Document and Content Analysis

Summer 2009

Lecture 7
Character Recognition

Thomas Breuel
Faisal Shafait

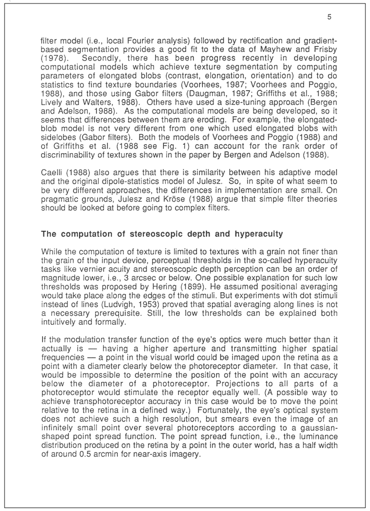
OCR steps

- cut apart pages into text lines and images
- cut apart each text line into characters
- recognize each character
- re-assemble the page text

page → text lines

(Talked about this last time.)

original page



filter model (i.e., local Fourier analysis) followed by rectification and gradient-based segmentation provides a good fit to the data of Mayhew and Frisby (1978). Secondly, there has been progress recently in developing computational models which achieve texture segmentation by computing parameters of elongated blobs (contrast, elongation, orientation) and to do statistics to find texture boundaries (Voorhees, 1987; Voorhees and Poggio, 1988), and those using Gabor filters (Daugman, 1987; Griffiths et al., 1988; Lively and Walters, 1988). Others have used a size-tuning approach (Bergen and Adelson, 1988). As the computational models are being developed, so it seems that differences between them are eroding. For example, the elongated-blob model is not very different from one which used elongated blobs with sidelobes (Gabor filters). Both the models of Voorhees and Poggio (1988) and of Griffiths et al. (1988 see Fig. 1) can account for the rank order of discriminability of textures shown in the paper by Bergen and Adelson (1988).

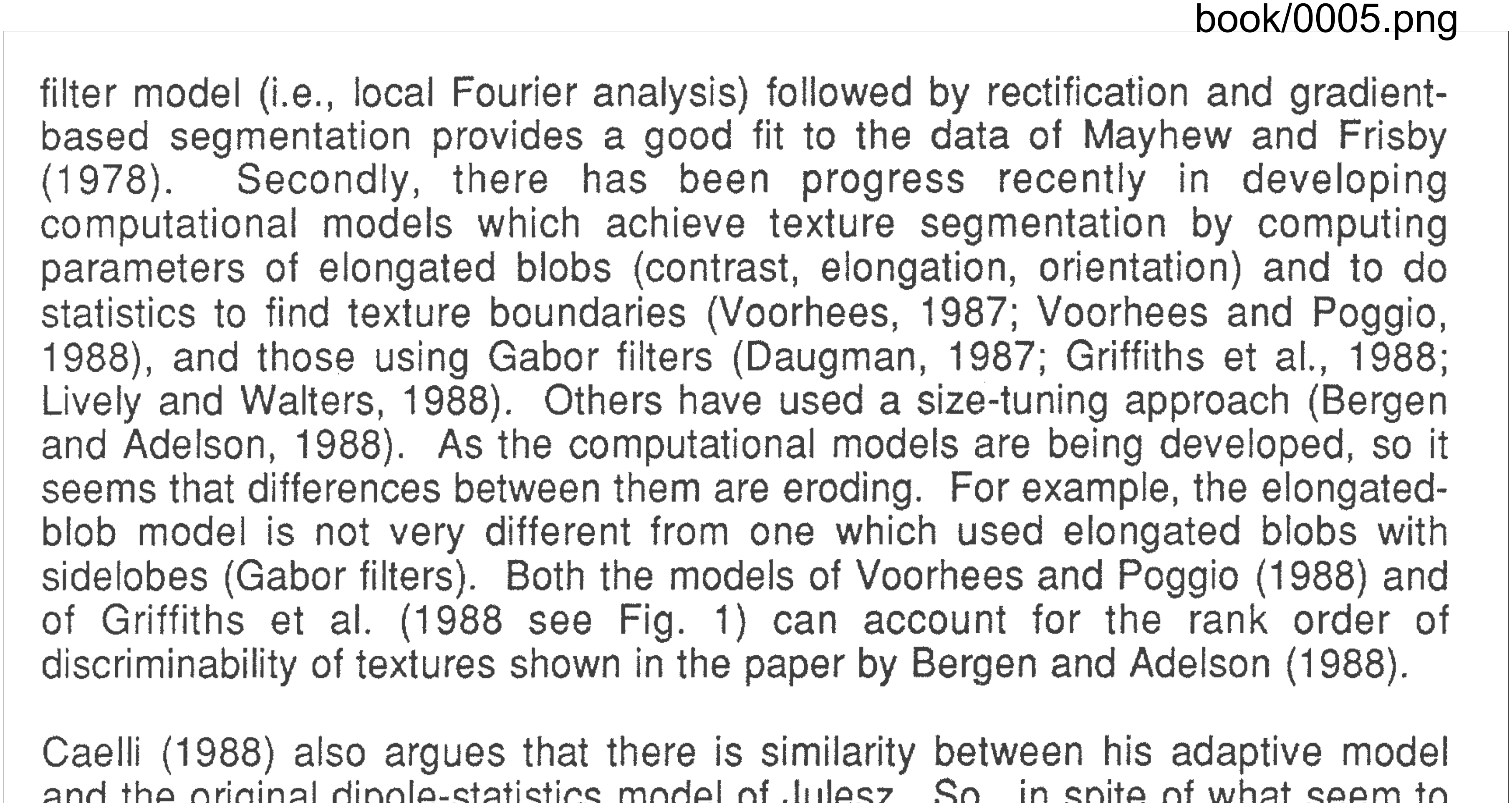
Caelli (1988) also argues that there is similarity between his adaptive model and the original dipole-statistics model of Julesz. So, in spite of what seem to be very different approaches, the differences in implementation are small. On pragmatic grounds, Julesz and Kröse (1988) argue that simple filter theories should be looked at before going to complex filters.

The computation of stereoscopic depth and hyperacuity

While the computation of texture is limited to textures with a grain not finer than the grain of the input device, perceptual thresholds in the so-called hyperacuity tasks like vernier acuity and stereoscopic depth perception can be an order of magnitude lower, i.e., 3 arcsec or below. One possible explanation for such low thresholds was proposed by Hering (1899). He assumed positional averaging would take place along the edges of the stimuli. But experiments with dot stimuli instead of lines (Ludvigh, 1953) proved that spatial averaging along lines is not a necessary prerequisite. Still, the low thresholds can be explained both intuitively and formally.

If the modulation transfer function of the eye's optics were much better than it actually is — having a higher aperture and transmitting higher spatial frequencies — a point in the visual world could be imaged upon the retina as a point with a diameter clearly below the photoreceptor diameter. In that case, it would be impossible to determine the position of the point with an accuracy below the diameter of a photoreceptor. Projections to all parts of a photoreceptor would stimulate the receptor equally well. (A possible way to achieve transphotoreceptor accuracy in this case would be to move the point relative to the retina in a defined way.) Fortunately, the eye's optical system does not achieve such a high resolution, but smears even the image of an infinitely small point over several photoreceptors according to a gaussian-shaped point spread function. The point spread function, i.e., the luminance distribution produced on the retina by a point in the outer world, has a half width of around 0.5 arcmin for near-axis imagery.

book/0005.png

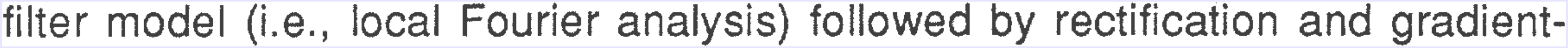


filter model (i.e., local Fourier analysis) followed by rectification and gradient-based segmentation provides a good fit to the data of Mayhew and Frisby (1978). Secondly, there has been progress recently in developing computational models which achieve texture segmentation by computing parameters of elongated blobs (contrast, elongation, orientation) and to do statistics to find texture boundaries (Voorhees, 1987; Voorhees and Poggio, 1988), and those using Gabor filters (Daugman, 1987; Griffiths et al., 1988; Lively and Walters, 1988). Others have used a size-tuning approach (Bergen and Adelson, 1988). As the computational models are being developed, so it seems that differences between them are eroding. For example, the elongated-blob model is not very different from one which used elongated blobs with sidelobes (Gabor filters). Both the models of Voorhees and Poggio (1988) and of Griffiths et al. (1988 see Fig. 1) can account for the rank order of discriminability of textures shown in the paper by Bergen and Adelson (1988).

Caelli (1988) also argues that there is similarity between his adaptive model

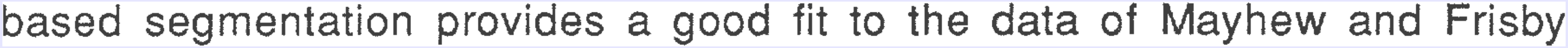
text lines

0005/0001.png



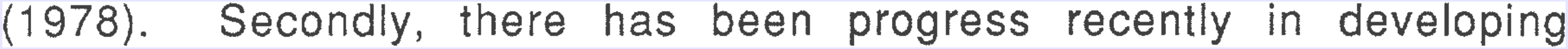
filter model (i.e., local Fourier analysis) followed by rectification and gradient-

0005/0002.png



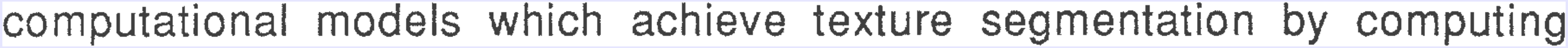
based segmentation provides a good fit to the data of Mayhew and Frisby

0005/0003.png



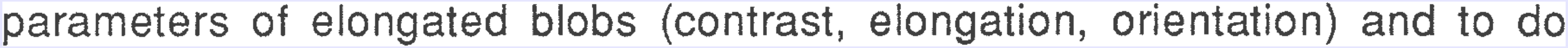
(1978). Secondly, there has been progress recently in developing

0005/0004.png



computational models which achieve texture segmentation by computing

0005/0005.png



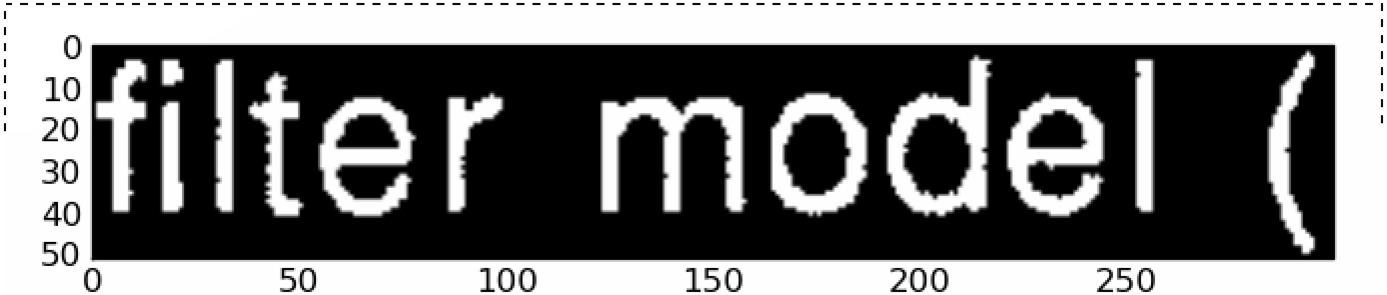
parameters of elongated blobs (contrast, elongation, orientation) and to do

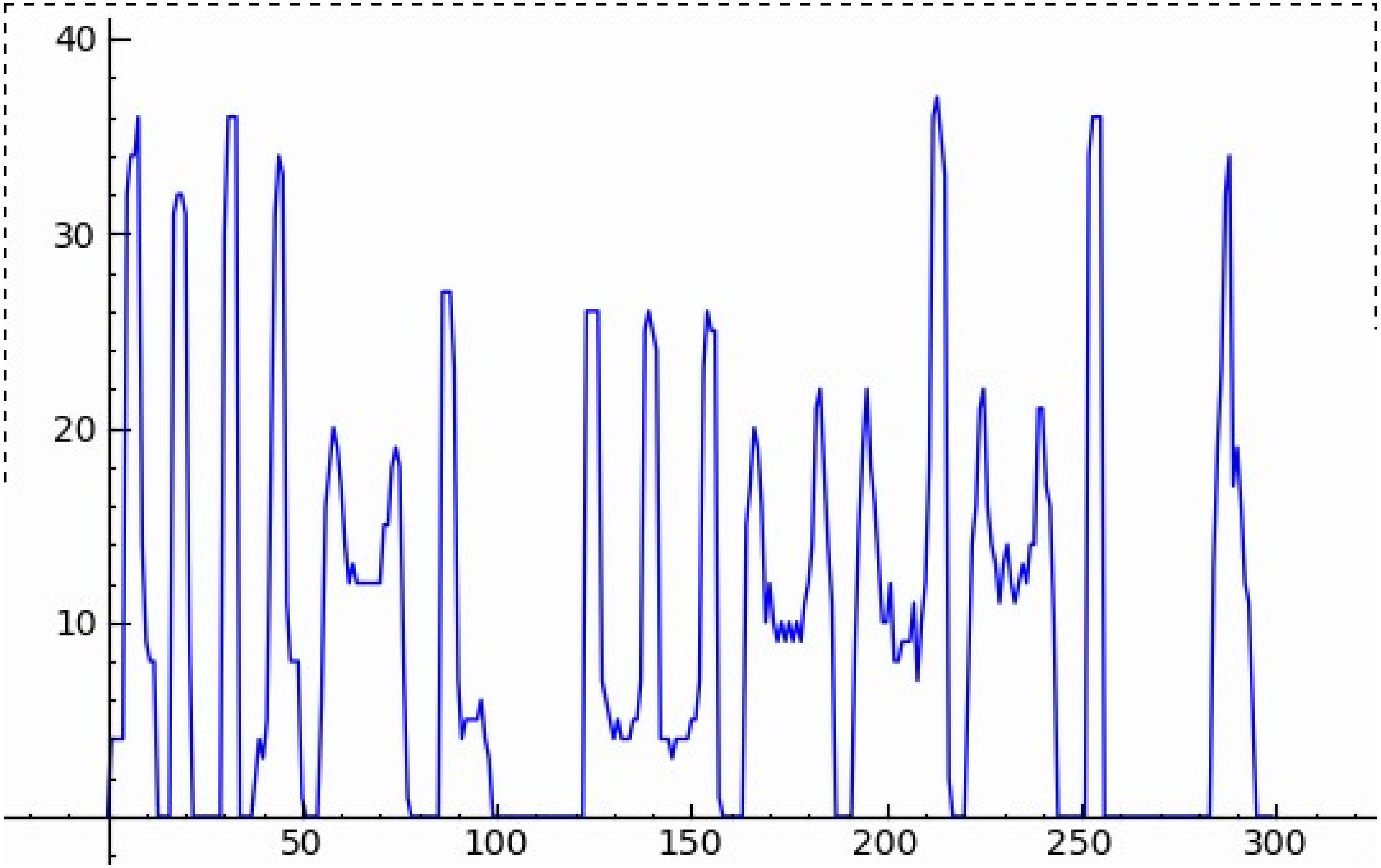
0005/0006.png ...

text line → characters

How do you cut apart a line of text into characters?

