Introduction to Normalized Correlation Pattern Matching

-- Optimization for OCR --

Tamotsu Tanabe

Topics

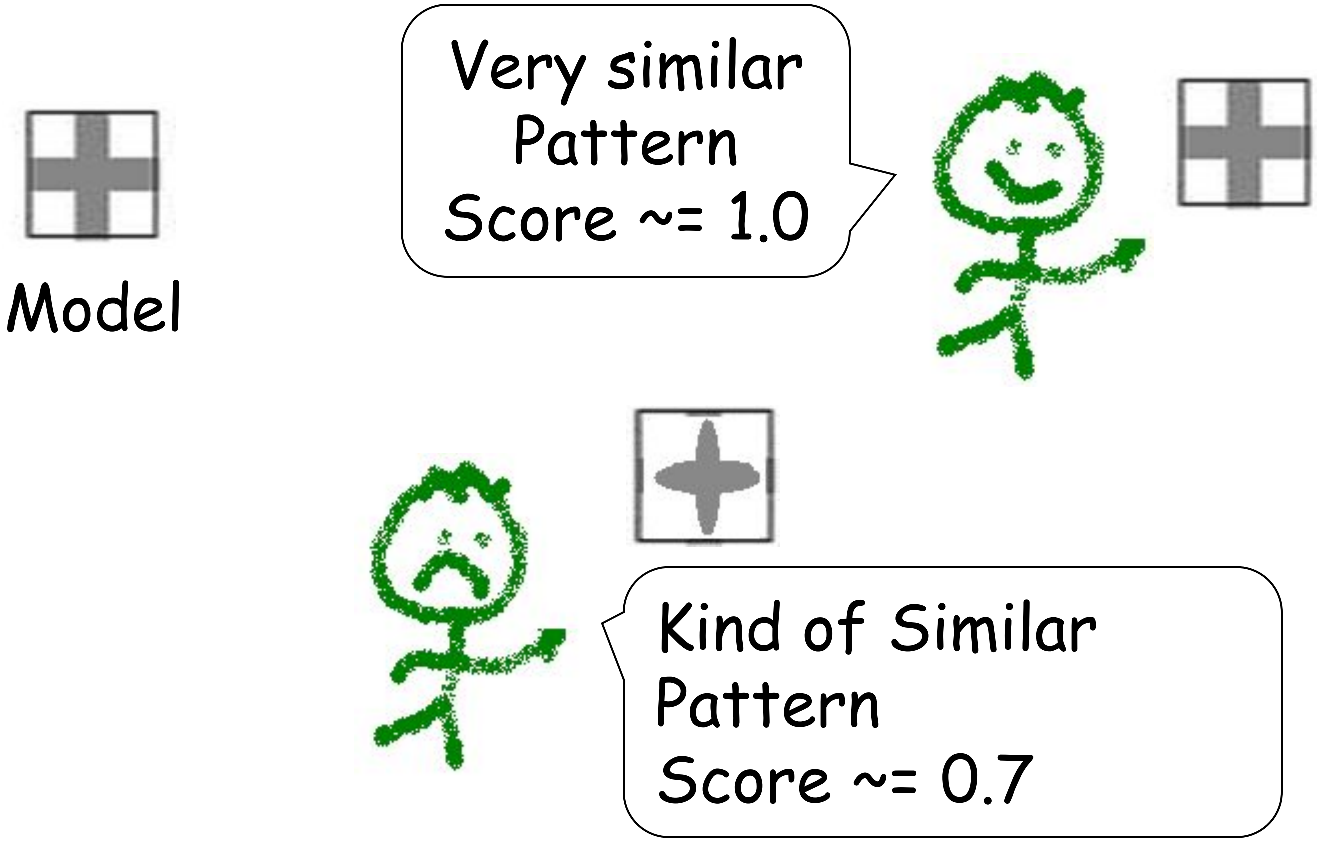
- Basics of “Normalized Correlation” pattern matching
- Normalized Correlation for OCR
  - Binary model pattern matching
  - Optimization for OCR

Basics

- What is Normalized Correlation Pattern Matching?
  – well-known technique for pattern matching
  – Pixel-based pattern matching

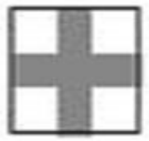
What does it do?

