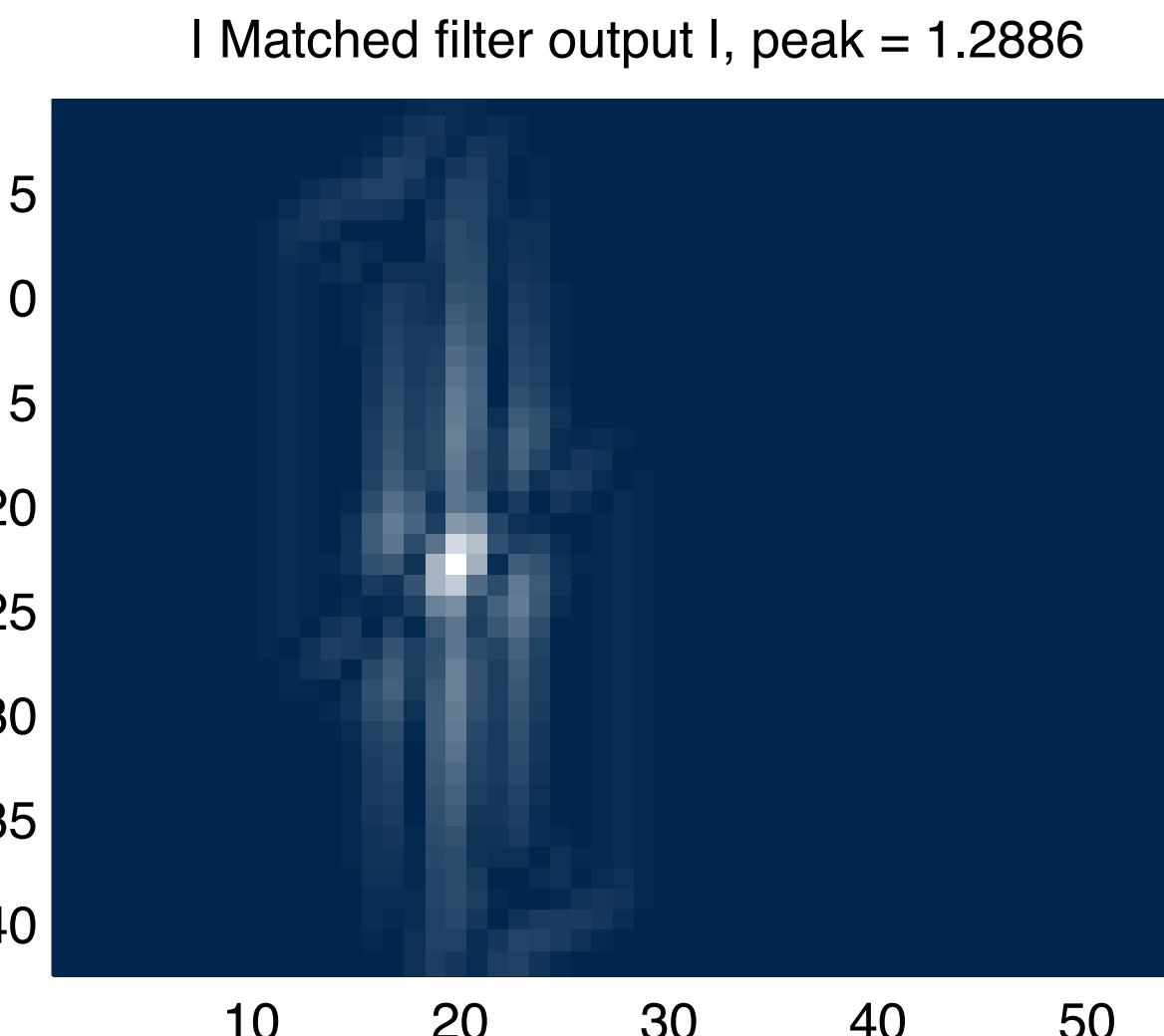
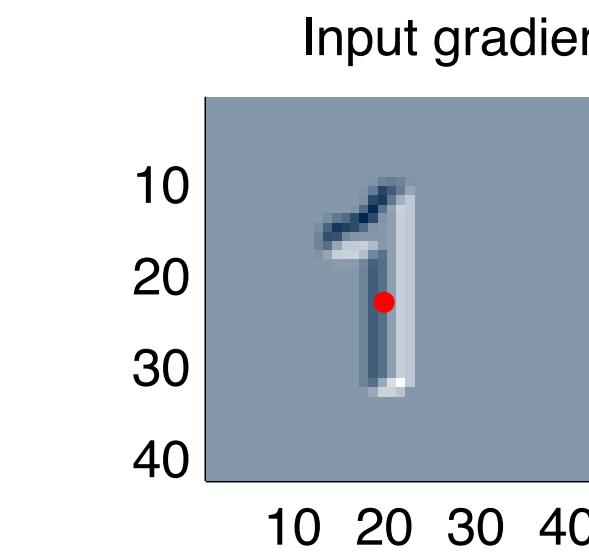
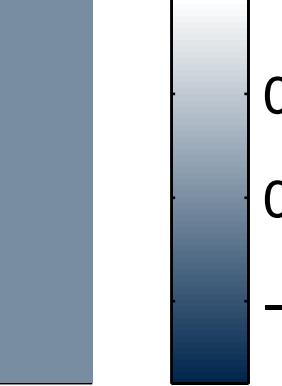
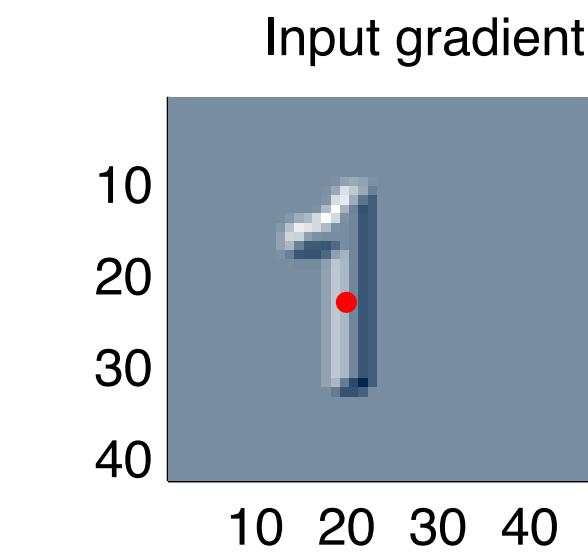
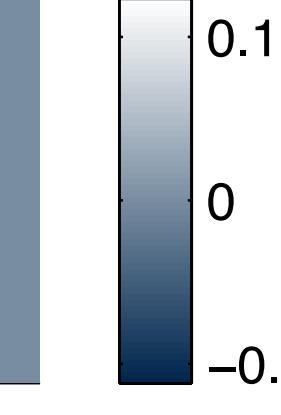
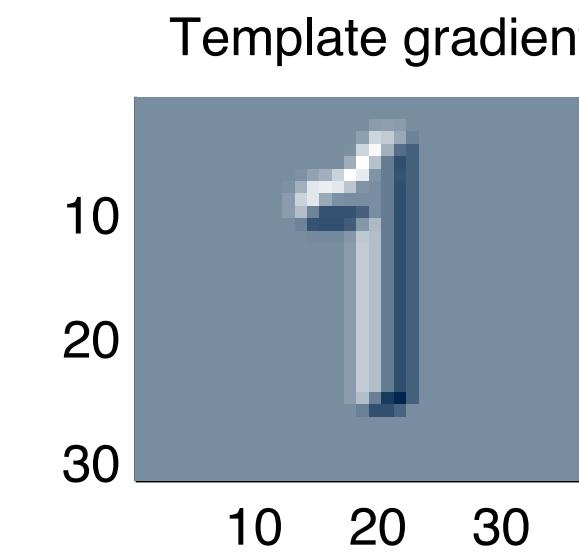