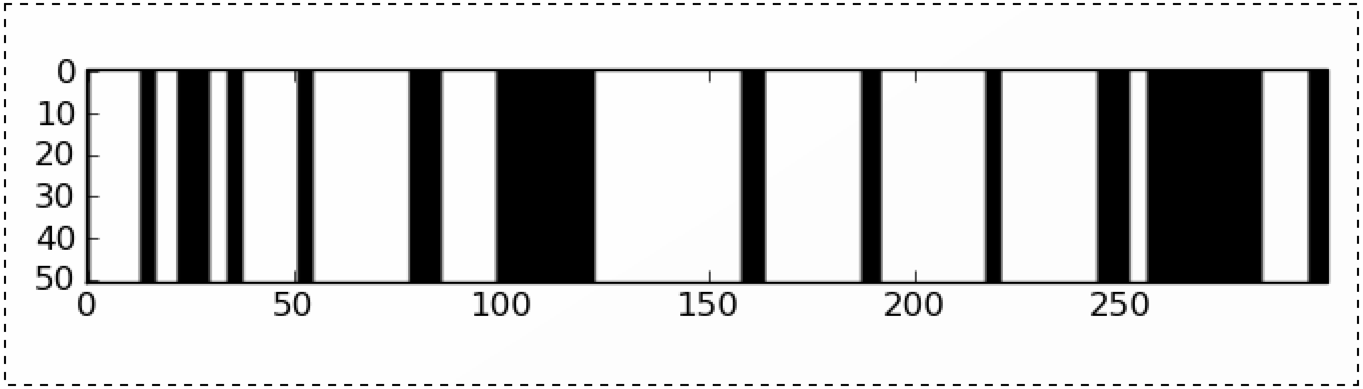
segmentation by projection



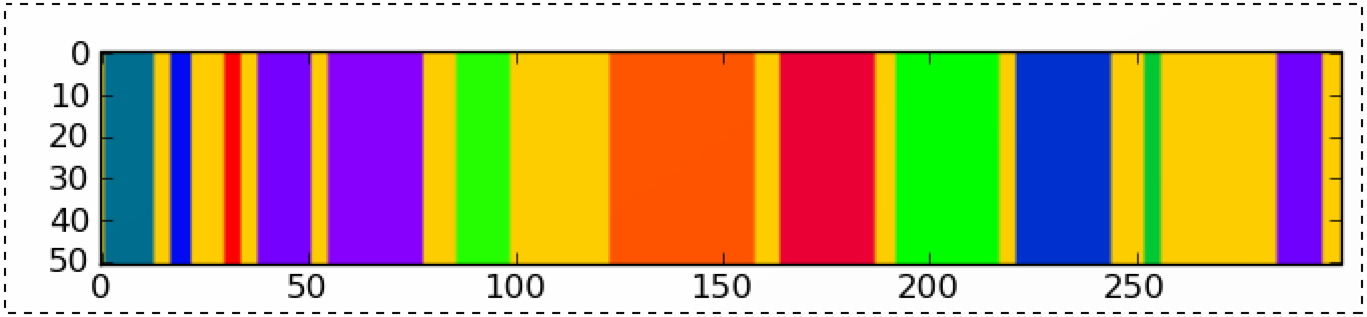


segmentation by projection

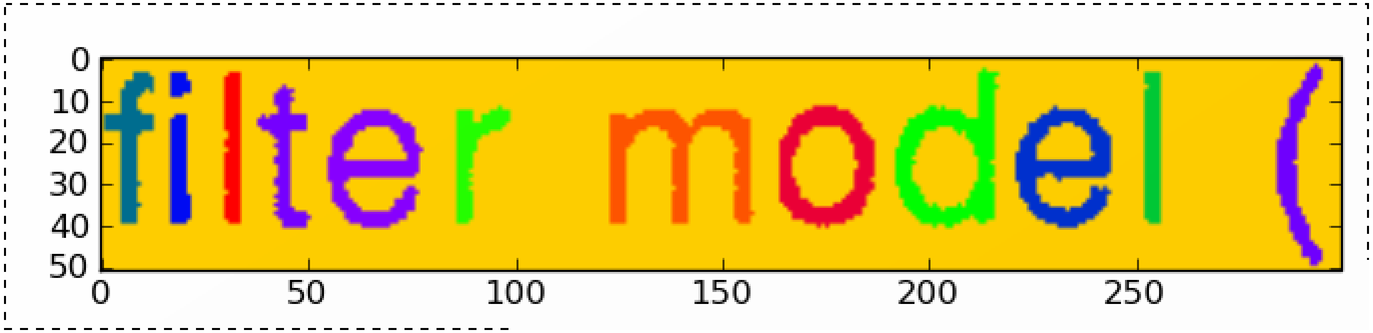
compute segmentation components



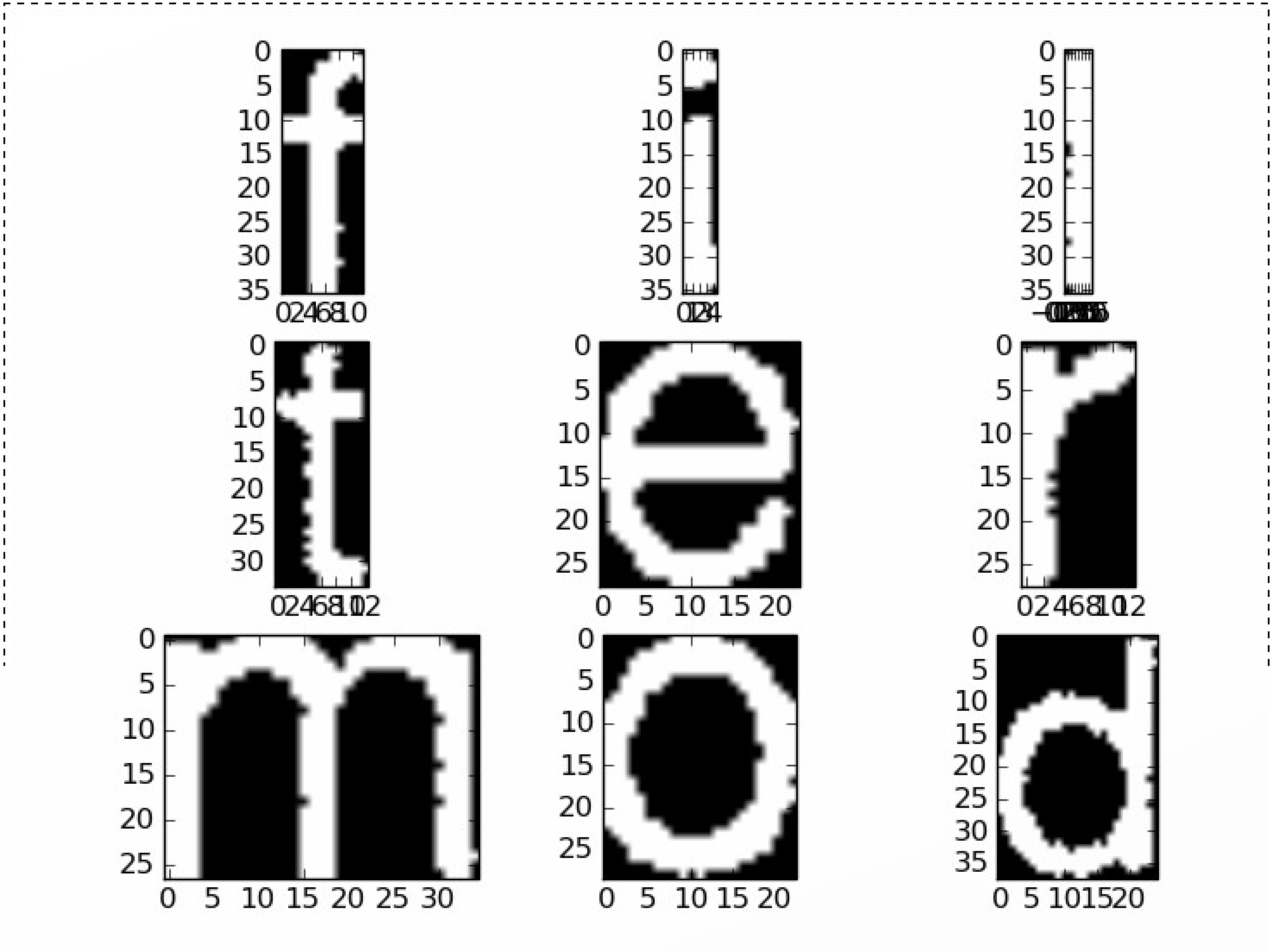
label connected components



“and” with original binary image



segmentation by projection

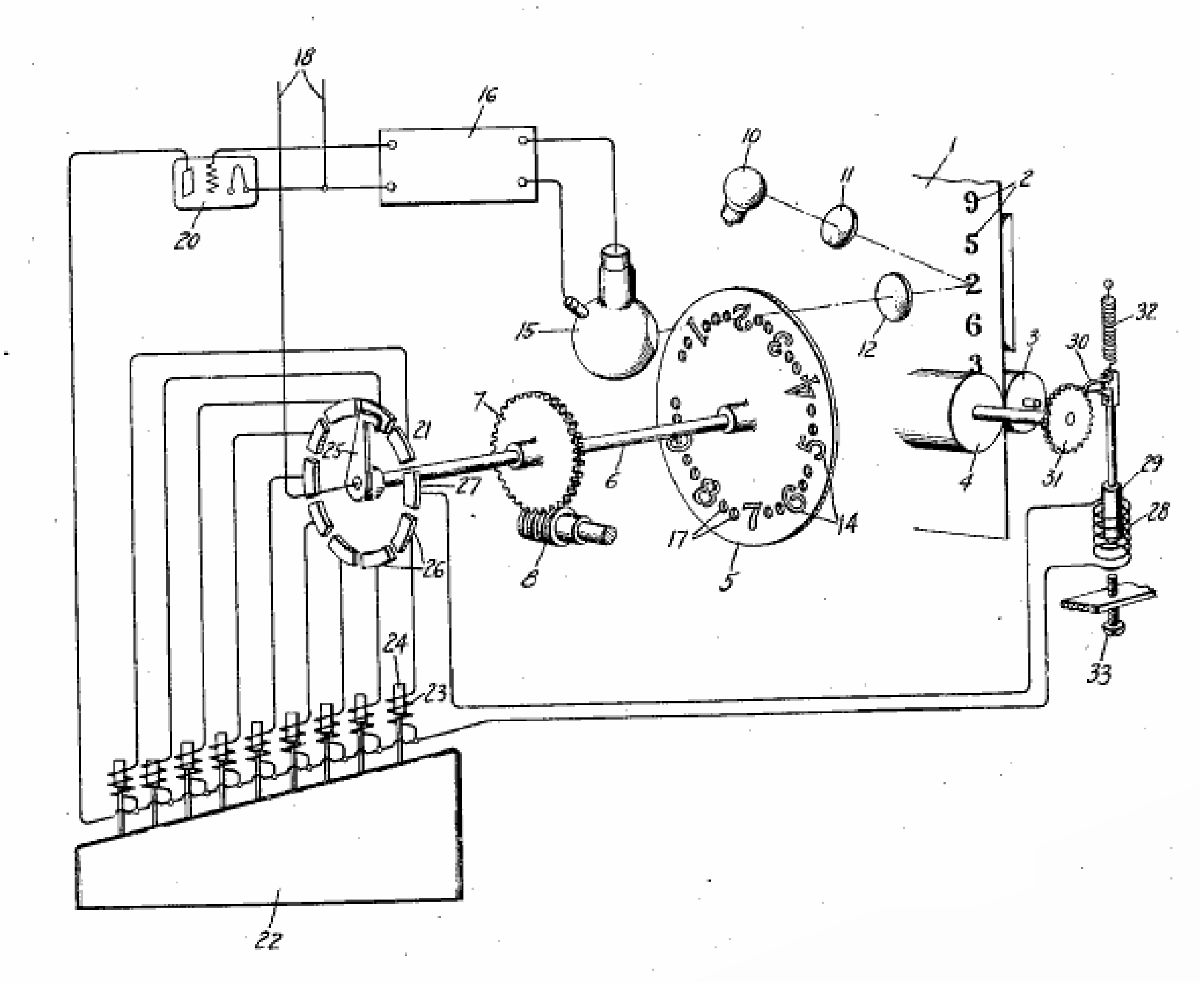


segmentation by projection

- **input**
  - binary text line image
- **algorithm**
  - project along the vertical axis
  - find low/zero regions in projection profile
  - generate an image where those “bands” are set to 0, all other values are 1
  - label connected components
  - intersect the connected components with the original binary image
- **output**
  - color-coded segmentation image

recognition by template matching

recognition by template matching



Handel, 1933

recognition by template matching

- **idea**
  - put different templates on top of each character
  - see which template matches best
  - return the identity associated with the template