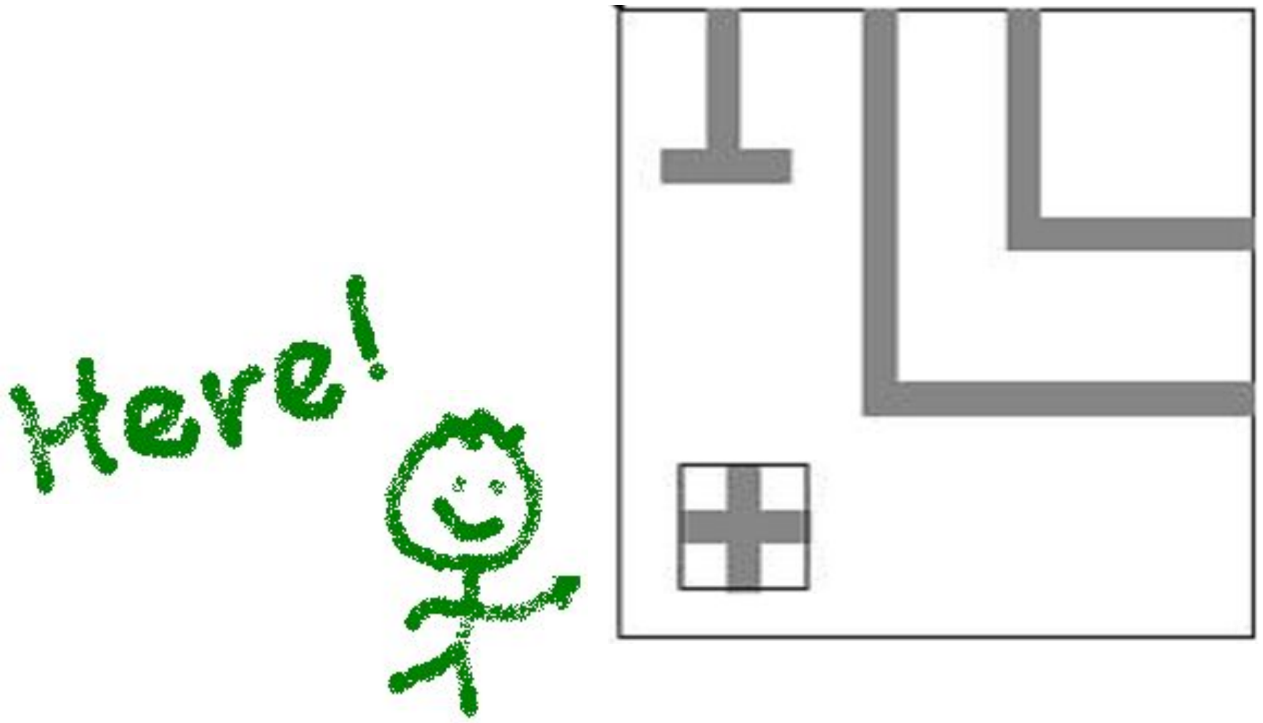
- Determines how “similar” a pattern is compared to a model.
- Similarity = Score (0 .. 1)


Model
Very similar Pattern Score ~= 1.0
Kind of Similar Pattern Score ~= 0.7

What does it do else?

- Also it can find a pattern which looks "similar" to the model in an image.
- Finding is same as iterating pattern matching at (x, y), where x = 0..~image width, y = 0.. ~image height


Model

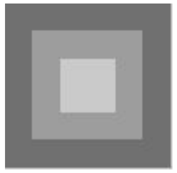


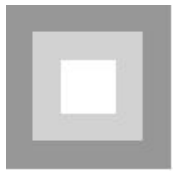
What’s good about it?

- Not affected by image’s brightness or contrast variation.

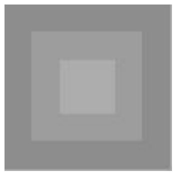
All the following patterns would return Score = 1.0!

Model

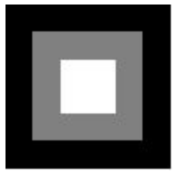




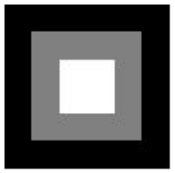
Brighter



Lower Contrast



Darker



Higher Contrast

How is it defined?

$$r = \frac{\sum_n (I_i - \overline{I})(M_i - \overline{M})}{\sqrt{\sum_n (I_i - \overline{I})^2}\sqrt{\sum_n (M_i - \overline{M})^2}}$$

I_i : grey value of image pixel, $\overline{I}$: average image grey value

M_i : grey value of model pixel, $\overline{M}$: average model grey value

n : number of pixels in the model

$r = -1.0 \sim 1.0$: Correation coefficient (Matching Score)

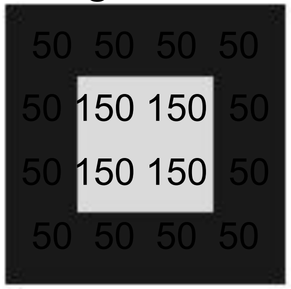
- Compute intensive
- Bigger r means more “similar”.
- r = 1.0 : perfect match

