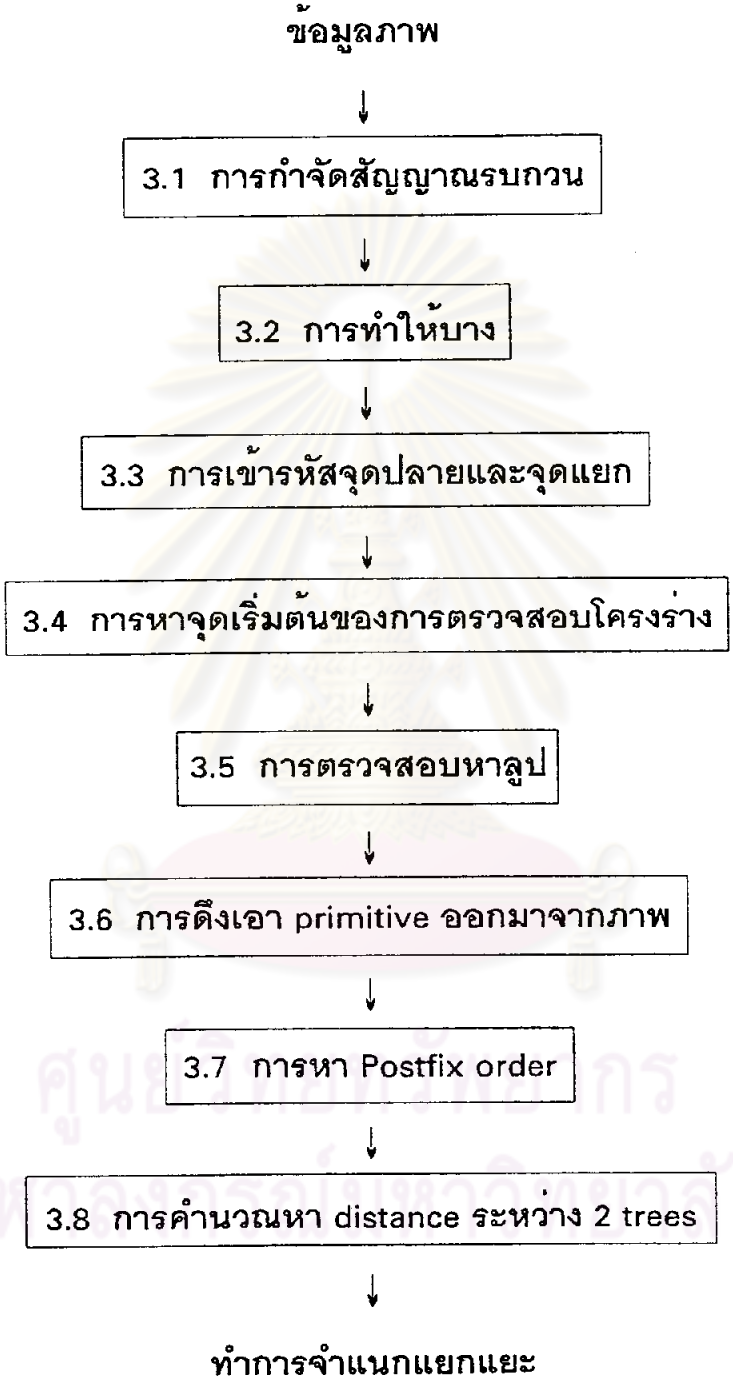
บทที่ 3

กระบวนการ (Processing)

ในงานวิจัยนี้ขั้นตอนการรู้จำที่ใช้จะอาศัยแนวทางของซินแทกติก ซึ่งจะอ้างอิงกระบวนการตามบล็อกไดอะแกรมที่แสดงในรูปที่ 2.2 โดยระบบการรู้จำในการศึกษานี้จะมีขั้นตอนดังแสดงในรูปที่ 3.1 ซึ่งประกอบด้วยกระบวนการดังนี้คือ การกำจัดสัญญาณรบกวน (Noise Reduction), การทำให้บาง (Thinning), การเข้ารหัสจุดปลายและจุดแยก, การหาจุดเริ่มต้นของการตรวจสอบโครงร่าง, การตรวจสอบหาลูป, การดึงเอา primitive ออกมาจากภาพ, การหาลำดับ postfix, และ การคำนวณหา distance ระหว่าง 2 trees โดยกระบวนการกำจัดสัญญาณรบกวน และ การทำให้บาง จะเป็นส่วนบล็อก Preprocessing ดังในรูปที่ 2.2 ส่วนกระบวนการหาจุดเริ่มต้นของการตรวจสอบโครงร่าง การตรวจสอบหาลูป การดึงเอา primitive ออกมาจากภาพ จะเป็นส่วนบล็อก Pattern Representation และกระบวนการหาลำดับ postfix กับ การคำนวณหา distance ระหว่าง 2 trees จะเป็นในส่วนของ Classification