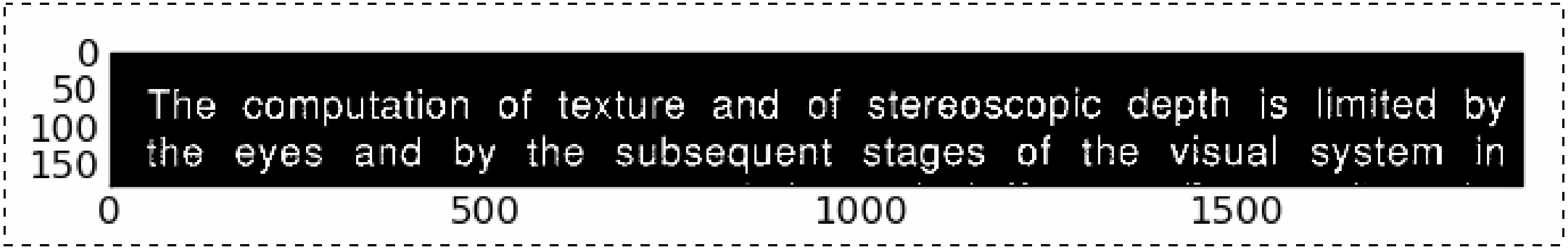
correlation

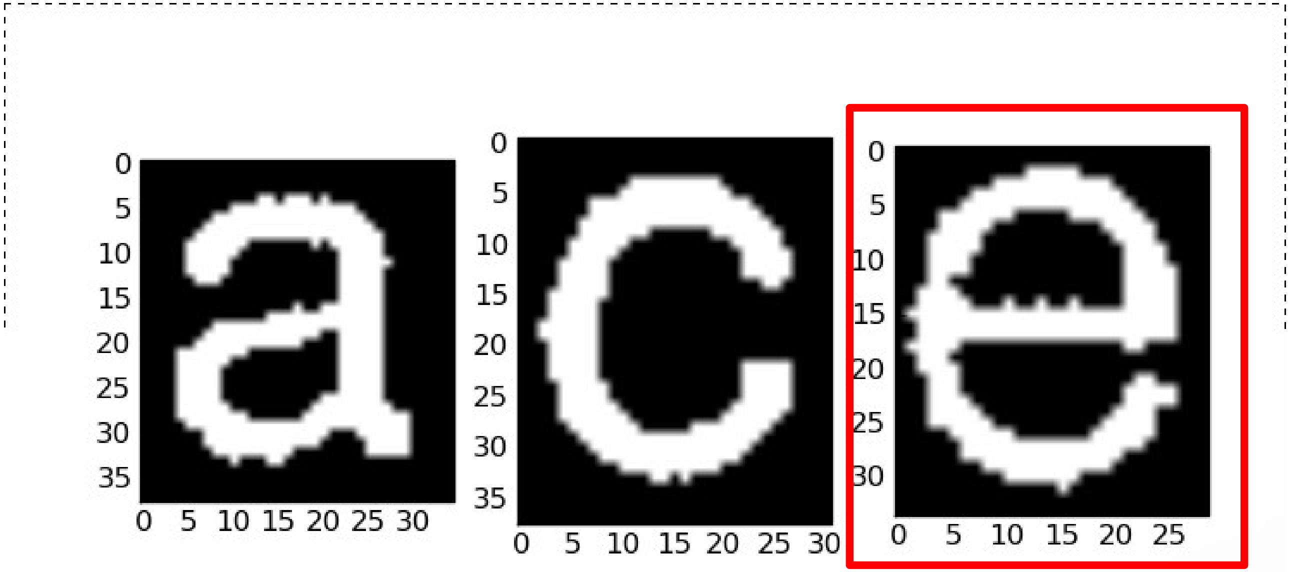
$$r_{i,j} = \sum_{s,t} s_{i+s,j+t} m_{s,t}$$

(Simple) correlation is like convolution, except that the offset is added, rather than subtracted.

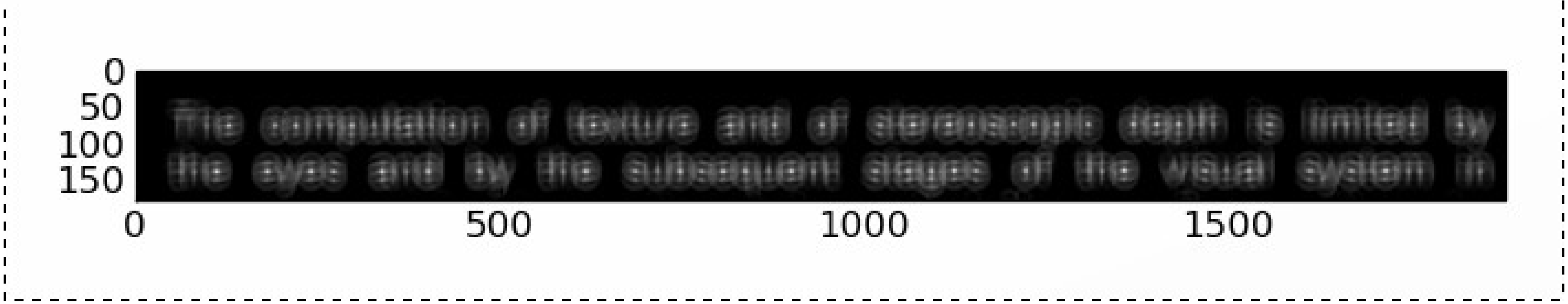
(Normalized or statistical cross-correlation is something different.)

template matching

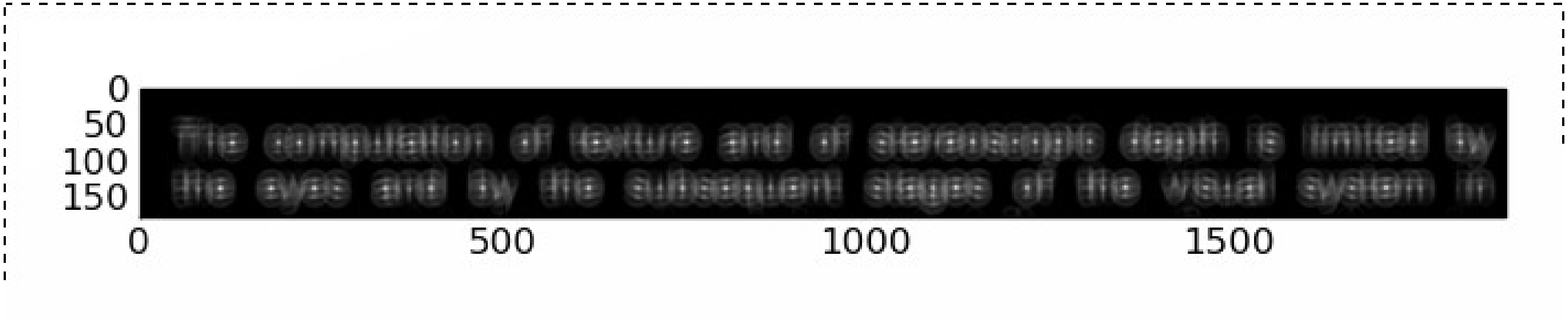

0
50
100
150
The computation of texture and of stereoscopic depth is limited by
the eyes and by the subsequent stages of the visual system in
0
500
1000
1500



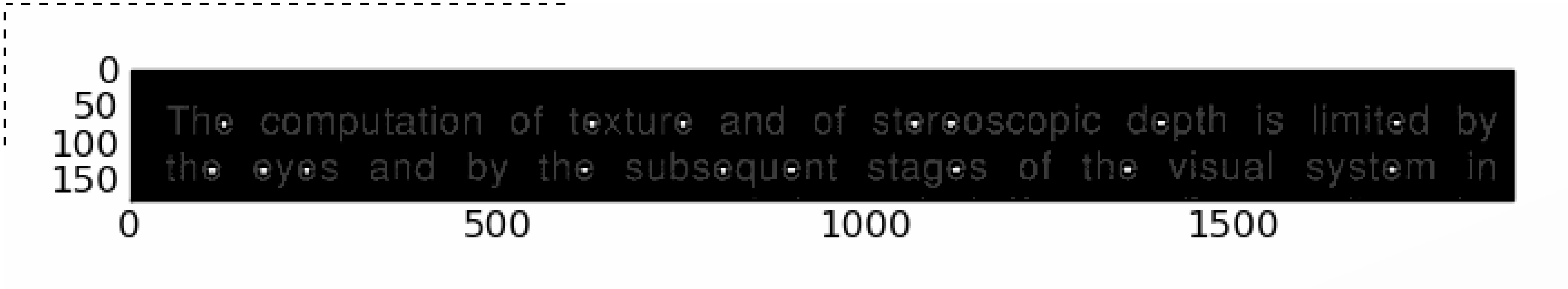
```
scipy.ndimage.filters.correlate(text,template_e)
```



detection threshold



```
markers = 1.0*(correlation>0.9*amax(correlation))
```

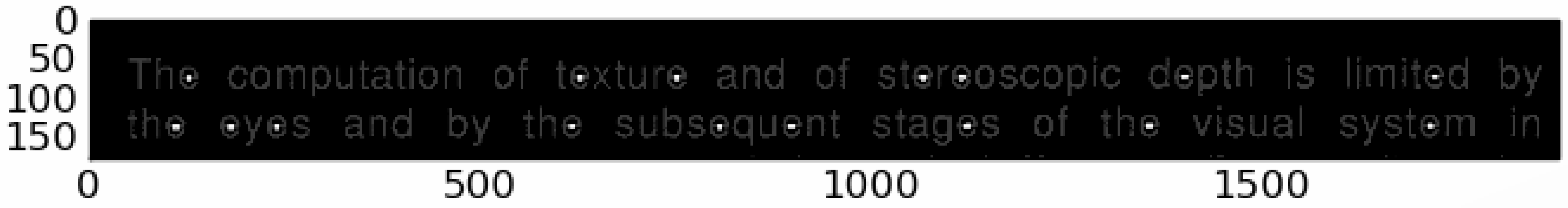


template matching

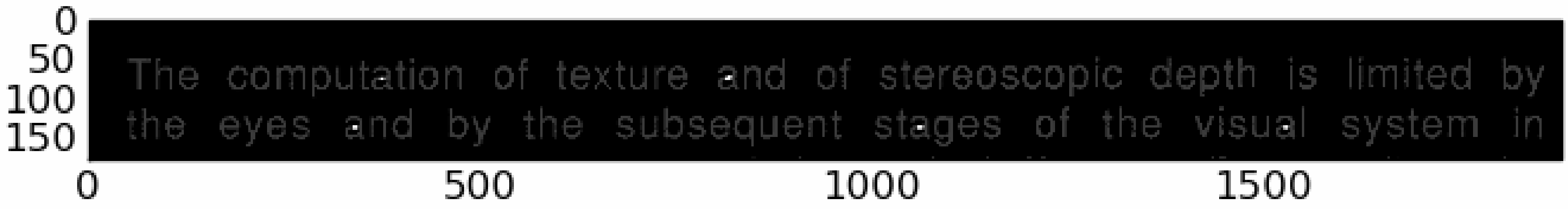
- **above example**
  - pick a template and slide it across entire image
  - inefficient: many places matched that needn't be
- **real-world recognition**
  - partition the image into character subimages
  - take each character
  - slide the template around each character a little bit
  - compute the maximum correlation as quality-of-match
  - return the best matching template