Example - 1

- Let's try to calculate correlation coefficient : r

$$r = \frac{\sum_n (I_i - \overline{I})(M_i - \overline{M})}{\sqrt{\sum_n (I_i - \overline{I})^2}\sqrt{\sum_n (M_i - \overline{M})^2}}$$

Image

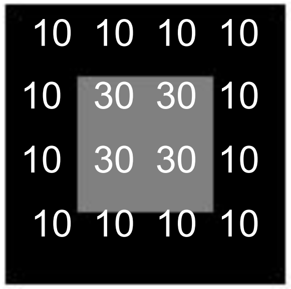


50	50	50	50
50	150	150	50
50	150	150	50
50	50	50	50

… well, it's a bit too much.

$Ii = \{50, 50, 50, 50,\quad 50, 150, 150, 50,\quad 50, 150, 150, 50,\quad 50, 50, 50, 50\}$

Model



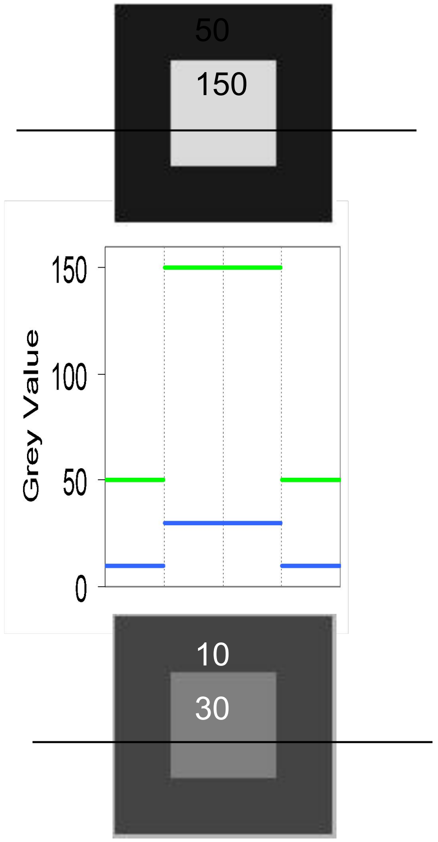
10	10	10	10
10	30	30	10
10	30	30	10
10	10	10	10

$Mi = \{10, 10, 10, 10,\quad 10, 30, 30, 10,\quad 10, 30, 30, 10,\quad 10, 10, 10, 10\}$

Example - 2

- For simplicity, just think about 1D pattern

$Ii = \{50, 150, 150, 50\}$



$Mi = \{10, 30, 30, 10\}$

- Calculate correlation coefficient : r

$$r = \frac{\sum_n (I_i - \overline{I})(M_i - \overline{M})}{\sqrt{\sum_n (I_i - \overline{I})^2}\sqrt{\sum_n (M_i - \overline{M})^2}}$$

$$\overline{I} = (50 + 150 + 150 + 50)/4 = 100$$

$$\overline{M} = (10 + 30 + 30 + 10)/4 = 20$$

$$r = \frac{\sum \{-50, 50, 50, -50\}\{-10, 10, 10, -10\}}{\sqrt{(-50)^2 + 50^2 + \ldots}\sqrt{(-10)^2 + 10^2 + \ldots}}$$

$$= \frac{-50 \cdot -10 + 50 \cdot 10 + 50 \cdot 10 + -50 \cdot -10}{\sqrt{2500 + 2500 + \ldots}\sqrt{100 + 100 + \ldots}}$$

$$= \frac{500 + 500 + 500 + 500}{\sqrt{10000}\sqrt{400}} = \frac{2000}{100 \cdot 20} = 1$$

- Painful, wasn't it?

More Examples

- See examples in Excel
- Notice that matching score is *not* affected by variations in Brightness or Contrast
- Try changing Image Grey Values and see how the score changes
- Also notice that what r = -1 means.

Take another look…

- Normalized correlation can be re-written as:

$$r = \frac{\sum (I_i - \overline{I})(M_i - \overline{M})}{\sqrt{\sum (I_i - \overline{I})^2} \sqrt{\sum (M_i - \overline{M})^2}}$$

$$= \frac{n\sum I_i M_i - \sum I_i \sum M_i}{\sqrt{n\sum I_i{}^2 - (\sum I_i)^2} \sqrt{n\sum M_i{}^2 - (\sum M_i)^2}}$$

- Calculation can be done in 1 pass (because you no longer need to calculate average gray values)

Normalized Correlation Review

- Based on pixel gray values of model and image
- Not affected by brightness or contrast
- Compute intensive