ในแต่ละกระบวนการที่ได้กล่าวถึงนี้จะมีวิธีการและอัลกอริทึมในการทำงานแตกต่างกันออกไป ซึ่งในบทนี้จะขอกล่าวถึงรายละเอียดของกระบวนการที่ใช้ในแต่ละขั้นตอนดังในหัวข้อที่จะกล่าวถึงต่อไป



รูปที่ 3.1 บล็อกไดอะแกรมของระบบการรู้จำที่ใช้

3.1 <u>การกำจัดสัญญาณรบกวน</u> (Noise reduction) [9]

สัญญาณรบกวน (noise) ที่ปรากฏในรูปตัวอักษรที่มักจะพบแบ่งได้เป็น 2 ประเภท คือ สัญญาณรบกวนที่มีลักษณะเป็นจุดอิสระ (Isolate point) และ สัญญาณรบกวนที่มีลักษณะเป็นรูอยู่ภายในภาพ (Isolate hold) การตรวจสอบหาจุดที่คาดว่าจะเป็นสัญญาณรบกวนในแบบที่กล่าวมานี้จะสามารถทำได้โดยการใช้วินโดว์ (window) ขนาด 3 x 3 จุดภาพ ดังแสดงในรูปที่ 3.2 สแกนไปบนรูปแบบสองระดับ (ที่มีค่าเป็น จุดขาว กับ จุดดำ หรือที่แทนค่าเป็น 0 กับ 1) ที่มีสัญญาณรบกวนปรากฏอยู่

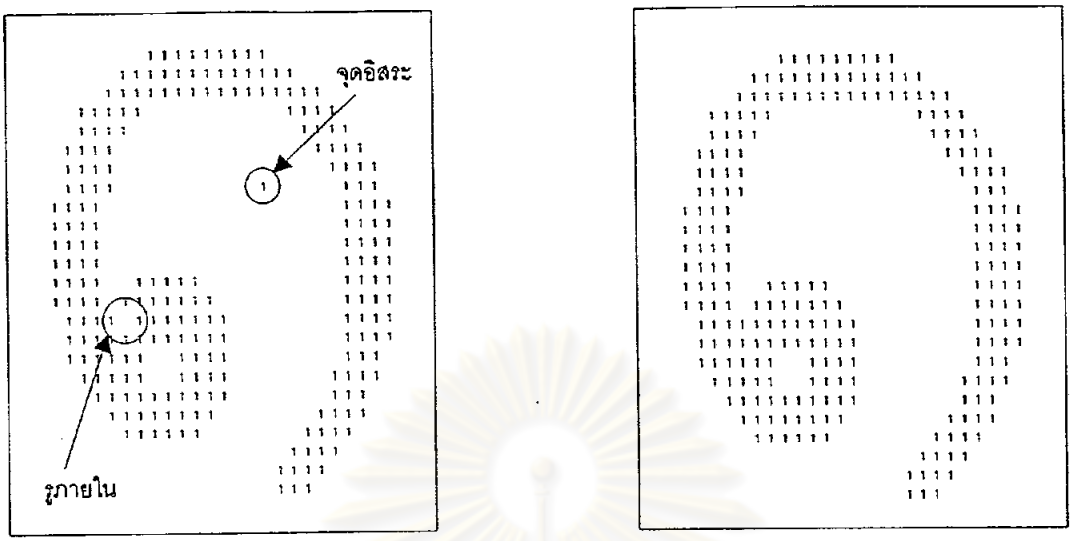
X_3	X_2	X_1
X_4	X	X_0
X_5	X_6	X_7

รูปที่ 3.2 วินโดว์ขนาด 3 x 3

โดยจุดภาพ X ในรูปที่ 3.2 จะเป็นจุดภาพที่ต้องการตรวจสอบ ส่วนจุดภาพ X_0 ถึง X_7 จะเป็นจุดภาพข้างเคียงทั้ง 8 ทิศทางของจุดภาพ X ซึ่งการตรวจสอบจะทำโดยการบวกรวมค่าจุดข้างเคียงของจุด X ทั้ง 8 จุดภาพ ดังสมการ (3.1)