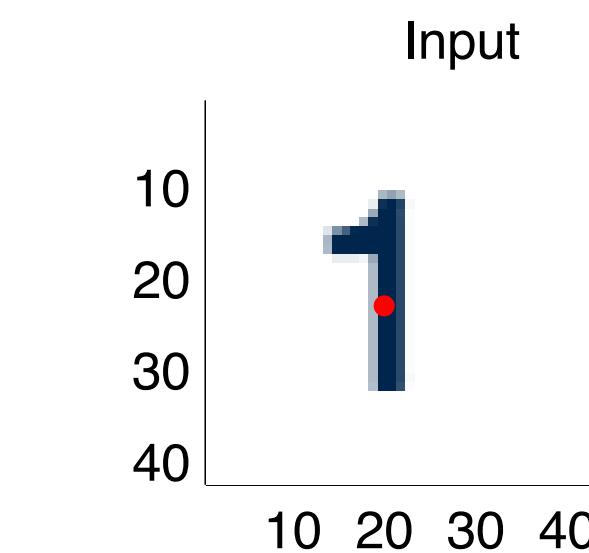
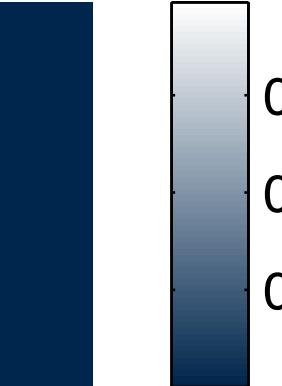
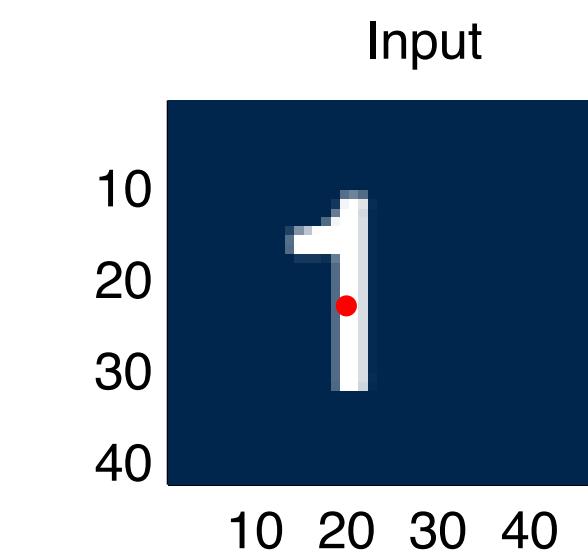
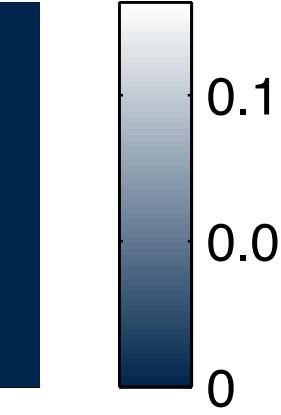
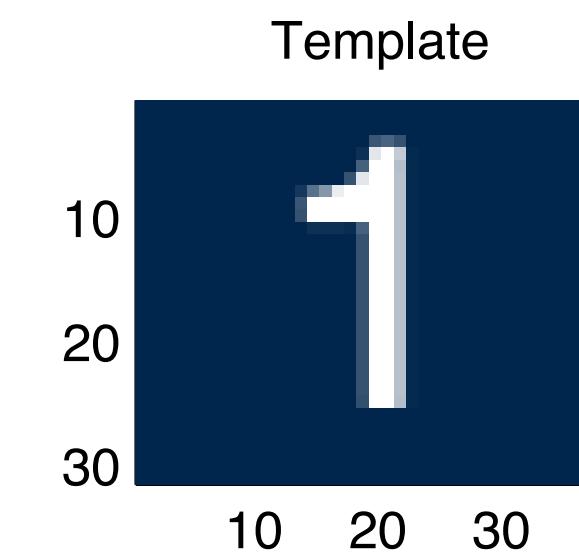
$$d(t) = \sum_k \frac{x(k)y(t-k)}{\sigma_x \sigma_{y(t)}}$$

