HAAR-like features for images

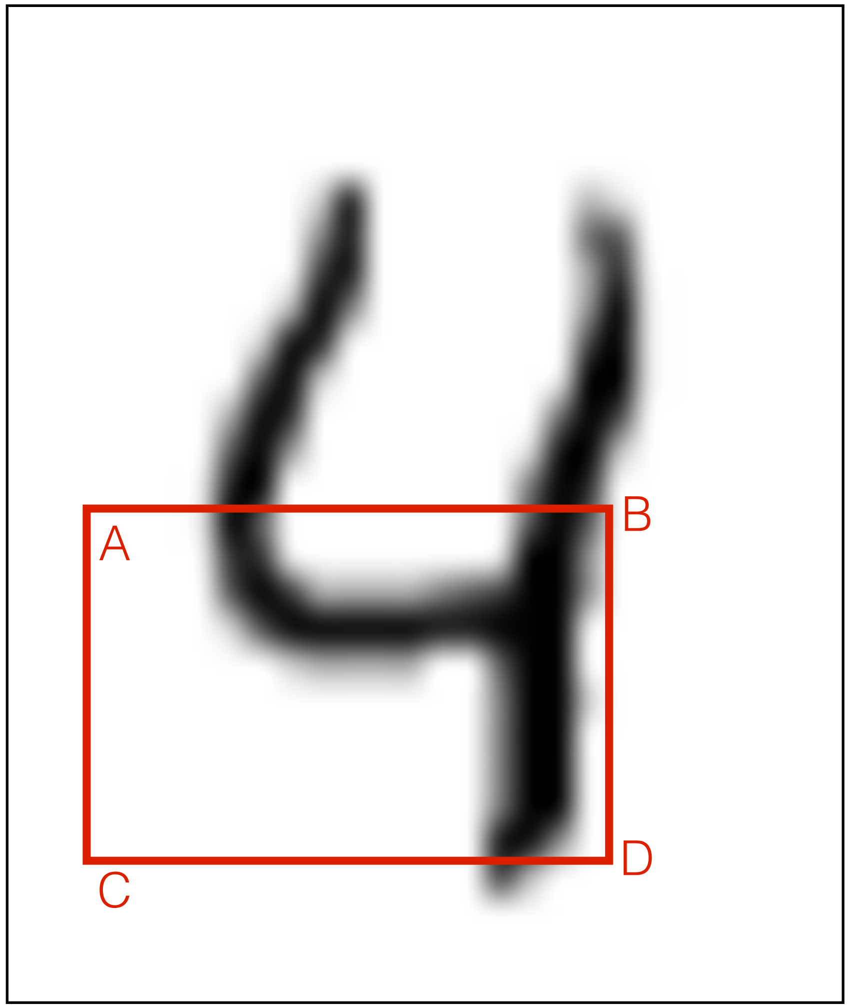
Images

- digit images are scanned hand written digits

Digit scan dataset

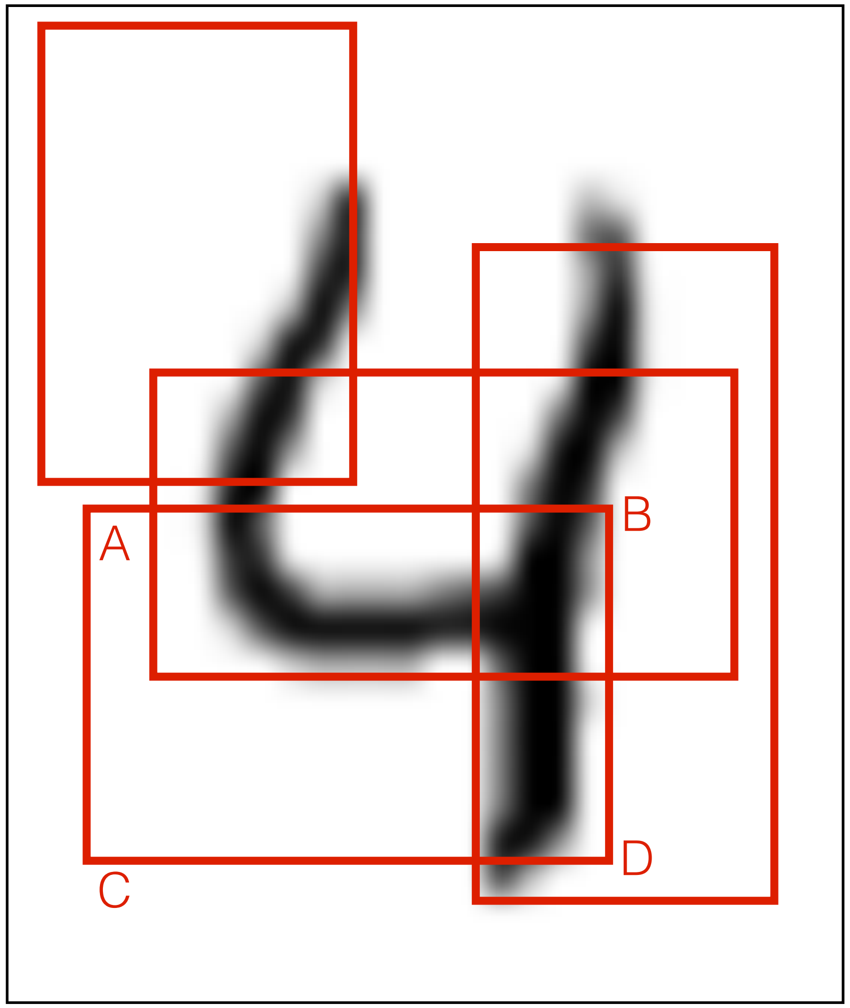
- 60,000 scans
- 10 classes : 0,1,2,...,9
  - roughly uniform distributed
- each scanned image 28x28 pixels square
- comes split into (train, test)
  - no cross validation
- very learnable: most algorithms score 5% or less error
- http://yann.lecun.com/exdb/mnist/

Rectangle black level



- rectangle ABCD can act like an image “mask” : it selects/cuts that rectangle out of an image
  - or of any image