Topics

☑ Basics of “Normalized Correlation” pattern matching

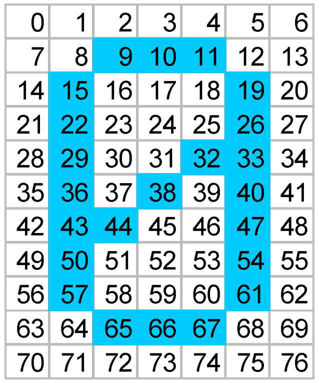
- Normalized Correlation for OCR
  - >> Binary model pattern matching
  - Optimization for OCR

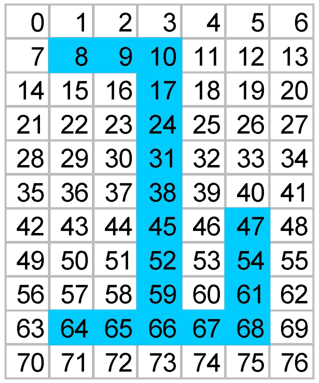
Normalized Correlation for OCR

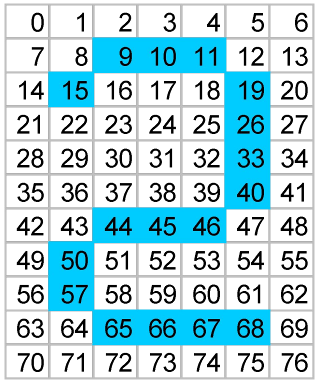
- Based on Normalized Correlation
- Must run fast!
  - Faster the matching is, you have more time to try other configurations (different image filtering, different lighting, etc)
  - For a single (x, y) position, you need to calculate matching scores for about 40 characters (Alphabets, Digits, etc)
- Normalized Correlation can be optimized, because:
  - Model is pre-defined (SEMI, OCR-A, etc)
  - Model is binary

Character Model

- SEMI Font
  - 7 x 11 pixels, including background pixels
  - Each pixel is either 0 or 1
  - Examples: “0”, “1”, and “2” are defined as:



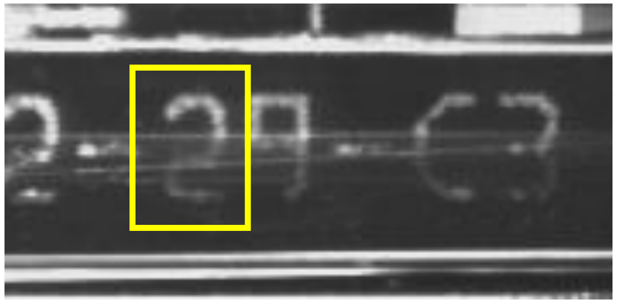




  - □ is “1”, otherwise “0”.

Matching with Multiple Chars - 1

- Assuming we already know the position of a character in image
- We need to calculate correlation scores for ~40 characters
- Q: How can we find the character with the highest score quickly?

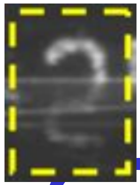


What is this character?
Need to compute
correlation values for all
the possible characters!
(about 40 for SEMI)

Matching with Multiple Chars - 2

A: Take a closer look at the correlation formula:

$$r = \frac{n\sum I_i M_i - \sum I_i \sum M_i}{\sqrt{n\sum I_i^2 - (\sum I_i)^2}\sqrt{n\sum M_i^2 - (\sum M_i)^2}}$$



Constant for a given (x, y) in image

Calculate only once for (x, y)

Constant for a given character

Can be calculated at compile time

How about this term?

Binary to Gray Correlation

Q: How can we calculate $\Sigma\, Ii\, Mi$ quickly?

$$r = \frac{n\sum I_i M_i - \sum I_i \sum M_i}{\sqrt{n\sum I_i^2 - (\sum I_i)^2}\sqrt{n\sum M_i^2 - (\sum M_i)^2}}$$

A: Recall that model is Binary - pixels are either 0 or 1:

$$\sum I_i M_i = I_0 M_0 + I_1 M_1 + \ldots + I_{76} M_{76}$$

$$= I_0(1 \text{ or } 0) + I_1(1 \text{ or } 0) + \ldots + I_{76}(1 \text{ or } 0)$$

Just a lot of additions!