- Where the denominator ensures a unit-length dot product with the template
- Likewise we can generalize to 2-D, etc.
- But FFT speedup is gone now

Solution 2: Gradient filtering

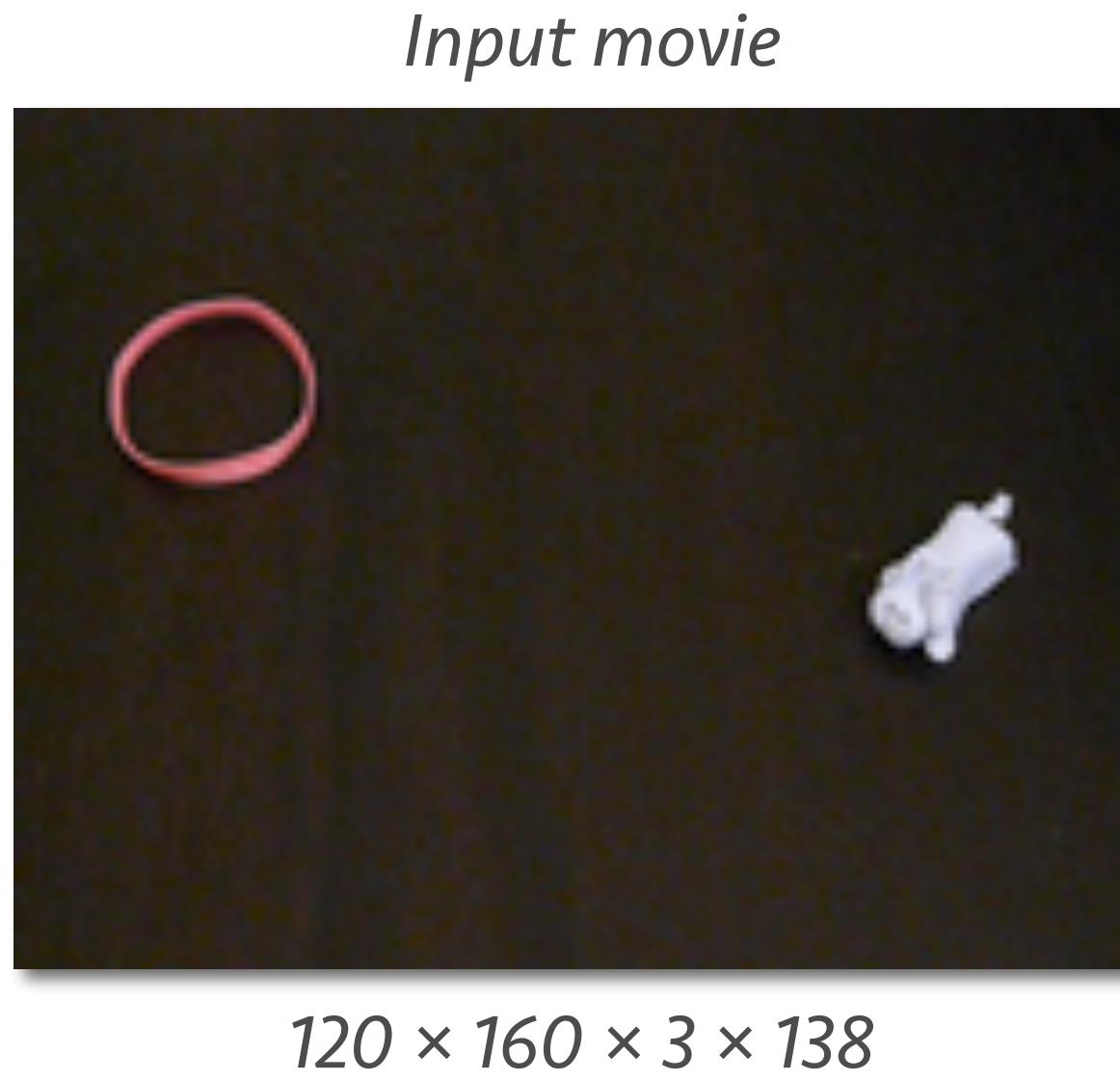
- Detect what matters
 - Remember what your retina likes? (hint: edges)
- Perform detection on the gradient image
 - This abstracts the color and local intensity
- Much more robust detector (for images at least)