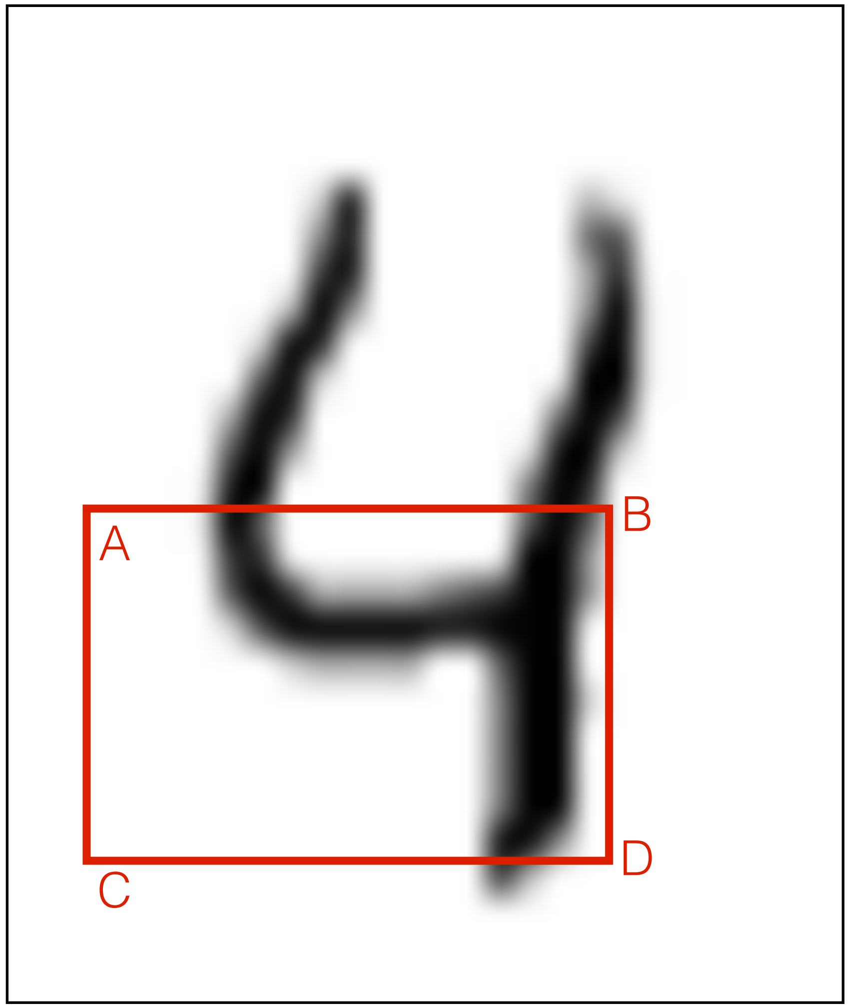
Rectangle black level



- a given set S of rectangles cuts S different masks for an image

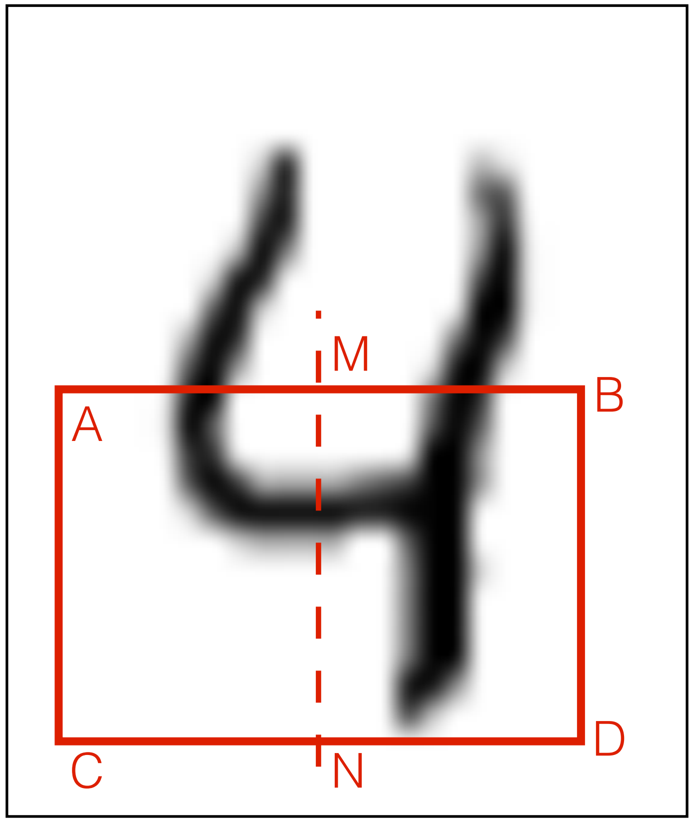
Rectangle black level

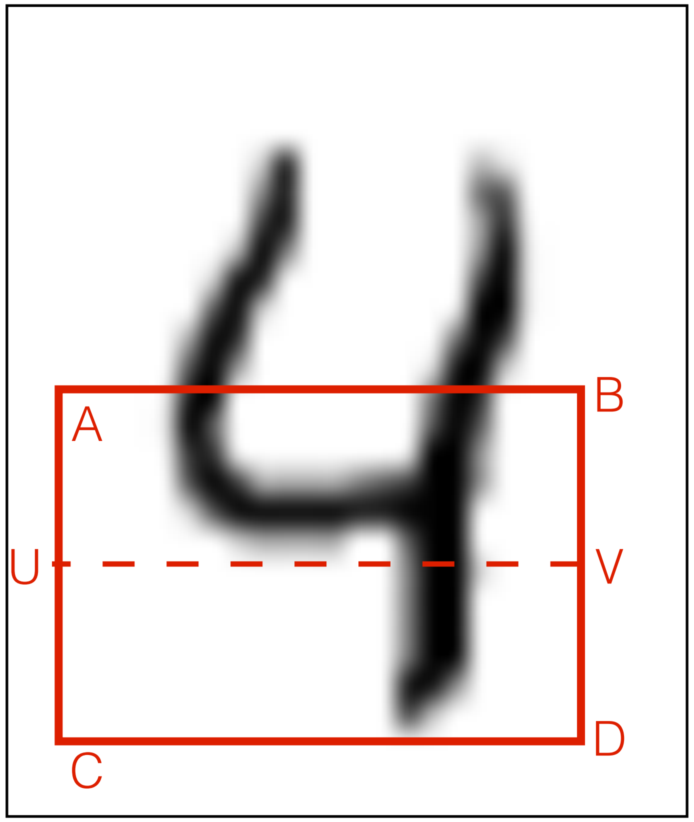
o



- for each rectangle r=ABCD on image X we can compute a “black value”
  - $\texttt{black}_r\texttt{(X)}$ = number of black pixels in the mask cut by r in image X
- we can compute $\texttt{black}_r\texttt{(X)}$ efficiently, if we compute in the right order!
  - dynamic programing