Example: SEMI “0”

$$\sum I_i M_i = I_0(1 \text{ or } 0) + I_1(1 \text{ or } 0) + \ldots + I_{76}(1 \text{ or } 0)$$

$$= I[9] + I[10] + I[11]$$

$$+ I[15] + I[19]$$

$$+ I[22] + I[26]$$

$$+ I[29] + I[32] + I[33]$$

$$+ I[36] + I[38] + I[40]$$

$$+ I[43] + I[44] + I[47]$$

$$+ I[50] + I[54]$$

$$+ I[57] + I[61]$$

$$+ I[65] + I[66] + I[67]$$

0	1	2	3	4	5	6
7	8	**9**	**10**	**11**	12	13
14	**15**	16	17	18	**19**	20
21	**22**	23	24	25	**26**	27
28	**29**	30	31	**32**	**33**	34
35	**36**	37	**38**	39	**40**	41
42	**43**	**44**	45	46	**47**	48
49	**50**	51	52	53	**54**	55
56	**57**	58	59	60	**61**	62
63	64	**65**	**66**	**67**	68	69
70	71	72	73	74	75	76

Total 23 additions

(Assuming that image is mapped to i16[77] array)

Normalized Correlation for OCR Review

- Normalized Correlation calculation can be optimized because:
  - Model is binary : Calculations can mostly done by additions
  - Model is pre-defined : Calculations can be “hard-coded”

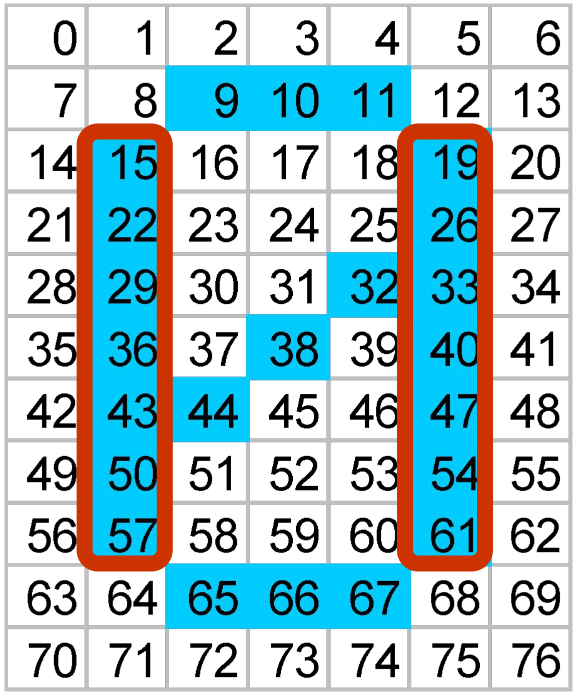
Topics

- ✓ Basics of “Normalized Correlation” pattern matching
- Normalized Correlation for OCR
  - ✓ Binary model pattern matching
  - >> Optimization for OCR

Further optimization

- Notice that there are lots of vertical bars in models
- Can we take advantage of that?

$$
\begin{aligned}
&\sum I_i M_i = I_0(1 \text{ or } 0) + I_1(1 \text{ or } 0) + \ldots + I_{76}(1 \text{ or } 0) \\
&= I[9] + I[10] + I[11] \\
&+ I[15] + I[19] \\
&+ I[22] + I[26] \\
&+ I[29] + I[32] + I[33] \\
&+ I[36] + I[38] + I[40] \\
&+ I[43] + I[44] + I[47] \\
&+ I[50] + I[54] \\
&+ I[57] + I[61] \\
&+ I[65] + I[66] + I[67]
\end{aligned}
$$



0	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31	32	33	34
35	36	37	38	39	40	41
42	43	44	45	46	47	48
49	50	51	52	53	54	55
56	57	58	59	60	61	62
63	64	65	66	67	68	69
70	71	72	73	74	75	76

Vertical Integral Image

- Each pixel holds the sum of the pixels above + itself
  (for convenience, use indices [77]..[153])
- For example:
  - `I[92]  =          I[15] + I[8] + I[1]`
  - `I[99]  = I[22] + I[15] + I[8] + I[1]`
- *Σ Ii Mi* for SEMI "0" can be re-written as:

$$\sum I_i M_i = I_0(1 \text{ or } 0) + I_1(1 \text{ or } 0) + \ldots + I_{76}(1 \text{ or } 0)$$

$$= I[9] + I[10] + I[11]$$
$$+ I[134] - I[85]$$
$$+ I[138] - I[89]$$
$$+ I[32]$$
$$+ I[38]$$
$$+ I[44]$$
$$+ I[65] + I[66] + I[67]$$

Total 13 add/sub operations

Original version had 23 add operations

0	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31	32	33	34
35	36	37	38	39	40	41
42	43	44	45	46	47	48
49	50	51	52	53	54	55
56	57	58	59	60	61	62
63	64	65	66	67	68	69
70	71	72	73	74	75	76

Vertical Integral Image

77	78	79	80	81	82	83
84	85	86	87	88	89	90
91	92	93	94	95	96	97
98	99	100	101	102	103	104
105	106	107	108	109	110	111
112	113	114	115	116	117	118
119	120	121	122	123	124	125
126	127	128	129	130	131	132
133	134	135	136	137	138	139
140	141	142	143	144	145	146
147	148	149	150	151	152	153