Example



Movie example

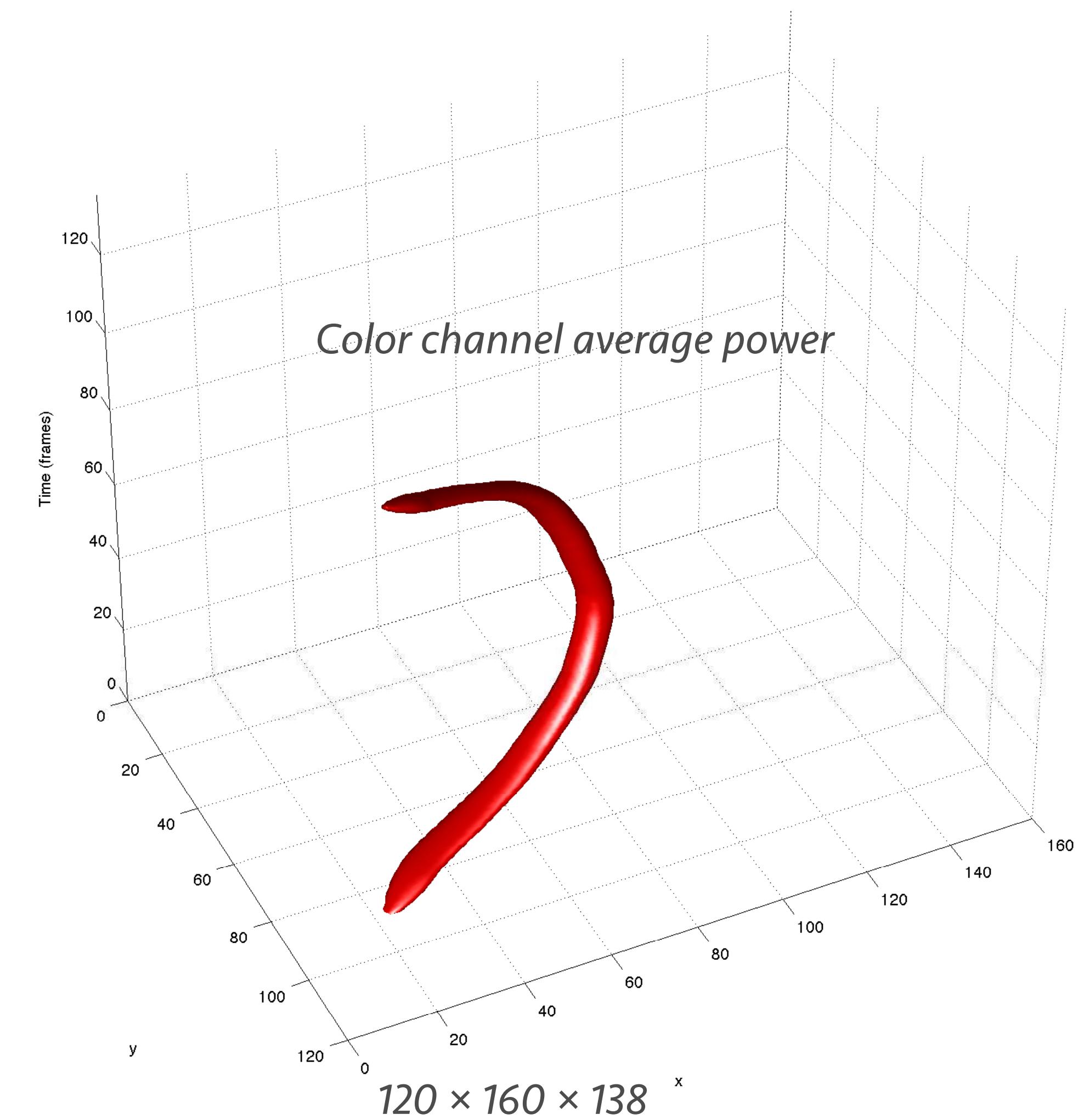
- Track the ball movement
 - Convolve in 4D space