$$sum = X0+X1+X2+X3+X4+X5+X6+X7 \qquad (3.1)$$

ถ้าค่า sum ที่ได้มีค่าเท่ากับ 0 หรือ 1 จะถือว่าจุด X นั้นอาจจะเป็นจุดที่เป็นสัญญาณรบกวนแบบจุดอิสระได้ ซึ่งเราจะทำการแก้ไขได้โดยการกำหนดค่าของจุดภาพ X นั้นมีค่าเท่ากับ 0 แต่ถ้าค่า sum ที่ได้มีค่าเท่ากับ 7 หรือ 8 เราจะถือว่าจุด X นั้นอาจจะเป็นจุดที่เป็นสัญญาณรบกวนแบบรูภายใน ซึ่งก็จะทำการแก้ไขได้โดยการกำหนดค่าของจุดภาพ X ให้มีค่าเท่ากับ 1 ตัวอย่างการกำจัดสัญญาณรบกวนด้วยกระบวนการนี้จะเป็นดังรูปที่ 3.3



(ก) (ข)

รูปที่ 3.3 ตัวอย่างการกำจัดสัญญาณรบกวน
(ก) ภาพต้นแบบที่มีสัญญาณรบกวน
(ข) ภาพภายหลังจากกำจัดสัญญาณรบกวนแล้ว

3.2 <u>การทำให้บาง</u> (Thinning) [9],[16]-[20]

โดยส่วนใหญ่แล้วตัวอักษรจะมีความหนาของลายเส้นปรากฏอยู่ ซึ่งในระบบการรู้จำในการศึกษานี้จะต้องการเพียงส่วนที่เป็นโครงร่างของตัวอักษรเท่านั้น ดังนั้นจึงจำเป็นต้องมีกระบวนการทำให้บาง (thinning) ขึ้น

การทำให้บาง (thinning) จะเป็นกรรมวิธีที่ใช้สำหรับลดความหนาของลายเส้นบนภาพให้มีความหนาลดลงเหลือเพียง 1 จุดภาพ ทั้งนี้เพื่อประโยชน์ในการลดจำนวนของข้อมูลภาพ และ ให้ได้เฉพาะส่วนที่เป็นโครงร่าง (skeleton) ของภาพเท่านั้น สำหรับหลักการในการหาโครงร่างของภาพนั้นจะทำได้โดยการสแกนไปบนทุกจุดของภาพ เพื่อทำการค้นหา จุดที่เป็นโครงร่างของภาพ และ จุดที่เป็นส่วนขอบของภาพ จากนั้นจะทำการกำจัดจุดที่เป็นขอบที่สามารถกำจัดได้ทิ้งไปทั้งหมด ทำเช่นนี้ซ้ำจนกระทั่งเหลือเพียงส่วนที่เป็นโครงร่างของภาพเท่านั้น

ได้มีการศึกษาค้นคว้าและพัฒนาอัลกอริทึมการทำให้บางขึ้นมามากมาย ซึ่งในการศึกษานี้จะมีอัลกอริทึมการทำให้บางปรากฏอยู่ 2 ตัว คือ Classical thinning algorithm ของ Pavlidis[16] กับ Thinning algorithm ของ Naccache และ Shinghal [9] แต่ในขั้นตอนการรู้จำ

จะใช้ Classical thinning algorithm เท่านั้น เนื่องจากว่าจากการทดสอบใช้งานกับข้อมูลภาพตัวเลขไทยที่นำมาทดสอบการรู้จำแล้ว พบว่า Classical thinning จะให้ผลการทำให้บางที่ดีกว่า คือ จะไม่มีการขาดของลายเส้นที่เป็นโครงร่าง และ จะมีลายเส้นที่เกินเกิดขึ้นน้อยกว่า ส่วนใน Thinning algorithm ของ Naccache และ Shinghal จะเกิดการขาดของลายเส้นในภาพตัวอักษรบางตัว และ มักจะมีลายเส้นที่เกินเกิดขึ้น ดังนั้นในหัวข้อนี้จะขอกล่าวถึง Classical thinning เท่านั้น ส่วนรายละเอียดของ Parallel thinning algorithm ดูได้ในภาคผนวก ก.