Code Sample

Here is a code snippet:

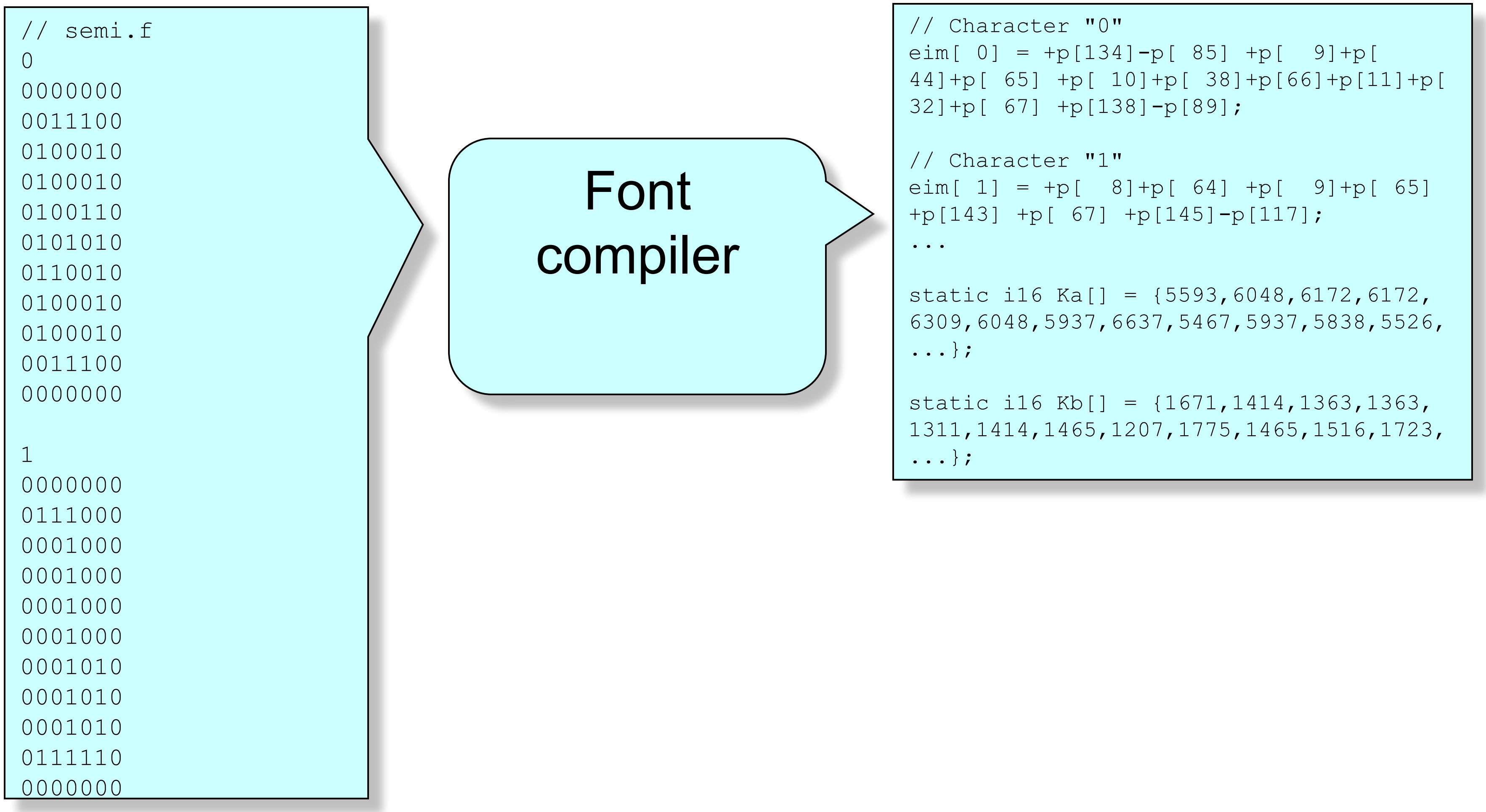
This can be hard-coded!

```
// Character "0"
eim[ 0] = +p[134]-p[ 85] +p[  9]+p[ 44]+p[ 65] +p[ 10]+p[ 38]+p[ 66]
          +p[ 11]+p[ 32]+p[ 67] +p[138]-p[ 89];

// Character "1"
eim[ 1] = +p[  8]+p[ 64] +p[  9]+p[ 65] +p[143] +p[ 67] +p[145]-p[117];

// Character "2"
eim[ 2] = +p[ 15]+p[ 50]+p[ 57] +p[  9]+p[ 44]+p[ 65] +p[ 10] +p[ 45]
          +p[ 66] +p[ 11]+p[ 46]+p[ 67] +p[117]-p[ 89]+p[ 68];

// Character "3"
eim[ 3] = +p[  8]+p[ 64] +p[  9]+p[ 37]+p[ 65] +p[ 10]+p[ 38]+p[ 66]
          +p[ 11]+p[ 39]+p[ 67] +p[110]-p[ 89]+p[138]-p[117];

// Character "4"
eim[ 4] = +p[120] +p[ 44] +p[ 45] +p[ 46] +p[145]-p[ 96];
...
```