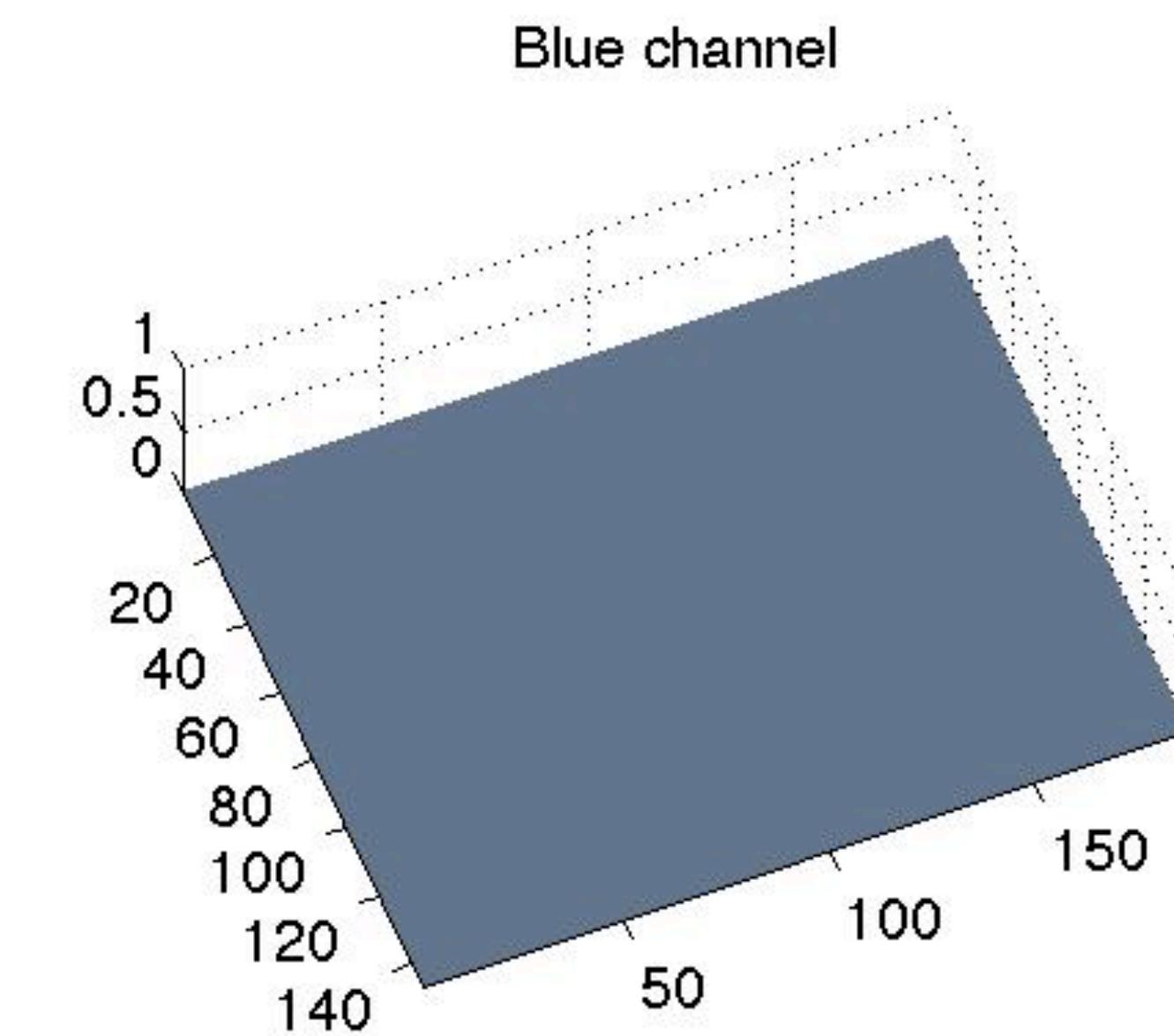
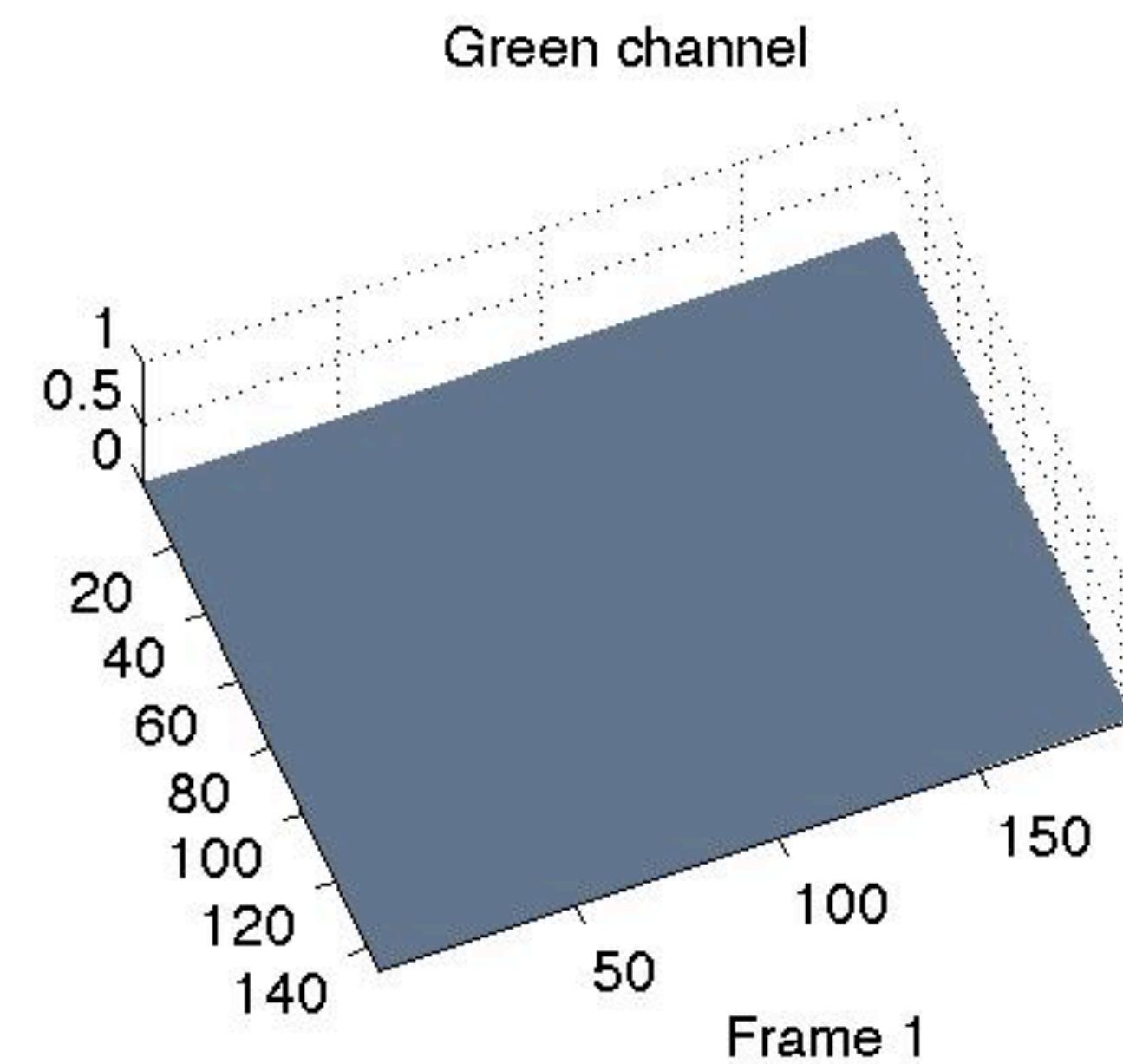