อัลกอริทึมของ Classical thinning จะเป็นดังแสดงใน Algorithm 3.1 โดยในการอ้างถึงจุดต่าง ๆ ในภาพของ Classical thinning ค่าตัวเลขที่ใช้จะแทนความหมายดังต่อไปนี้

0 แทน จุดว่างเปล่าบนภาพ (blank pixel)

1 แทน จุดภาพ (image pixel)

2 แทน จุดที่เป็นโครงร่างของภาพ (skeletal pixel)

3 แทน จุดที่สามารถกำจัดทิ้งได้ (deletable pixel)

ในการตรวจสอบแต่ละจุดภาพก็จะทำโดยการใช้วินโดว์ขนาด 3 x 3 จุดภาพดังในรูปที่ 3.2 โดยรูปแบบในการพิจารณาหาจุดที่เป็นโครงร่างของภาพจะเป็นดังแสดงในรูปที่ 3.4 ซึ่งในแต่ละกลุ่มที่แทนด้วยตัวอักษร A หรือ B จะต้องมีจุดที่มีค่าไม่เป็น 0 อย่างน้อย 1 จุดจึงจะถือว่าจุด P นั้นเป็นจุดที่เป็นโครงร่างของภาพ ส่วนขอบของภาพนั้นจะแบ่งได้ออกเป็น 4 อย่าง คือ ขอบบน, ขอบล่าง, ขอบซ้าย, และ ขอบขวา จุดที่เป็นขอบของภาพในแต่ละขอบจะพิจารณาเป็นดังนี้ (ดูรูปที่ 3.2 ประกอบ)

จุด X จะเป็น "ขอบบน" ของภาพเมื่อจุดภาพข้างเคียง X_2 มีค่าเป็น 0

จุด X จะเป็น "ขอบล่าง" ของภาพเมื่อจุดภาพข้างเคียง X_6 มีค่าเป็น 0

จุด X จะเป็น "ขอบซ้าย" ของภาพเมื่อจุดภาพข้างเคียง X_4 มีค่าเป็น 0

จุด X จะเป็น "ขอบขวา" ของภาพเมื่อจุดภาพข้างเคียง X_0 มีค่าเป็น 0

A	A	A
0	P	0
B	B	B

A	A	A
A	P	0
A	0	2

รูปที่ 3.4 รูปแบบในการตรวจสอบหาจุดที่เป็นโครงร่างของภาพ

Algorithm 3.1 Classical Thinning Algorithm.

Notation: I is the input image. P is the set of patterns of neighborhoods of skeletal pixels shown in figure 3.4 , including also a 90^{o} rotation of the first pattern and three 90^{o} rotations of the second pattern. Flag remain, when set of true, is used to denote that nonskeletal pixels may remain. Flag skel is set to true when the neighborhood of a pixel matches one of the patterns in P. A "one" in the pattern matches any nonzero element of the neighborhood.

1. Set the flag remain to true.
2. While remain is true do step 3 - 12.
   Begin.
3. Set remain to false. {No change has been made.}
4. For j = 0, 2, 4, and 6 do step 5 - 12.
   Begin.
5. For all pixels p of the image I do step 6 - 10.
   Begin.
6. If p is 1 and if its j - neighbor is 0 then do steps 7 - 10.
   Begin.
7. Set flag skel to false.
8. For all patterns P do step 9.
   Begin
9. If the neighborhood of p matches the pattern P, then set skel to true and exit from the loop.
   End.

10. if the flag skel is true, then set p to 2 {skeletal pixel}, else set p to 3 { deletable pixel} and also set remain to true.

End.

End.

11. For all pixels p of I do step 12.

Begin.

12. If p is 3, then set p equal to 0.

End.

End.

End.

13. End of Algorithm.

สำหรับตัวอย่างการทำงานของ classical thinning algorithm นี้ในการทำภาพตัวอักษรให้บางจะเป็นดังรูปที่ 3.5

รูปที่ 3.5 ผลของการทำให้บางด้วย Classical thinning algorithm

"-" แทนจุดภาพที่ถูกลบทิ้ง และ "0" แทนจุดภาพที่เป็นโครงร่าง

3.3 การเข้ารหัสจุดปลายและจุดแยก

เมื่อข้อมูลภาพได้ผ่านกระบวนการต่าง ๆ จนกระทั่งเหลือเฉพาะส่วนที่เป็นโครงร่างแล้ว ก่อนที่จะทำการตรวจสอบไปตามโครงร่างของตัวเลข จะต้องทำการค้นหาส่วนที่เป็นจุดปลาย (end point) และ จุดแยก (branch point) เสียก่อน เพื่อจะได้ทราบว่าโครงร่างของภาพตัวเลขนั้นในแต่ละช่วงของลายเส้นมีจุดสิ้นสุดหรือมีการเชื่อมต่อกับลายเส้นอื่นที่จุดใดของโครงร่าง ซึ่งวิธีการตรวจสอบหาจุดปลายและจุดแยกที่ใช้งานวิจัยนี้นำจากกระบวนการ Decomposition ในเอกสารอ้างอิง [6] โดยนำมาใช้งานในบางส่วนเพื่อให้เหมาะสมกับแนวทางที่จะใช้ในงานวิจัยนี้