NOTE: Creating Integral Image increases the time, but the total calculation time is shorter because it reduces the number of add/sub operations

Font-compiler tool

- Generates C code from Font definition file
  - C program code to calculate $\Sigma\ Ii\ Mi$
  - Calculate constant values for $\sum M_i\ ,\ \sqrt{n\sum M_i^{\,2} - (\sum M_i)^2}$



```
// semi.f
0
0000000
0011100
0100010
0100010
0100110
0101010
0110010
0100010
0100010
0011100
0000000

1
0000000
0111000
0001000
0001000
0001000
0001000
0001010
0001010
0001010
0111110
0000000
```

```
// Character "0"
eim[ 0] = +p[134]-p[ 85] +p[  9]+p[
44]+p[ 65] +p[ 10]+p[ 38]+p[66]+p[11]+p[
32]+p[ 67] +p[138]-p[89];

// Character "1"
eim[ 1] = +p[  8]+p[ 64] +p[  9]+p[ 65]
+p[143] +p[ 67] +p[145]-p[117];
...

static i16 Ka[] = {5593,6048,6172,6172,
6309,6048,5937,6637,5467,5937,5838,5526,
...};

static i16 Kb[] = {1671,1414,1363,1363,
1311,1414,1465,1207,1775,1465,1516,1723,
...};
```

OCR correlation - Summary

- Steps to calculate OCR correlation for a given (x, y)

$$r = \frac{n\sum I_i M_i - \sum I_i \sum M_i}{\sqrt{n\sum I_i^{\,2} - (\sum I_i)^2}\sqrt{n\sum M_i^{\,2} - (\sum M_i)^2}}$$

<u>Compile time (Run Fontgen once when you added a new font)</u>

- For each char, calc: $\sum M_i \,, \sqrt{n\sum M_i^{\,2} - (\sum M_i)^2}$
- Generate program to calc $\sum$ *Ii Mi* for each char:

$$0: \sum I_i M_i = I[9] + I[10] + I[11] + \ldots,$$
$$1: \sum I_i M_i = I[8] + I[9] + \ldots$$

<u>Runtime</u>

- At (x, y) in image, calc: $\sum I_i \,, \sqrt{n\sum I_i^{\,2} - (\sum I_i)^2}$
- At (x, y) in image, for each char, calc:

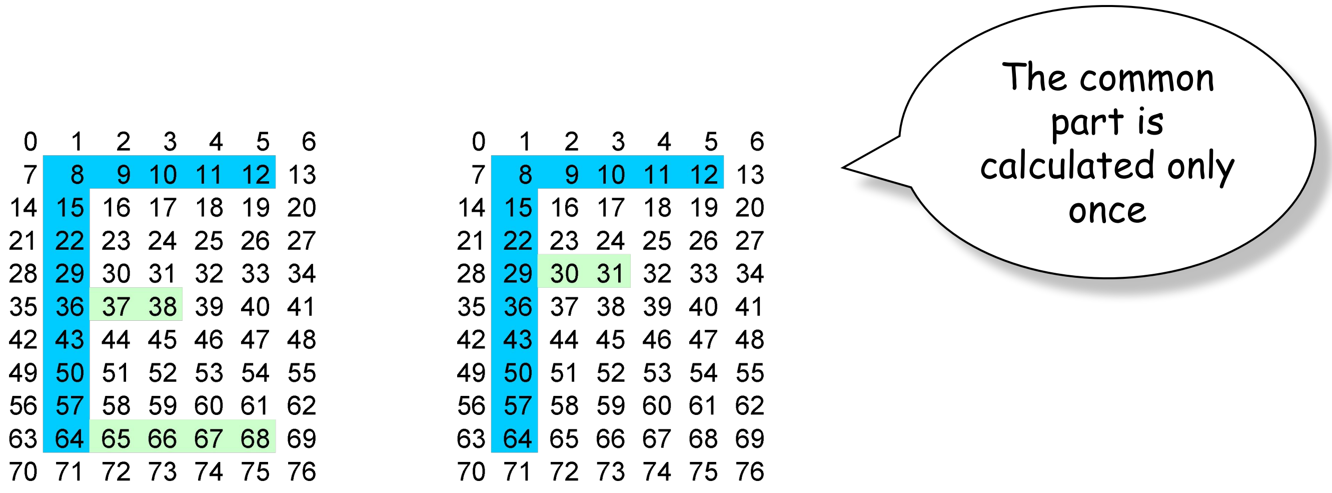
$$0: \sum I_i M_i = I[9] + I[10] + I[11] + \ldots,$$
$$1: \sum I_i M_i = I[8] + I[9] + \ldots$$