template matching

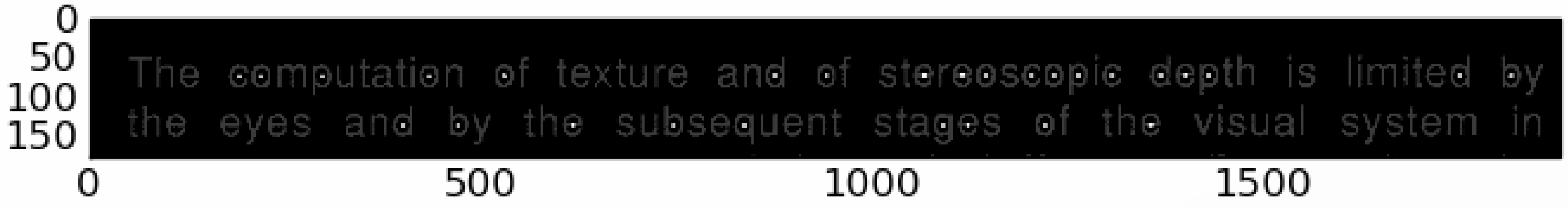
template_e



template_a

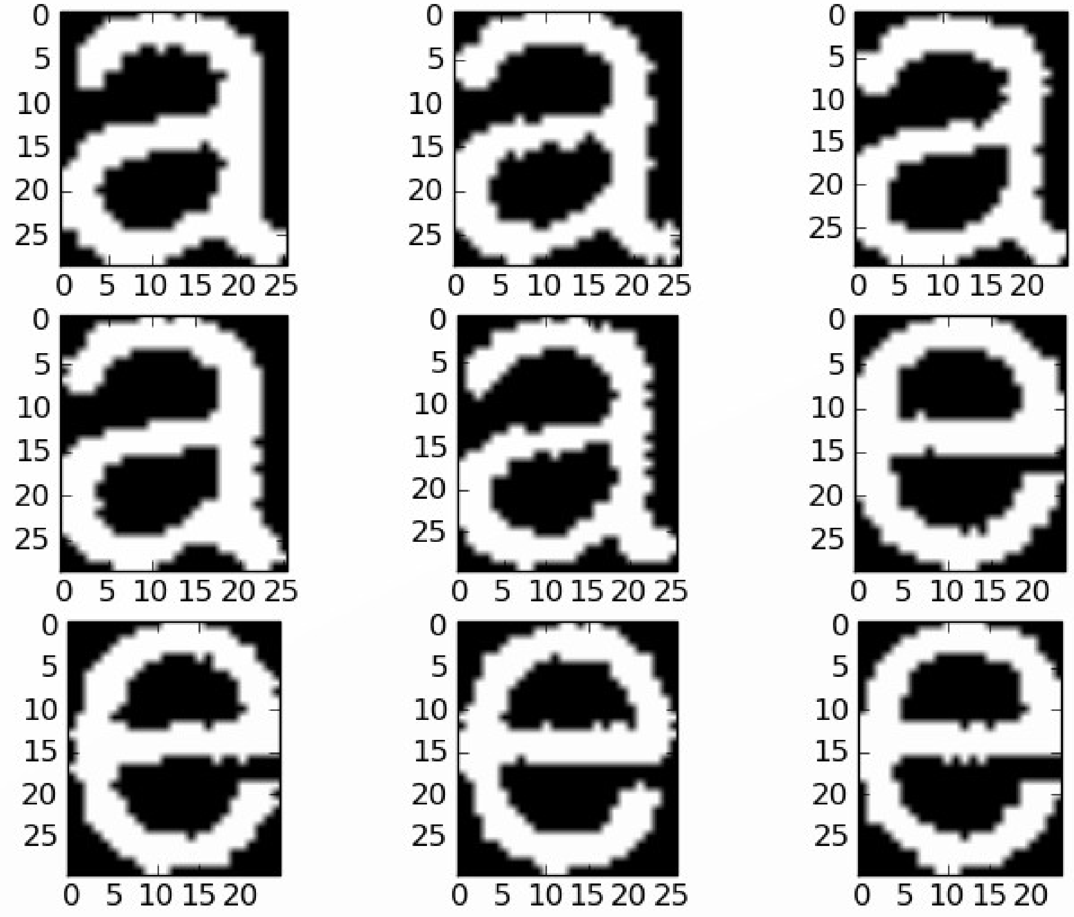


template_c

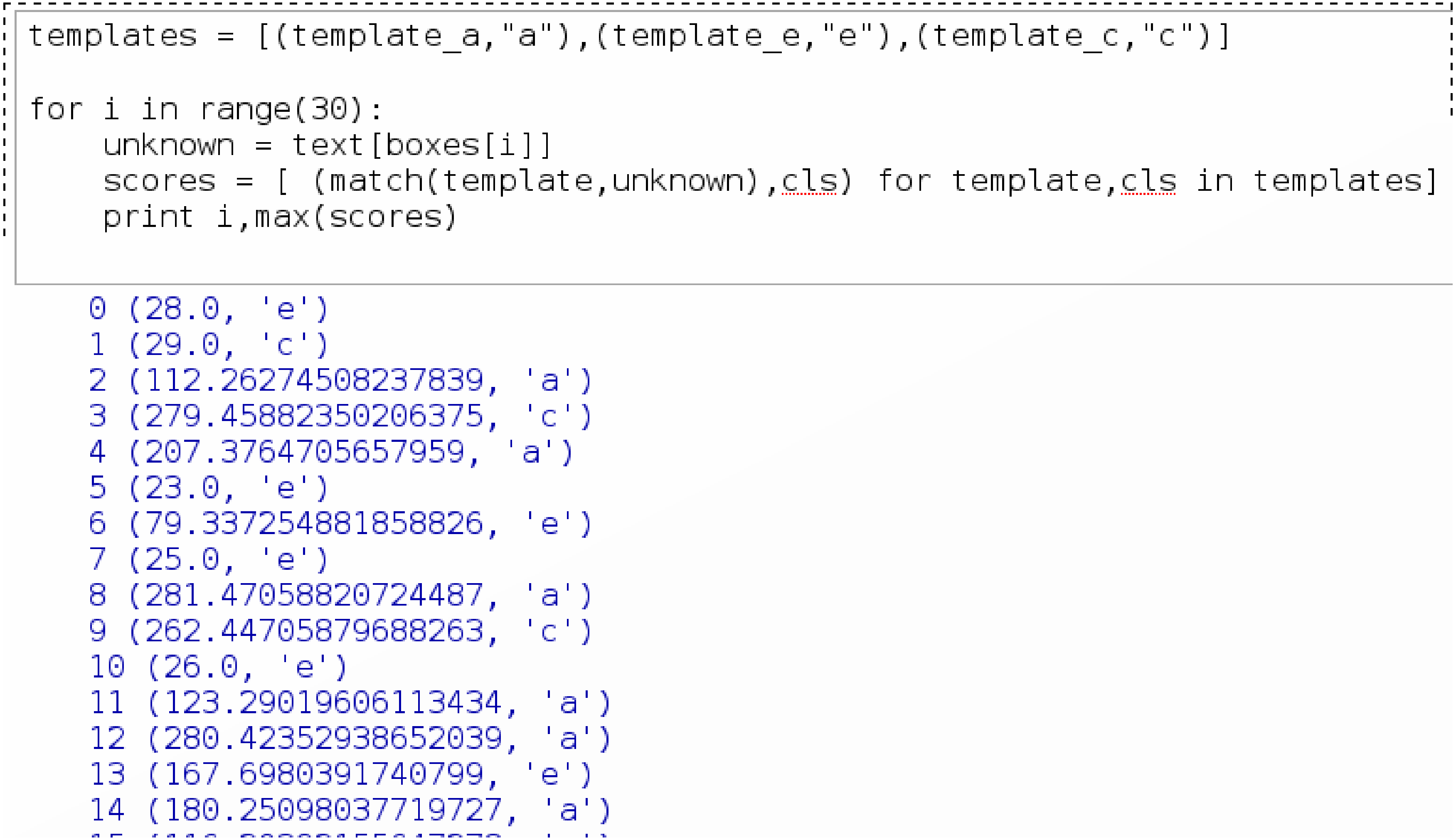


recognition by template matching

```
correlations = [ (match(text[boxes[i]],template_a),i) for i in range(len(boxes)) ]
```



template matching



```
templates = [(template_a,"a"),(template_e,"e"),(template_c,"c")]

for i in range(30):
    unknown = text[boxes[i]]
    scores = [ (match(template,unknown),cls) for template,cls in templates]
    print i,max(scores)
```

```
    0 (28.0, 'e')
    1 (29.0, 'c')
    2 (112.26274508237839, 'a')
    3 (279.45882350206375, 'c')
    4 (207.3764705657959, 'a')
    5 (23.0, 'e')
    6 (79.337254881858826, 'e')
    7 (25.0, 'e')
    8 (281.47058820724487, 'a')
    9 (262.44705879688263, 'c')
    10 (26.0, 'e')
    11 (123.29019606113434, 'a')
    12 (280.42352938652039, 'a')
    13 (167.6980391740799, 'e')
    14 (180.25098037719727, 'a')
```

text line recognition

- **text line segmentation**
  - compute vertical projection profile
  - threshold
  - segment into characters
- **recognition**
  - extract each character
  - match against a list of templates
  - return the character code of the best-matching template

Are we done?

problems

- **segmentation by projection**
  - italics, ligatures, kerning
  - touching / broken characters
- **recognition by template matching**
  - slow
  - bad recognition accuracy
    - size, shape variation, noise, italics
    - new fonts etc.
    - thresholds?

better segmentation

typographic phenomena

To

kerning

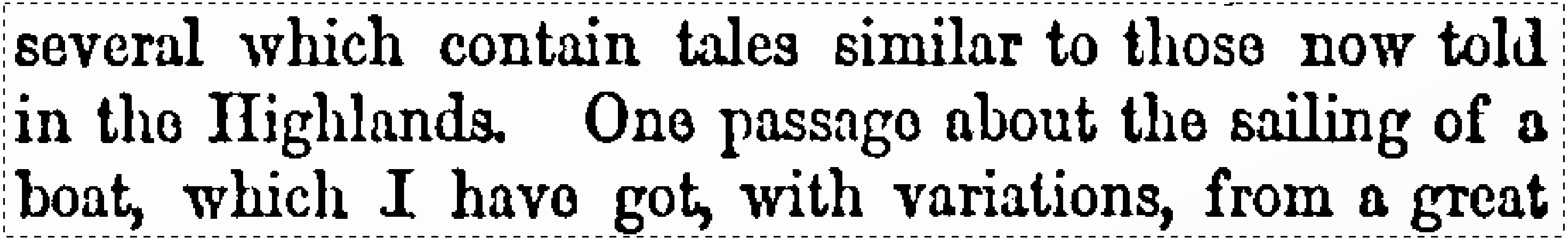
affine

touching characters &
ligatures

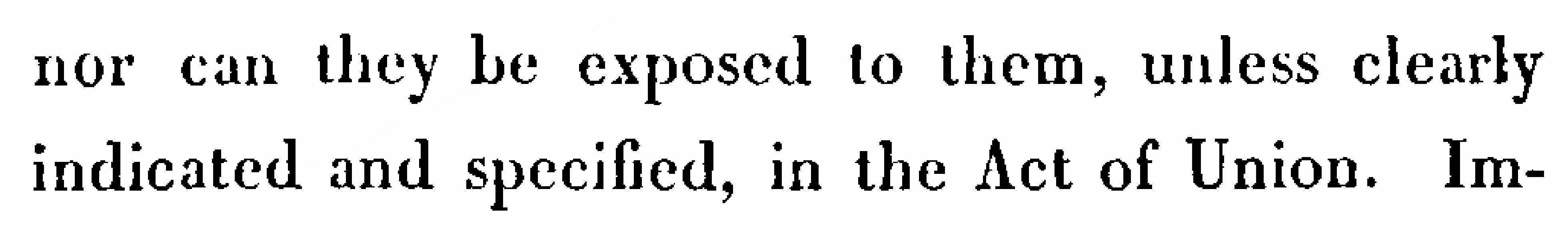
light

italics

image degradation



several which contain tales similar to those now told in the Highlands. One passage about the sailing of a boat, which I have got, with variations, from a great



nor can they be exposed to them, unless clearly indicated and specified, in the Act of Union. Im-