Vertical, horizontal features for a rectangle





- horizontal feature

  $\Delta_{hr}(X) = black_{r\text{-}left}() - black_{r\text{-}right}(X)$

  $= black_{AMCN}(X) - black_{MBND}(X)$

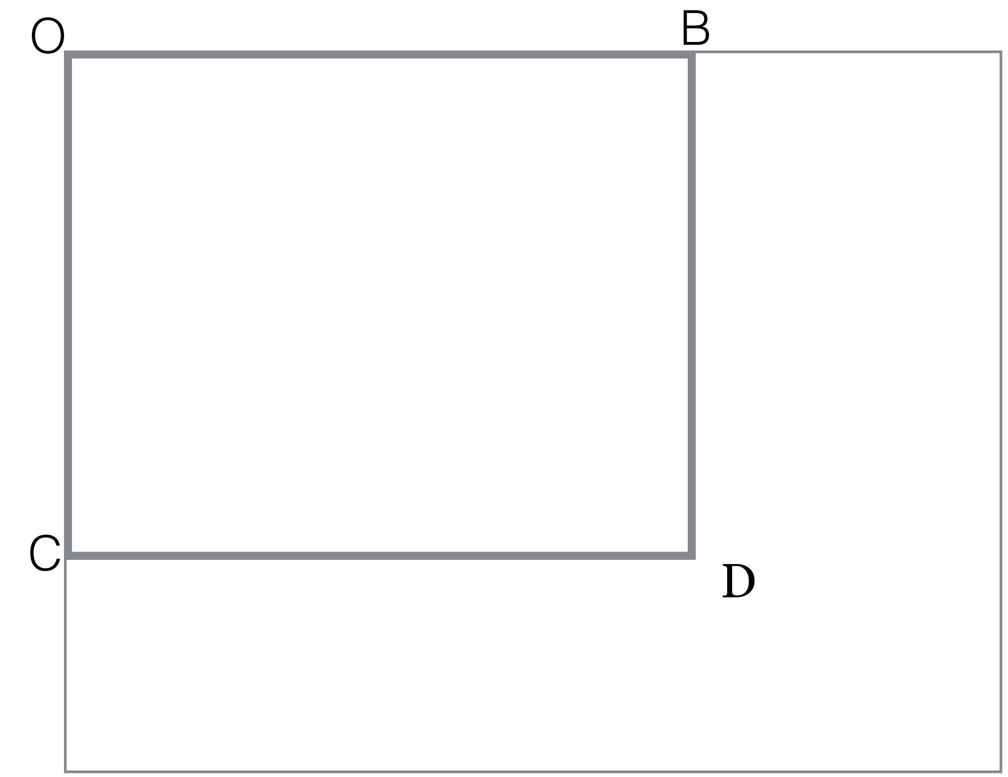
- vertical feature

  $\Delta_{hv}(X) = black_{r\text{-}top}() - black_{r\text{-}bottom}(X)$

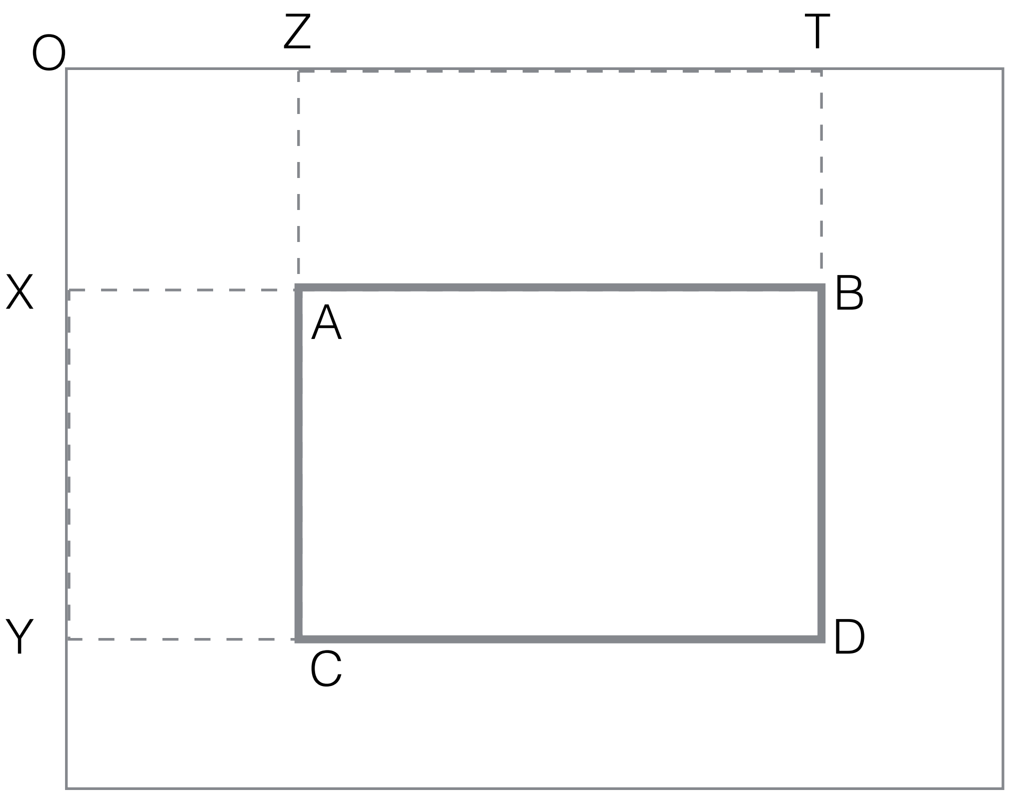
  $= black_{ABUV}(X) - black_{UVCD}(X)$

- $|S|$ rectangles, 2 features each $\Rightarrow 2|S|$ features extracted (from each image)
  - if we also store the $black_r(X)$ value, thats 3 features/rectangle ($black_r(X)$, $\Delta_{hr}(X)$, $\Delta_{hv}(X)$) for $3|S|$ features extracted.