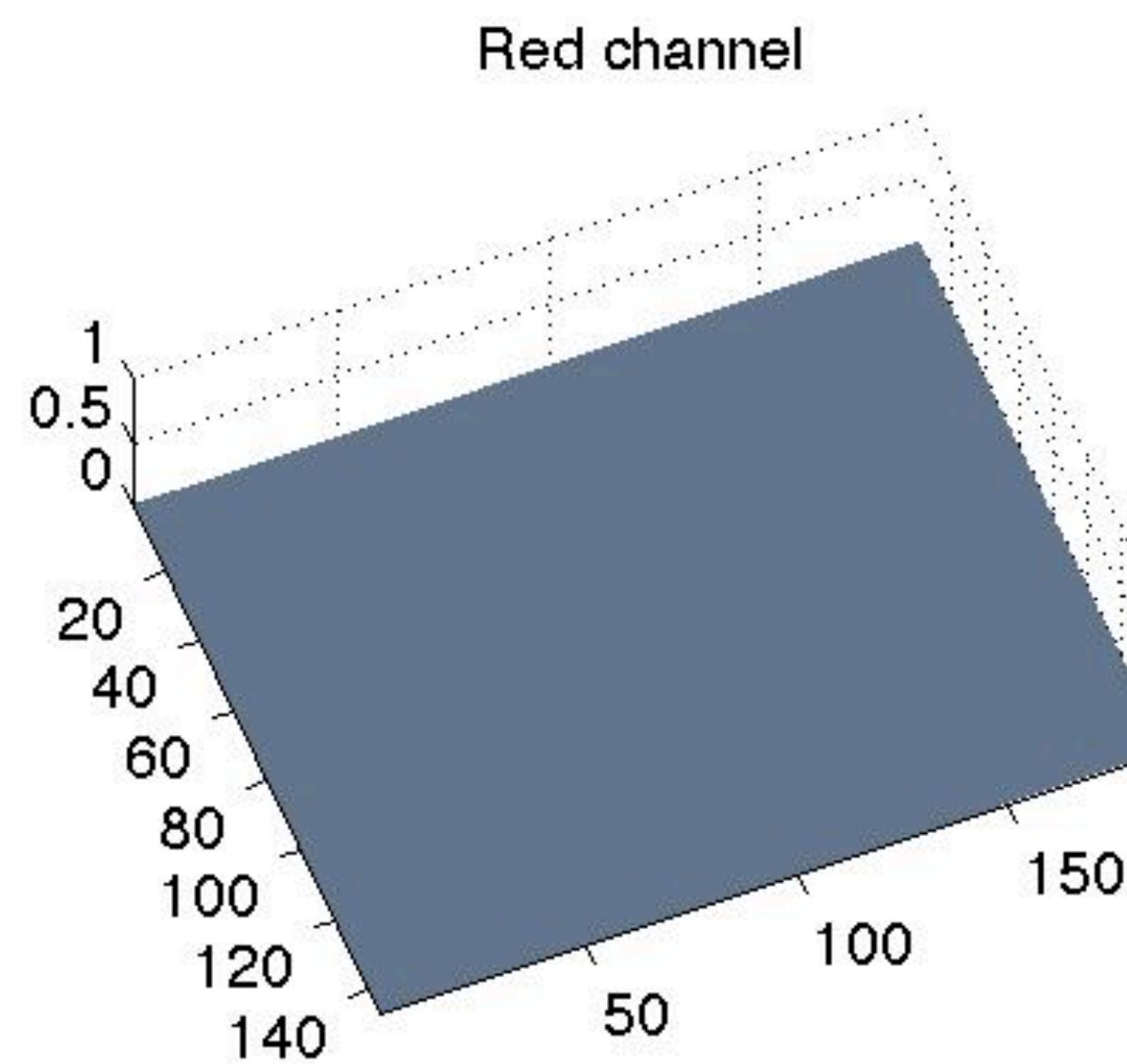
Ball template