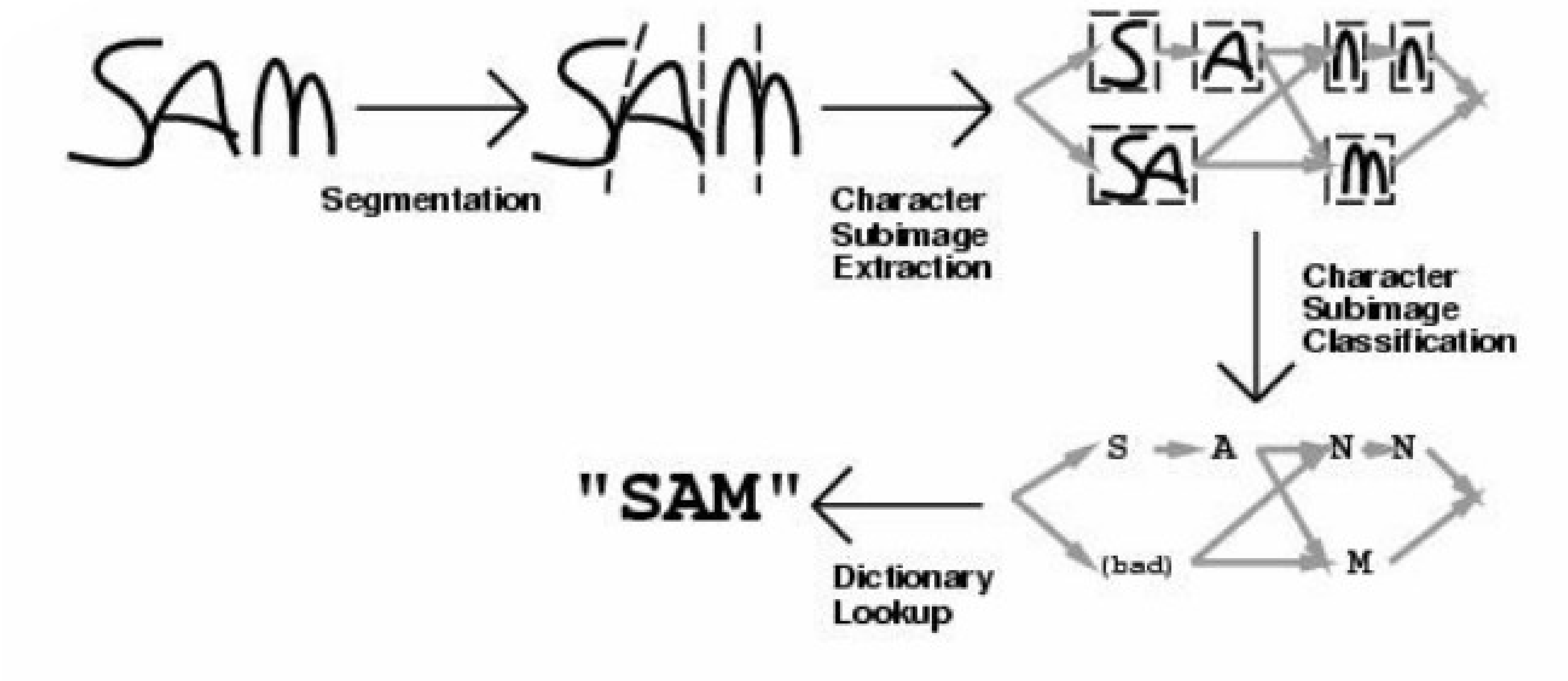
“simple approach”

- **assume that segmentation mostly works**
- **try to recognize things**
- **check the resulting words against dictionary**
- **if it doesn't fit...**
    - backtrack...
    - try to identify characters that can be split/merged...
    - try to recognize again
- **properties**
    - fast, intuitive... but requires a lot of tuning
    - doesn't generalize well to severely degraded

better text line segmentation

- **better cutting algorithm**
  - allow curved cuts
  - dynamic programming algorithm finds optimal paths
- **two step**
  - first, cut characters apart
  - second, group character parts together

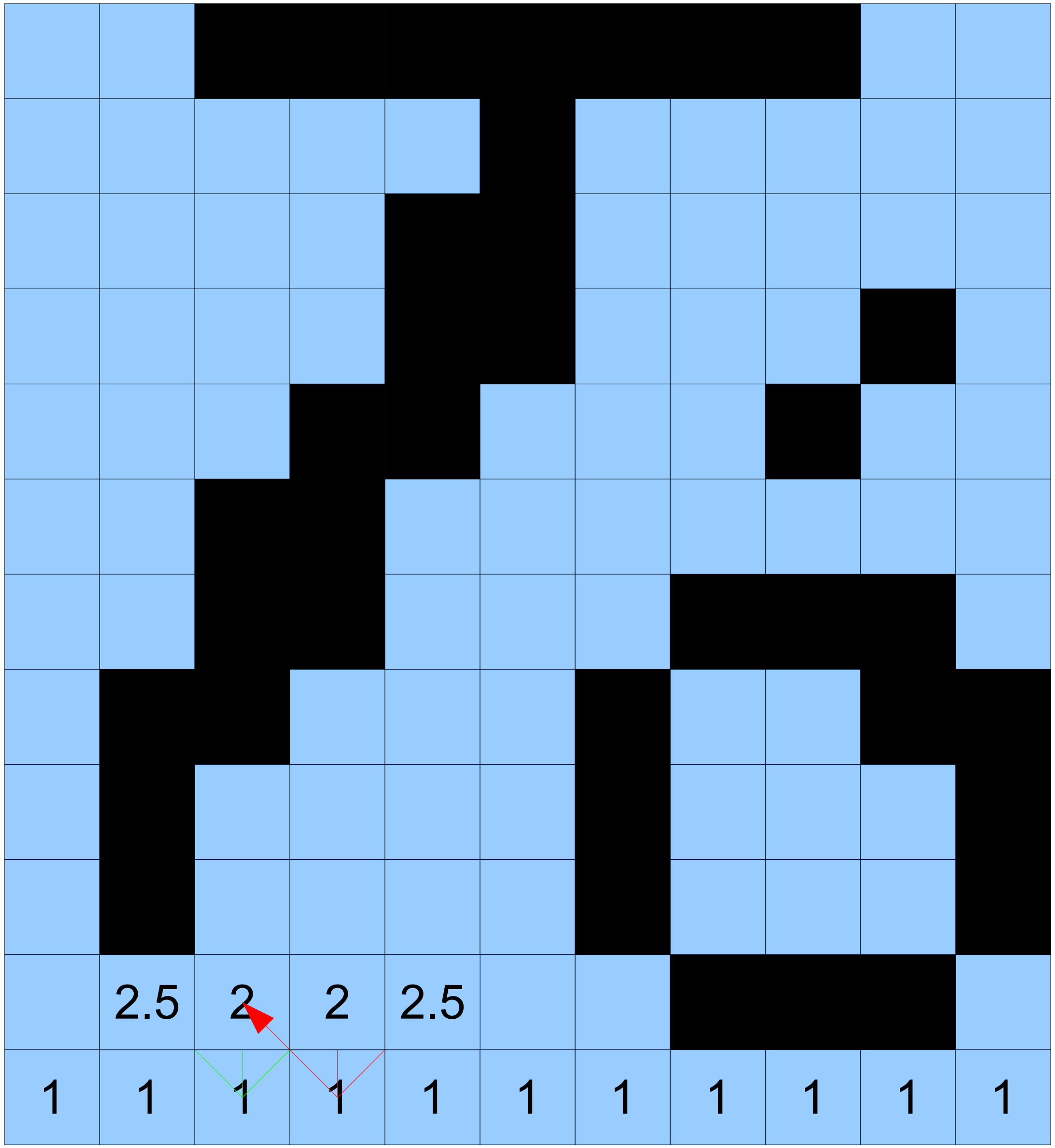
oversegmentation



SAM
Segmentation
Character Subimage Extraction
Character Subimage Classification
S
A
N
N
(bad)
M
Dictionary Lookup
"SAM"