Topics

- ✓ Basics of “Normalized Correlation” pattern matching
- ✓ Normalized Correlation for OCR
  - ✓ Binary model pattern matching
  - ✓ Optimization for OCR

A few more...

- Font_compile (enhanced)
  - Generates faster code for $\Sigma\, Ii\ Mi$ than fontgen
    - Average less than 9 add ops per character for SEMI font
  - Doesn't use Integral Image
  - Minimizes number of additions by combining partial additions
    - For example, "E" and "F" have lots in common



Patterns as Vectors

- By “normalizing” Ii and Mi, the correlation coefficient can be re-written as follows

$$r = \frac{\sum (I_i - \overline{I})(M_i - \overline{M})}{\sqrt{\sum (I_i - \overline{I})^2}\sqrt{\sum (M_i - \overline{M})^2}} \qquad A_i = I_i - \overline{I}_i,\ B_i = M_i - \overline{M}_i$$

$$= \frac{\sum A_i B_i}{\sqrt{\sum A_i^2}\sqrt{\sum B_i^2}} \qquad A = \{A_1, A_2, \ldots A_n\},\ B = \{B_1, B_2, \ldots B_n\}$$

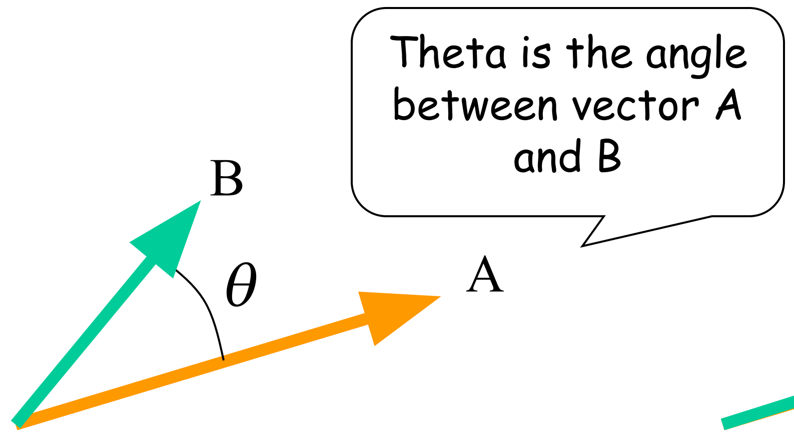
$$= \frac{A \cdot B}{|A||B|}$$

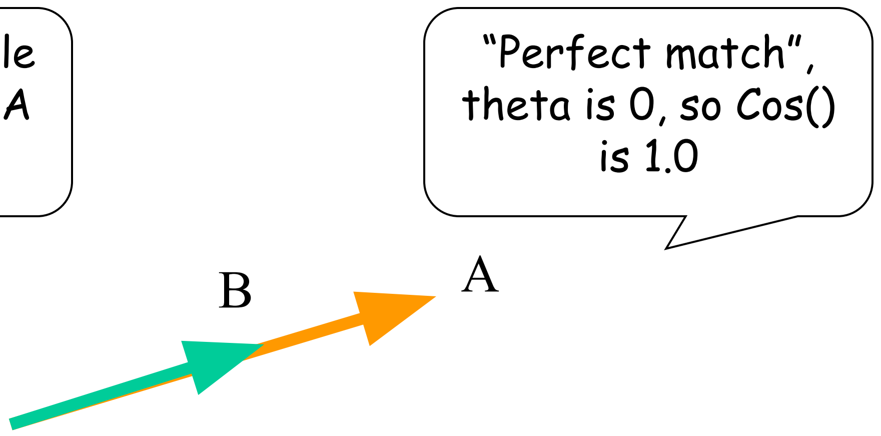
$A \cdot B$ is called "inner product", defined as $|A||B|\cos\theta$

$|A|$ is called size or norm of vector A

$$= \cos\theta$$

- A, B can be considered as n-dimensional vectors


B
θ
A
Theta is the angle between vector A and B


B
A
“Perfect match”, theta is 0, so Cos() is 1.0