$$\ast \quad \quad =$$

$28 \times 29 \times 3$



Each channel separately



Frame 1

Template matching applications

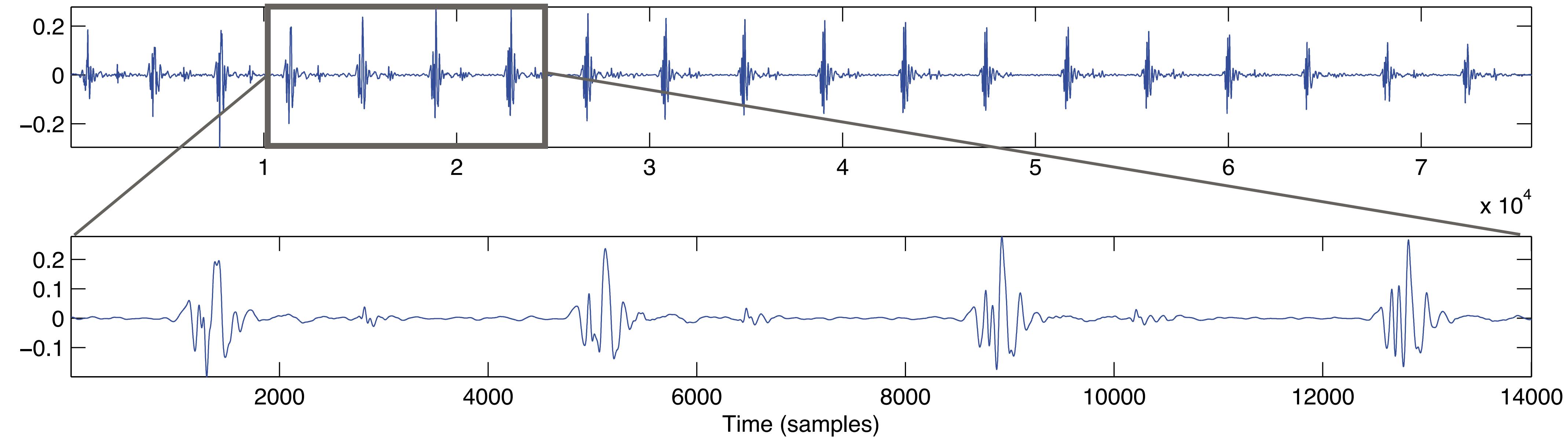
- Popular tool in computer vision
 - face/car/pedestrian/footballer detection
 - Templates are example images of the above
- Good for event detection in audio
 - gunshot detection, room measurements
- Bio/geological/space/underwater signals etc ...