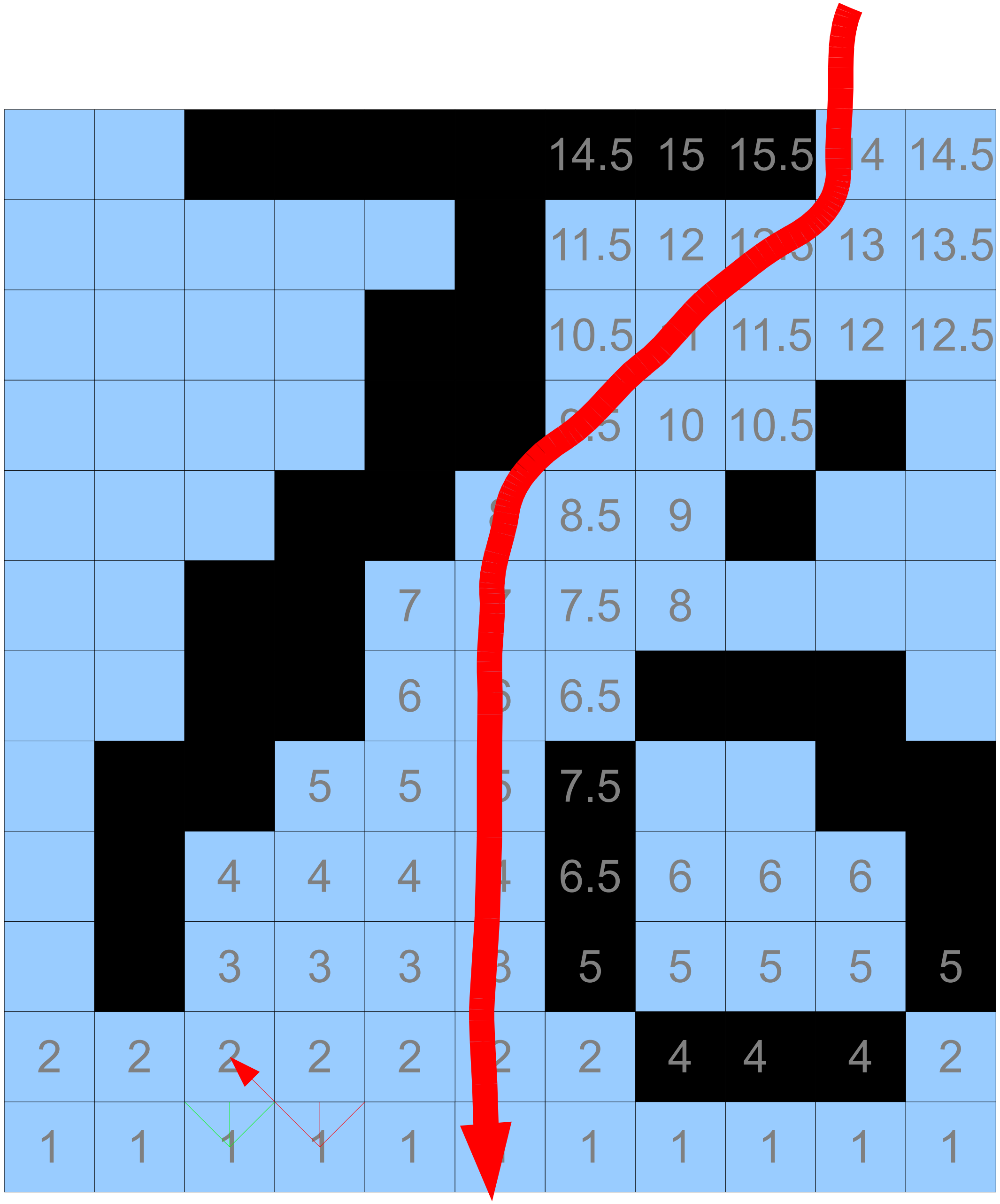
cuts by dynamic programming

- **step = 1**
- **diagonal = +0.5**
- **black = +2**

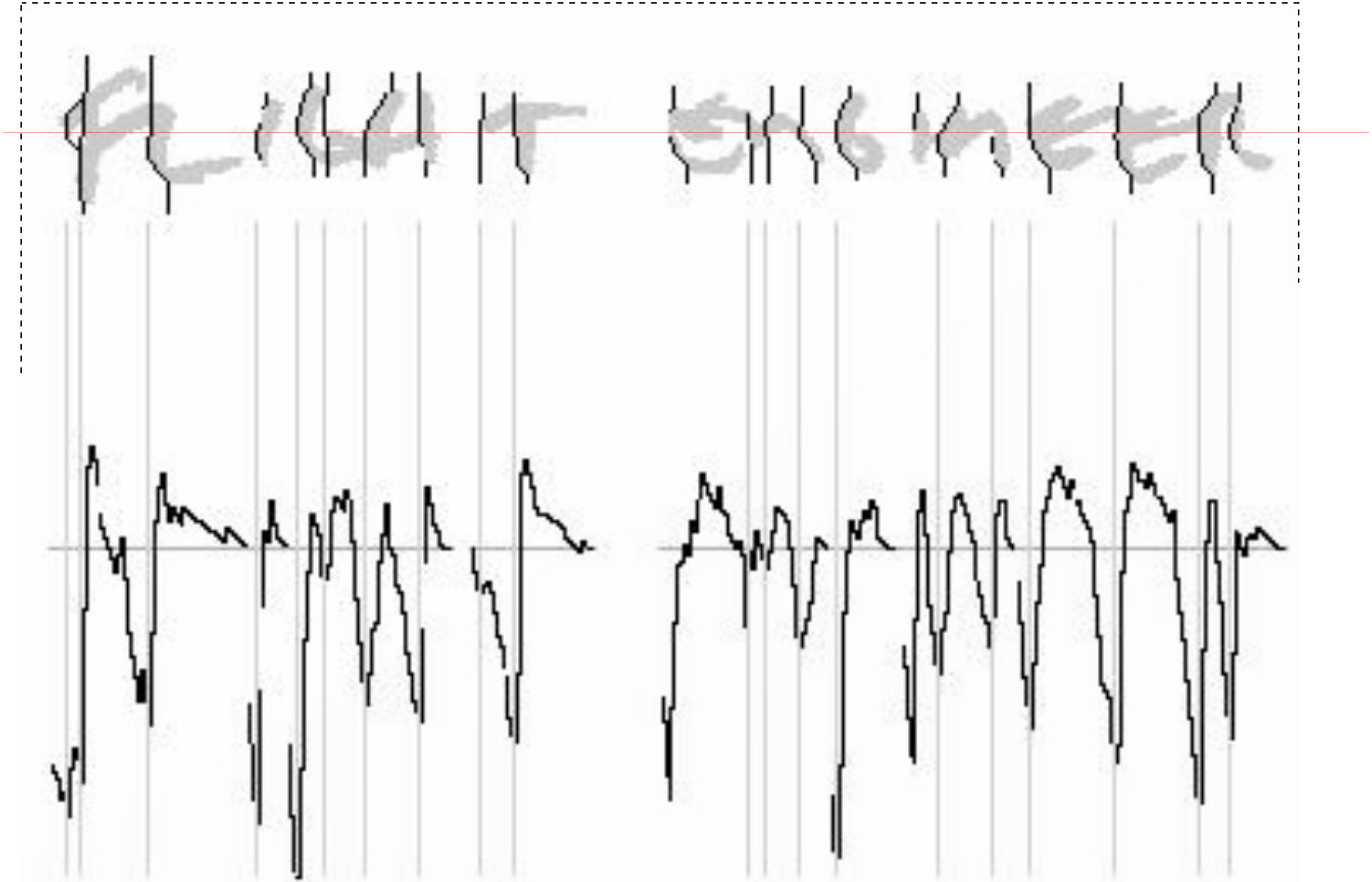


cuts by dynamic programming

- **step = 1**
- **diagonal = +0.5**
- **black = +2**



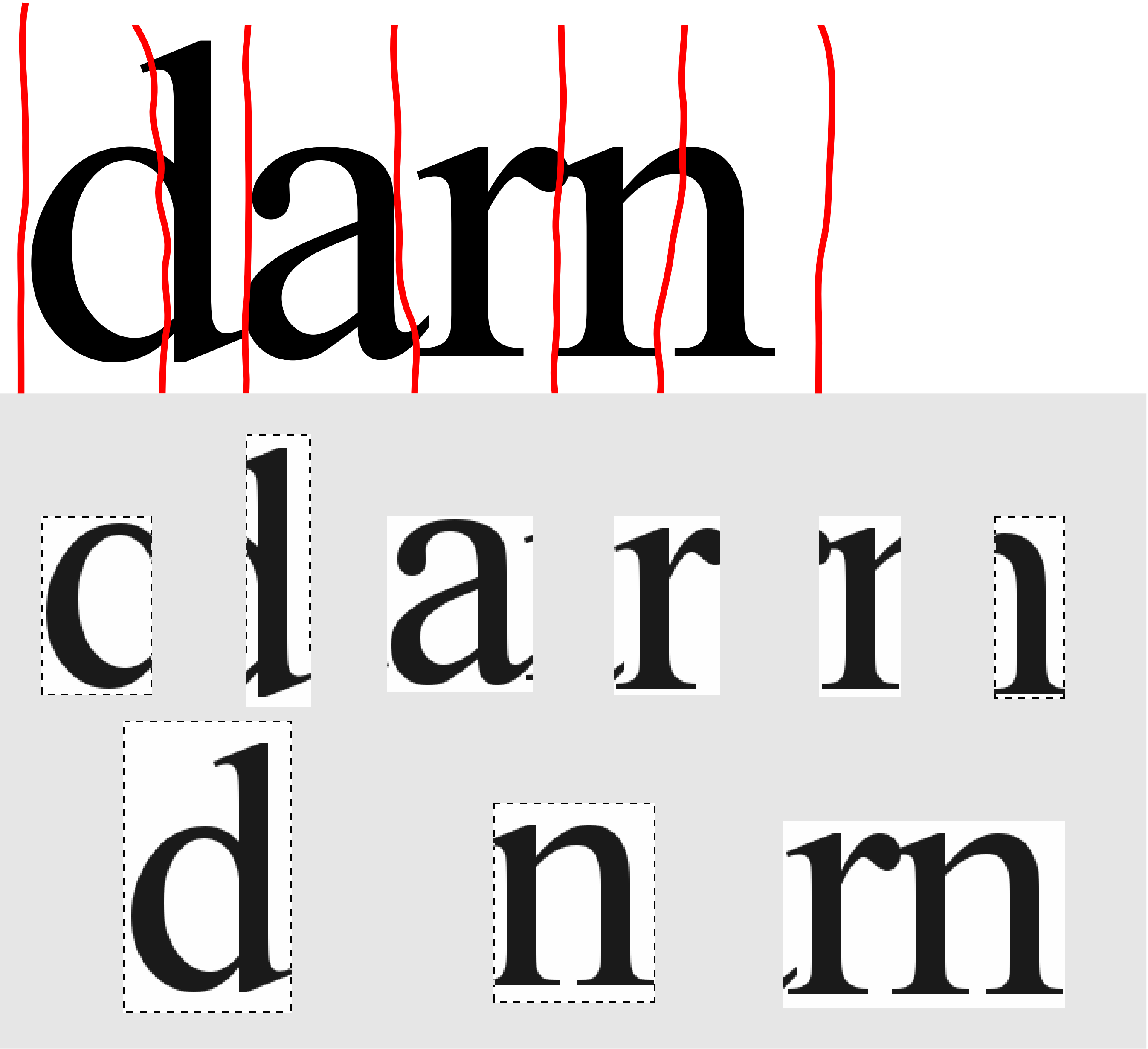
cuts by dynamic programming



cuts by dynamic programming

- **compute center line of text line**
- **propagate costs inward**
- **trace paths outward from center line**
  - this guarantees a path at every point
- **plot costs vs x-position**
- **pick those cut paths that have a local min.**

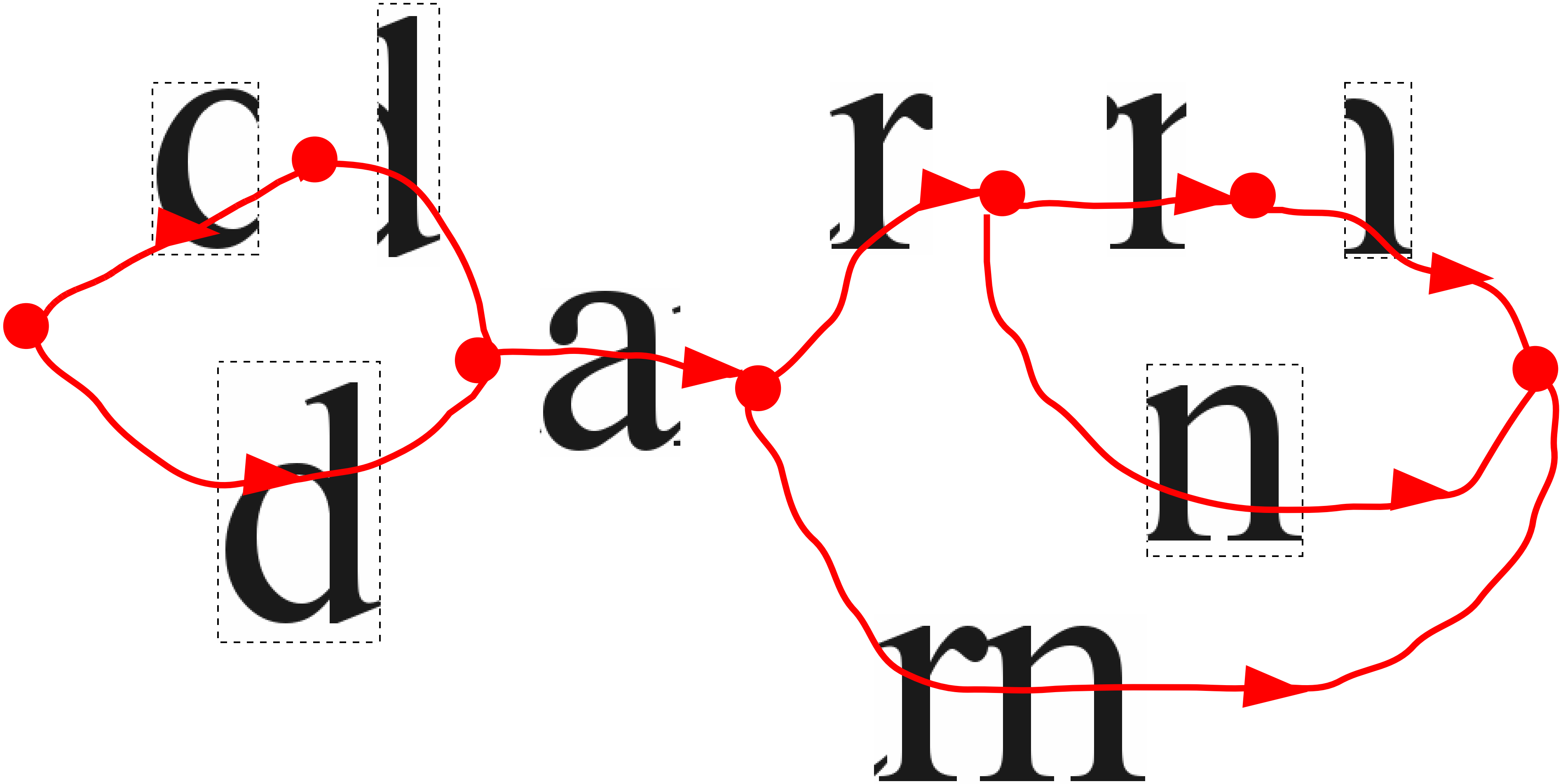
cuts vs characters



segmentation graph

- **consider each cut a node in a graph**
- **for every pair of cuts no more than n apart**
    - consider the edge between the two nodes...
    - associate it with the image delimited by the cuts