How to compute $\texttt{black}_r\texttt{(X)}$ efficiently

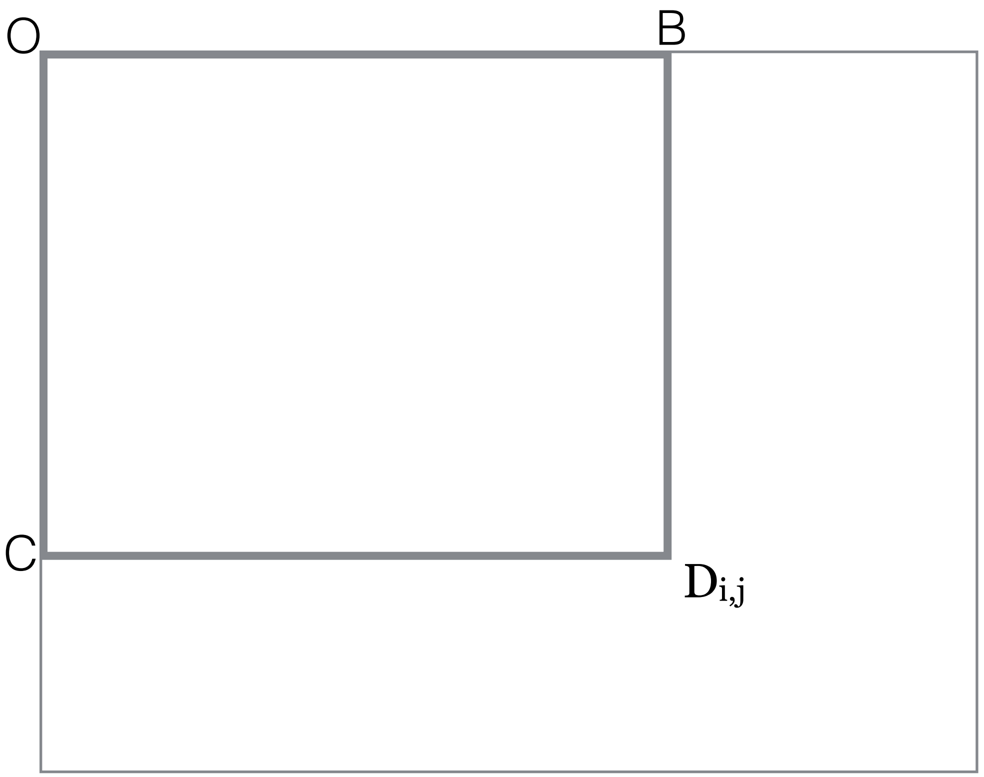


- first compute it for all rectangles cornered in O (A=O) fix image corner.
  - That is compute $\texttt{black}_r\texttt{(X)}$ for each pixel D



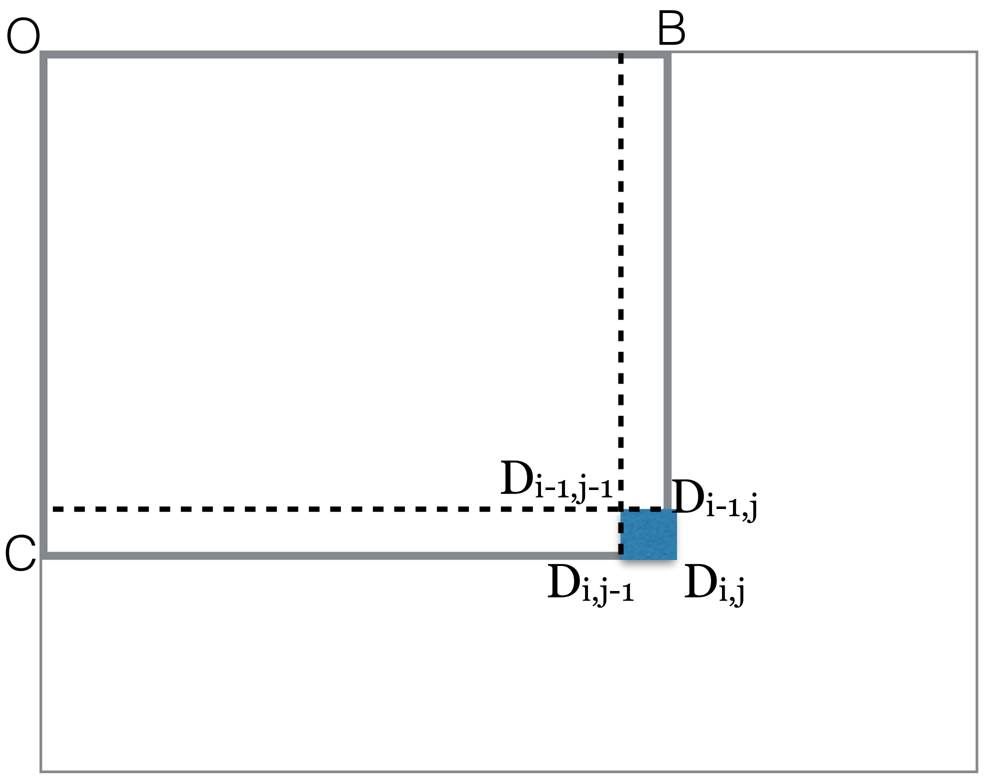
- then every rectangle r=ABCD can be computed in constant time from O-cornered rectangles
- `black(rectangle ABCD) = black(OTYD) - black(OTXB) - black(OZYC) + black(OZXA)`

O–corner rectangles computation



- r=OBCD determined by D
- naively one can compute all $black_r(X) = black_D(X)$ for all rectangles as
- `for i=1:n`
- `for j=1:n`
  - `D=Dij pixel`
  - `blackDij(X) = count of black pixels in OBCD`
- total $O(n^4)$ running time
  - n = size of the square image

O-corner rectangles : dynamic programing



- r=OBCD determined by D
- dynamic programing computes a rectangle from the rectangle computed already

- `for i=1:n`
- `for j=1:n`
  - `D=Dij pixel`

```
black_Dij(X)   =
    black_Di,j-1(X) +
    black_Di-1,j(X) -
    black_Di-1,j-1(X)+
    black(pixel_Dij ,X)
```

- total $O(n^2)$ running time
  - much better