Matching without a template

- Sometimes we don't know the template
 - But we know it appears a lot
- We can use *auto-correlation* to find structure
 - Use the full input as a template
- Peaks will denote repeating elements

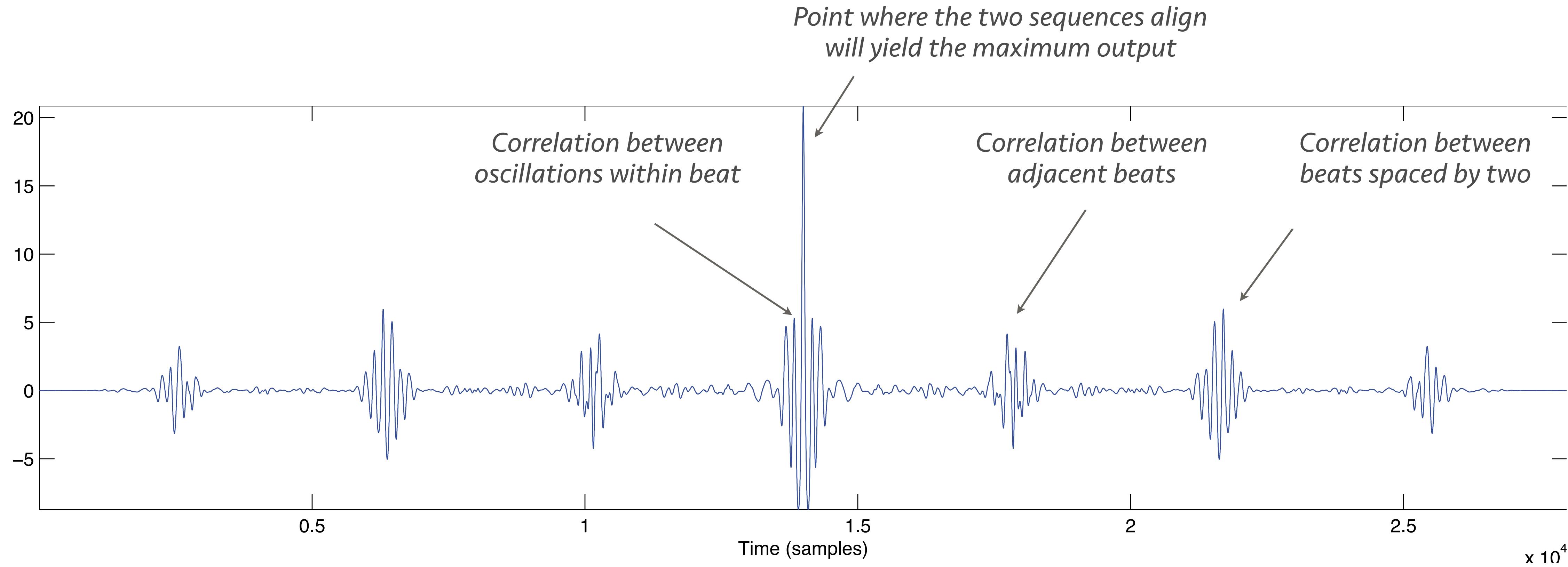
Back to the heartbeat example

- We don't know the exact template, but there is a pattern that is repeating in time
 - We can still find structure regardless



Autocorrelation

- Use the entire input as a template
 - Match the input on itself



Probing matching criteria

- Revisiting the starting point
 - Suppose we instead tried to find the minimal distance between the template and the input

$$D(m, n) = \sum_i \sum_j |x(i - m, j - n) - y(i, j)|^2$$

- This is a simple Euclidean distance

From Euclidean to dot product

- Expanding the square in the distance formula we get:

$$\begin{aligned} D(m, n) = & \sum_i \sum_j |y(i, j)|^2 + \sum_i \sum_j |x(i, j)|^2 \\ & - 2 \sum_i \sum_j y(i, j) x(i - m, j - n) \end{aligned}$$

- If we assume a locally constant input y
 - Minimizing D , maximizes the last term (which is the dot product!)
 - Using the dot product we are ultimately minimizing a Euclidean distance between the template and the input

Getting to a likelihood

- The Euclidean distance also implies a Gaussian

$$P(\mathbf{x}; \mathbf{y}) \propto e^{-\frac{1}{2}(\mathbf{x}-\mathbf{y})^\top \cdot (\mathbf{x}-\mathbf{y})}$$

- We can now (sort of) get a likelihood of match
- We can use various other metrics to define distance, each implying a different distribution
 - More on that later

And don't forget features!

- We focused on detection on raw data
 - We can instead convolve on the feature weights
 - We sort of did that with the DFT and the gradients
- This can buy us noise robustness, invariance, and other desirable properties
 - You will rarely operate on raw data
- All of the previous theory applies here as well

Recap

- Detection of interesting elements
- Matched filtering
 - 1D to 2D
 - Scale/rotation invariance
 - Normalized cross-correlation
 - Detection on gradients
 - Autocorrelation
- From filters to likelihoods

Next lecture

- Linear classifiers
- Discriminant models
- Multi-class recognition
- It won't be next week, I'll be at the MLSP conference!
 - Next lecture will be on Sept 26
 - The calendar on Teams is appropriately updated for these things

Reading

- Fast normalized correlation
 - <http://scribblethink.org/Work/nvisionInterface/nip.pdf>
- Transform invariant templates examples (optional)
 - https://link.springer.com/content/pdf/10.1007%2F978-3-540-77129-6_13.pdf
 - <http://www.lps.usp.br/~hae/software/forapro/index.html>