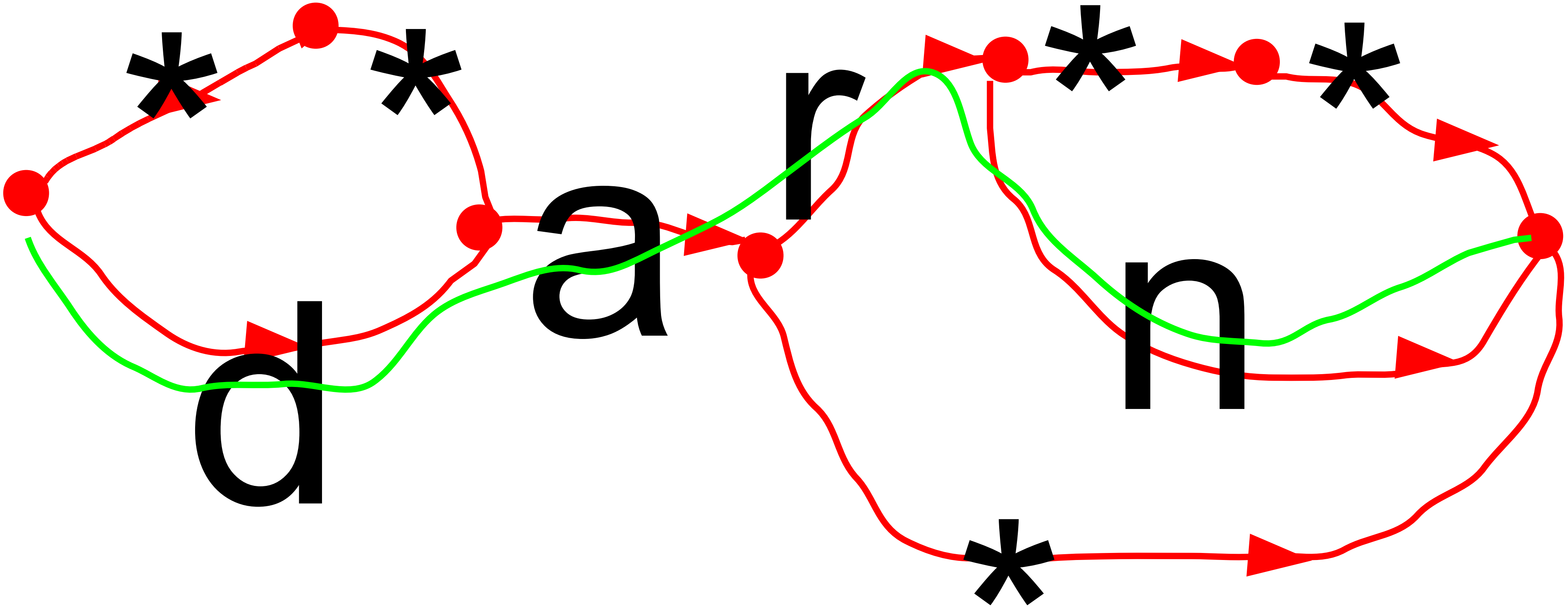
segmentation graph



segmentation graph

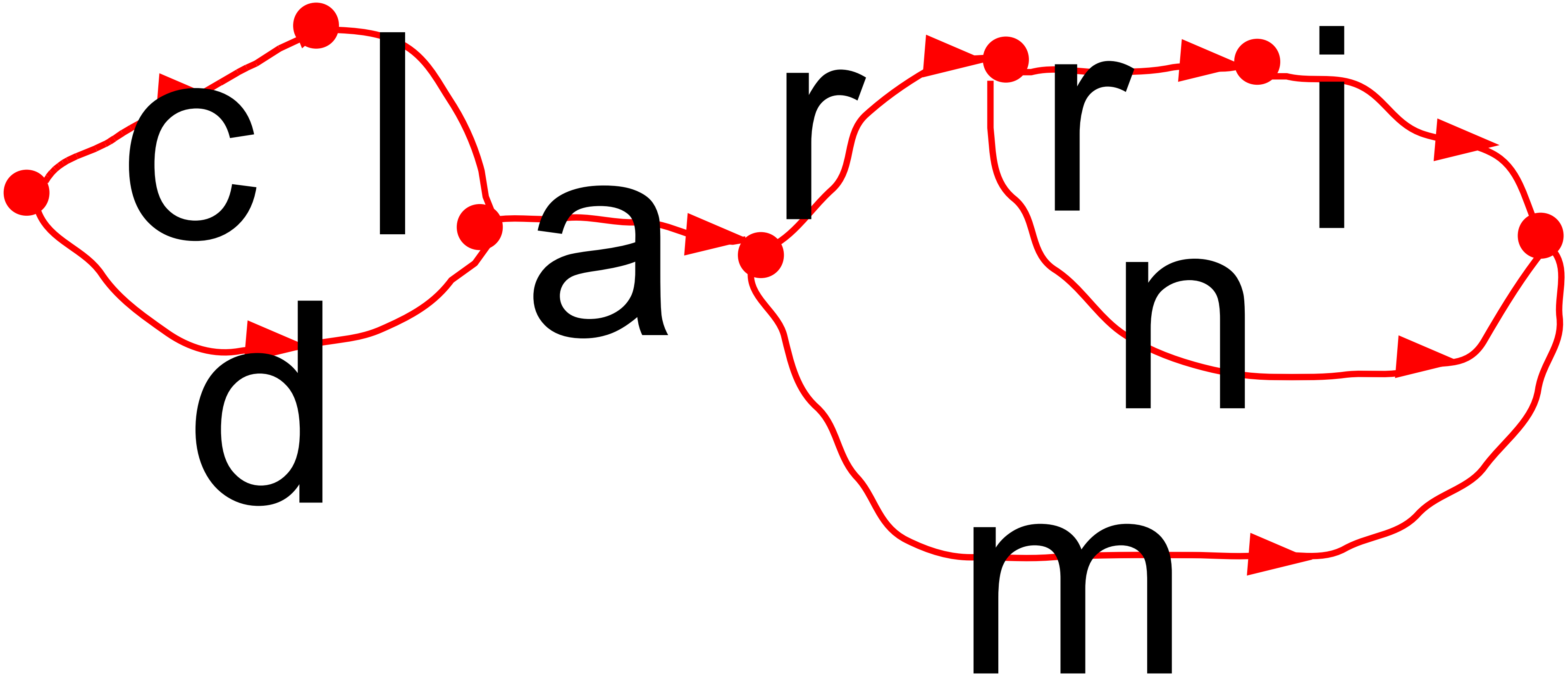
- **nodes**
  - cuts
- **edges**
  - images delimited by cuts
- **small encoding of possible segmentations**
  - exponential # segmentations possible
- **path through the graph**
  - a particular interpretation of the input

after classification



after classification

clarri, clarn, darri, darn, clam, dam



c l

d

a

r

r i

n

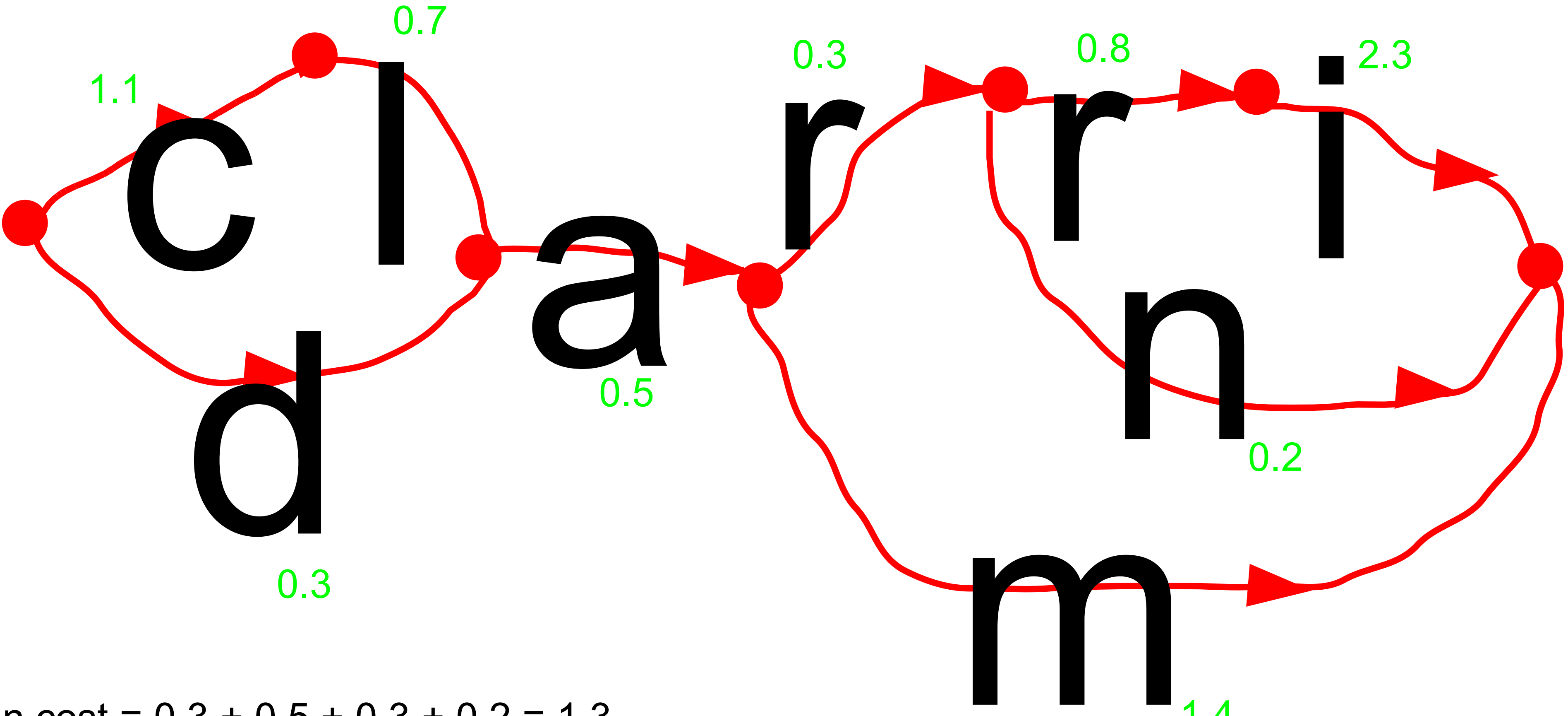
m

segmentation graph interpretation

- classifications aren't perfect
- each classification has a “cost”
- what's the best interpretation of the graph?

after classification

clarri, clarn, darri, darn, clam, dam



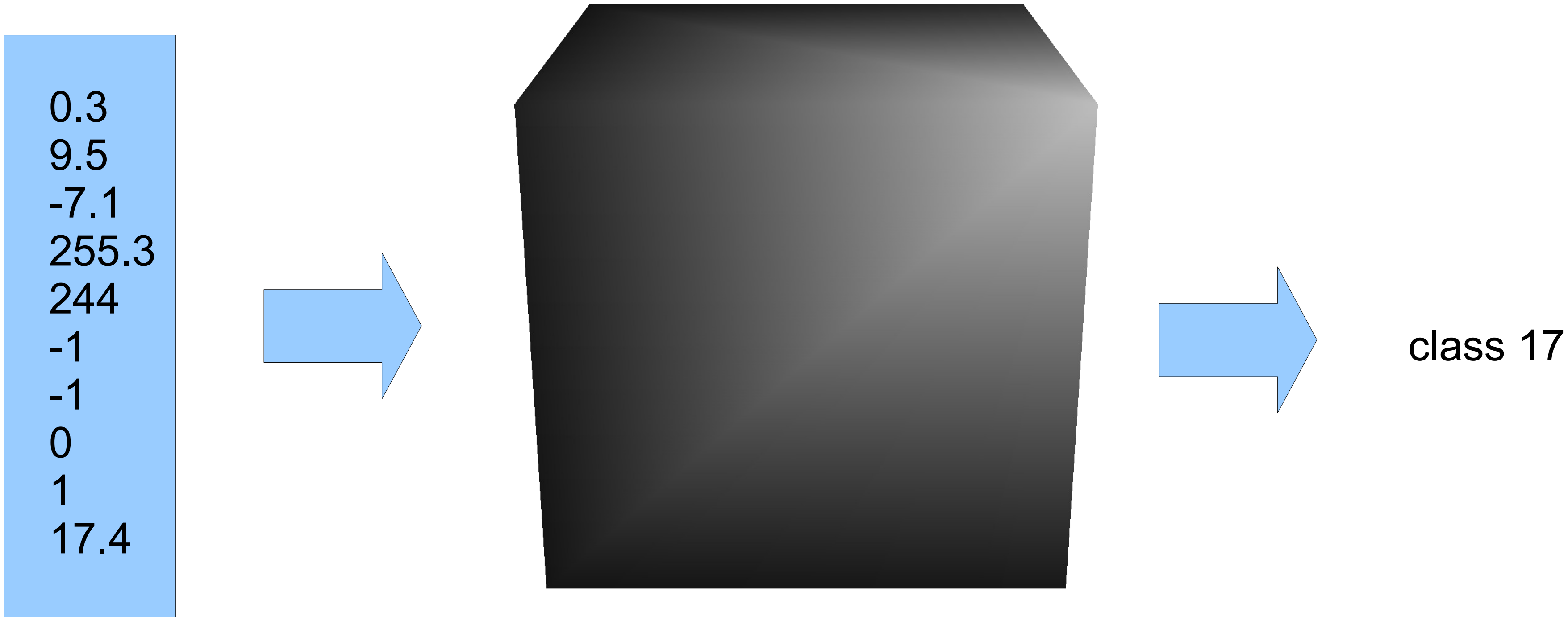
min cost = 0.3 + 0.5 + 0.3 + 0.2 = 1.3

questions

- how should we compute the “costs”?
- should we add them?
- can we compare different paths?
- what about paths with different lenghts?
- what about non-characters?
- what about dictionaries?
- relationship to HMMs?

better classification

classifiers



0.3
9.5
-7.1
255.3
244
-1
-1
0
1
17.4

class 17

nearest neighbor classifier

```
def classifier(x):

    best_c = -1
    best_d = infinity

    for v,c in training_examples:
        d = dissimilarity(v,x)
        if d<best_d:
            best_d = d
            best_c = c

    return best_c
```

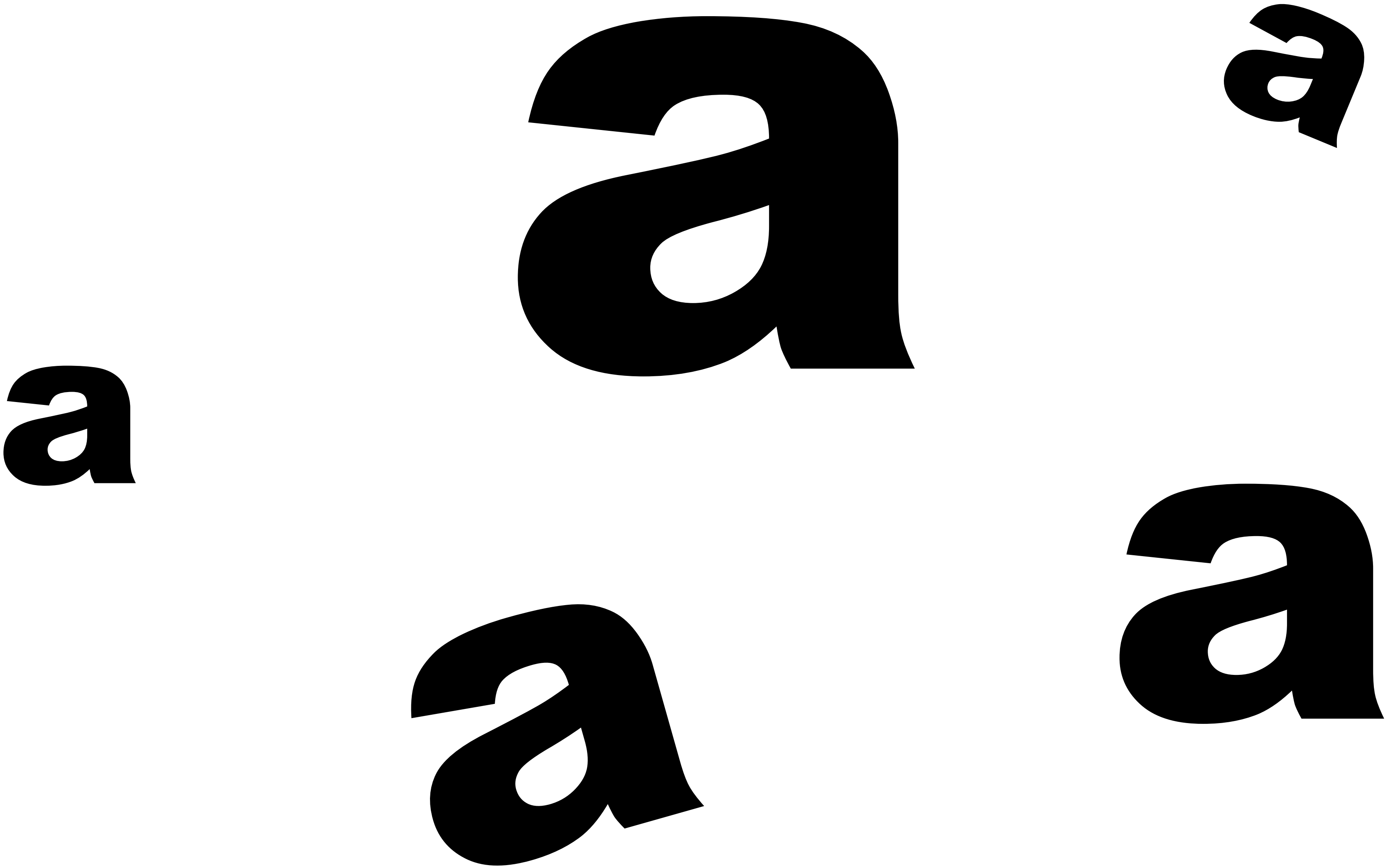
classifiers

- a classifier assigns classes to input patterns
- classifiers classify based on training data
- new patterns are often not identical to known ones, merely similar
- classifiers need to generalize to such novel patterns
- input patterns are typically real vectors of fixed dimension

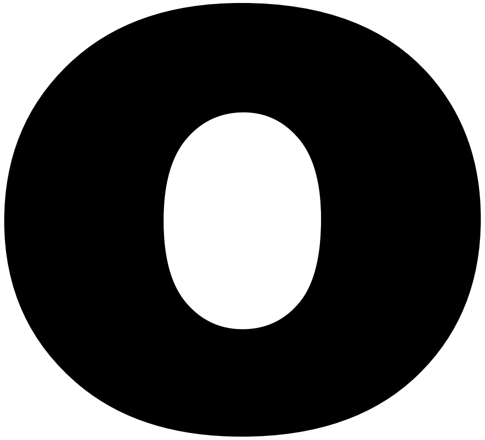
what makes classification hard?

- variations in position, size, orientation
- noise and image degradation
- large number of fonts
- context-dependence of interpretation

example: size / orientation



example: context



lower case “o”

capital “O”

digit “0”

mathematical circle symbol

example: context

To

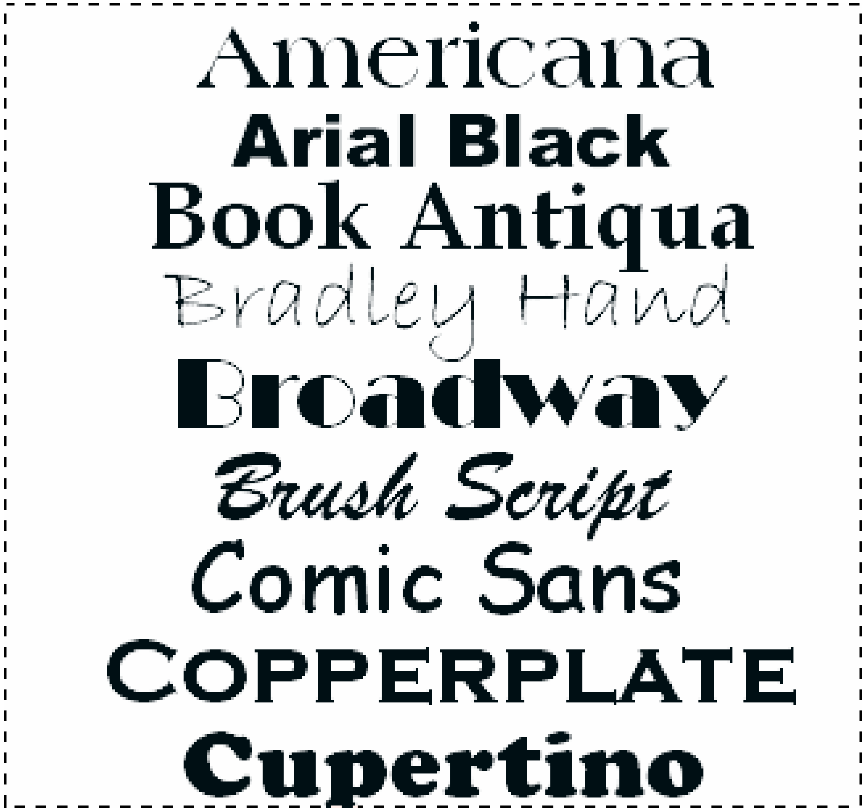
lower case “o”

~~capital “O”~~

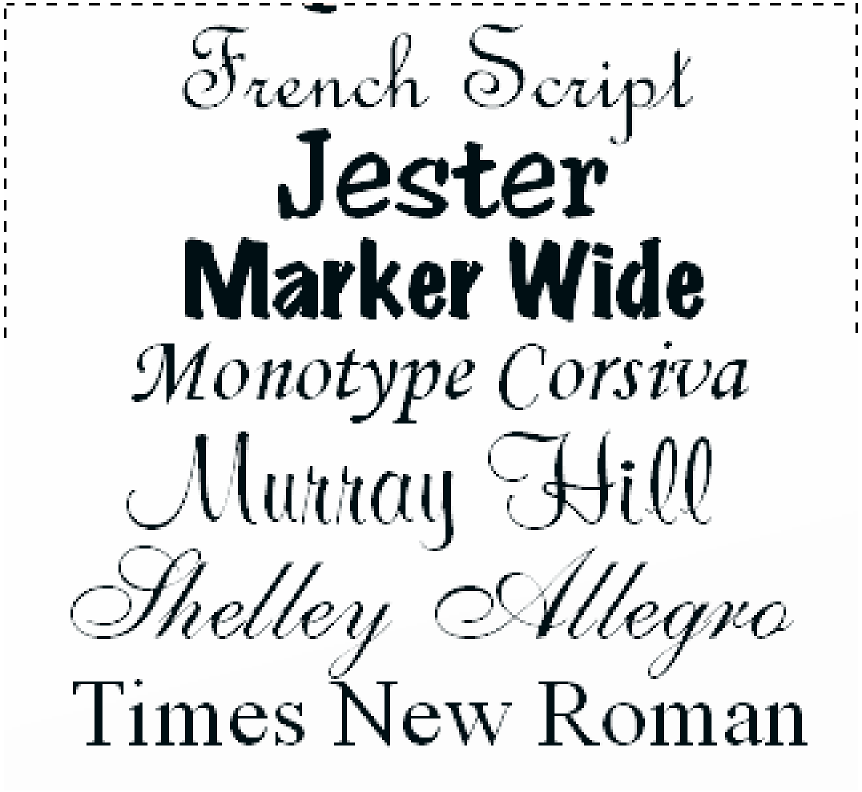
~~digit “0”~~

~~mathematical circle symbol~~

example: font variability



Americana
Arial Black
Book Antiqua
Bradley Hand
Broadway
Brush Script
Comic Sans
COPPERPLATE
Cupertino



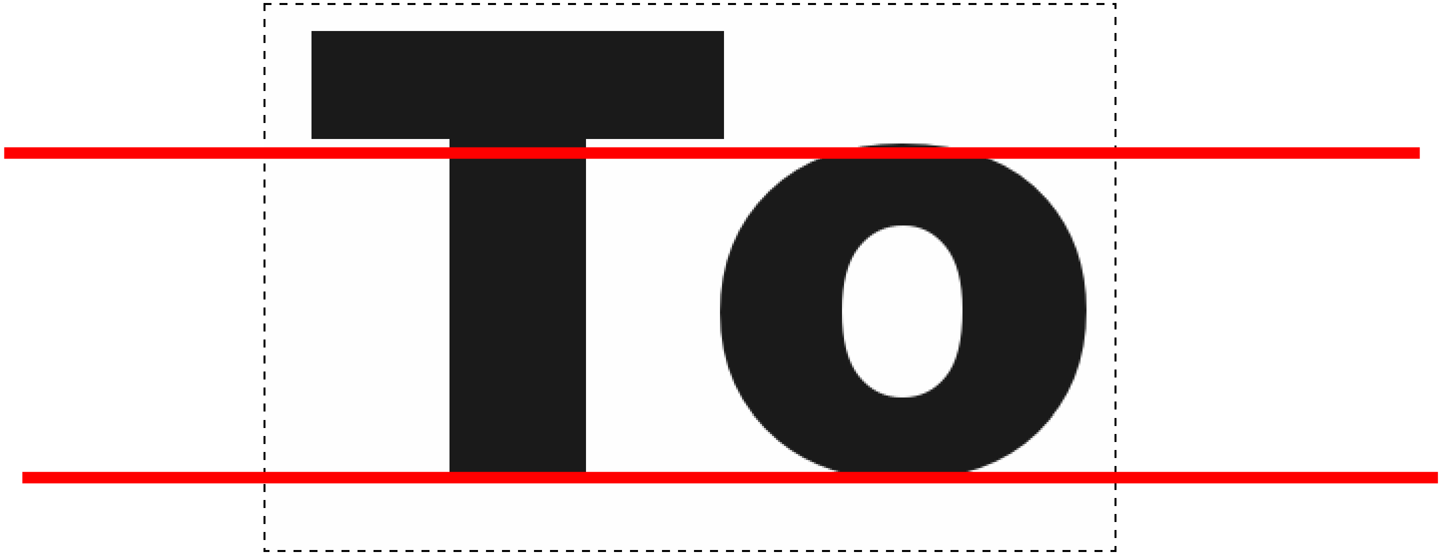
French Script
Jester
Marker Wide
Monotype Corsiva
Murray Hill
Shelley Allegro
Times New Roman

- similar to humans
- very different as bit patterns

dealing with variability & context

- noise removal & preprocessing
- text line modeling
- size normalization
- slant normalization
- context-dependent classifiers

text line modeling



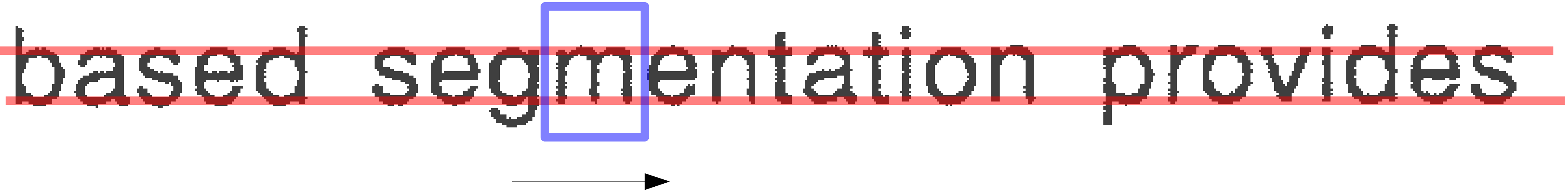
- **char 1: ascender**
- **char 2: no ascender/descender**

size normalization

- classifier takes fixed dimensional input
- characters come in all sizes
- absolute size/position is meaningless
- relative size/position is important

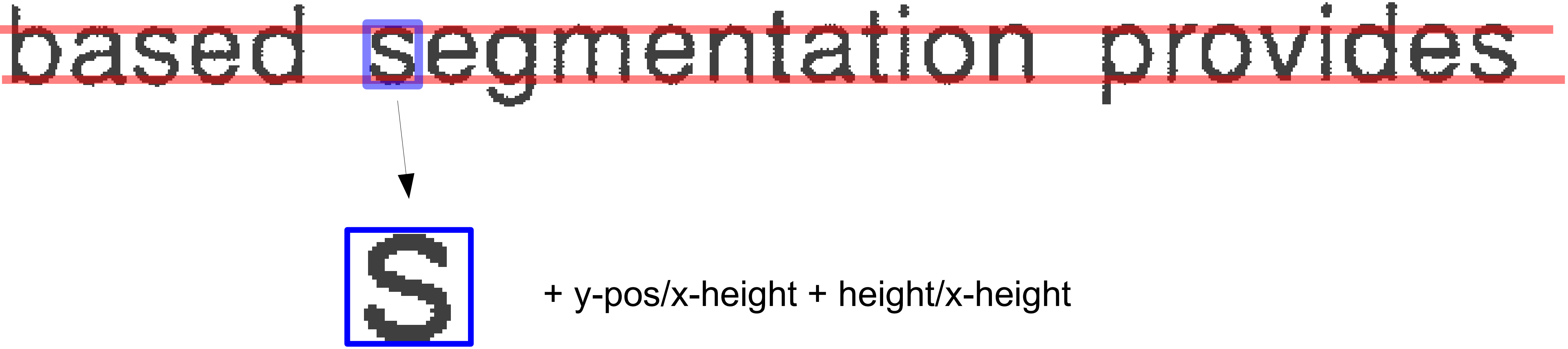
- how do we reconcile this?

approach 1: scanning



- **locate the baseline and x-height**
- **rescale the line s.t. the x-height is 10 pixels**
- **make the window 30x30**
- **place baseline at y=10, x-height at y=20**
  - (why not simply make the whole line 30 pixels high?)
- **center on characters and mask neighbors**

approach 2: extract, encode, resize


based segmentation provides
S
+ y-pos/x-height + height/x-height

- **extract each character**
- **rescale to fit into 30x30 bounding box**
- **encode y-position relative to baseline**
- **encode height**
- **units: x-height**

normalization by moments

- **fit into target image of size w x h**
- **compute the centroid of the character**
  - put the centroid at the center of the image
- **compute the trace of the covariance matrix**
  - rescale to given value
- **compute the slant of the character**
  - apply a skew transformation to achieve zero slant

normalization by bounding box

- **fit into target image of size w x h**

- **two possibilities**
  - rescale anisotropically so that the character fills the target
  - rescale by the minimum horizontal / vertical scale

- **more possibilities**
  - switch between anisotropic and isotropic
  - don't scale up, only scale down

image scaling

- **image scaling by subsampling**
  - doesn't work well by itself
  - think about thin lines
- **image scaling by antialiasing + subsampling**
  - the correct thing to do
  - works significantly better

summary

text line recognition

- **simple approach**
  - projection-based segmentation into characters
  - template matching for character recognition
  - dictionary + backtracking
- **better approach**
  - dynamic programming segmentation
  - segmentation graph
  - best path search
  - shape normalization prior to classification