กระบวนการนี้จะทำโดยการสแกนไปบนทุกจุดของภาพด้วยวินโดว์ขนาด 3 x 3 ดังรูปที่ 3.2 โดยจุดภาพที่จะถือว่าเป็นจุดปลายของโครงร่าง จะเป็นกรณีดังรูปที่ 3.6 ซึ่งจะมีผลรวมของจุดภาพข้างเคียงดังสมการ 3.1 เท่ากับ 1 ส่วนจุดแยกจะเป็นกรณีที่ผลรวมของจุดภาพข้างเคียงมากกว่า 2 ซึ่งมีกรณียกเว้นที่เราจะไม่ถือเป็นจุดแยกดังในรูปที่ 3.7 โดยที่ในกลุ่มของ A และ B จะมีจุดที่เป็น 1 เพียงจุดเดียวเท่านั้น

0	0	0
0	X	1
0	0	0

0	0	1
0	X	0
0	0	0

0	1	0
0	X	0
0	0	0

1	0	0
0	X	0
0	0	0

0	0	0
1	X	0
0	0	0

0	0	0
0	X	0
1	0	0

0	0	0
0	X	0
0	1	0

0	0	0
0	X	0
0	0	1

รูปที่ 3.6 รูปแบบของจุดปลาย

A	0	B
A	X	1
A	0	B

B	1	B
0	X	0
A	A	A

B	0	A
1	X	A
B	0	A

A	A	A
0	X	0
B	1	B

รูปที่ 3.7 กรณียกเว้นที่ไม่นับเป็นจุดแยก

ซึ่งเมื่อสแกนพบจุดปลายและจุดแยกแล้ว จะทำการเข้ารหัสเพื่อให้ทราบว่าจุดดังกล่าวเป็นจุดแบบใด โดยทำการแทนค่าตัวเลขเป็นดังนี้

-1 แทน จุดที่เป็นจุดปลาย (end point)

-9 แทน จุดที่เป็นจุดแยก (branch point)

1 แทน จุดภาพปรกติ (image point)

ตัวอย่างของการทำการเข้ารหัสนี้จะเป็นดังแสดงในรูปที่ 3.8

(ก) (ข)

รูปที่ 3.8 ตัวอย่างการเข้ารหัสจุดปลายและจุดแยก
(ก) ภาพต้นแบบที่มีแต่โครงร่าง
(ข) ภาพหลังจากทำการเข้ารหัสเรียบร้อยแล้ว

3.4 <u>การหาจุดเริ่มต้นของการตรวจสอบโครงร่าง</u>

ก่อนที่จะทำการตรวจสอบโครงร่างของภาพ จะต้องทราบเสียก่อนว่าจะเริ่มทำการตรวจสอบที่จุดใด นั้นคือจะต้องทำการค้นหาจุดเริ่มต้นเสียก่อน ในการศึกษานี้จุดที่พิจารณาใช้เป็นจุดเริ่มของการตรวจสอบโครงร่างมีอยู่ 3 กรณี คือ จุดปลายที่อยู่บนซ้ายสุด, จุดปลายที่อยู่ล่างขวาสุด, และจุดภาพที่อยู่บนซ้ายสุด การเลือกใช้จะแล้วแต่ข้อมูลภาพโดยจะทำการตรวจสอบในแต่ละกรณีตามลำดับ การที่กำหนดจุดที่จะเป็นจุดเริ่มต้นตามที่กล่าว

มานี้ทำโดยการพิจารณาจากลักษณะของตัวเลขไทย ซึ่งมีจุดปลายปรากฏอยู่ในลักษณะที่ได้กำหนดไว้ คือ กรณีแรก ได้แก่ เลข ๒, ๔, ๕, ๖, ๗, ๘, และ๙ กรณีที่สอง ได้แก่ เลข ๑ และ ๓ กรณีสุดท้าย ได้แก่ เลข ๐

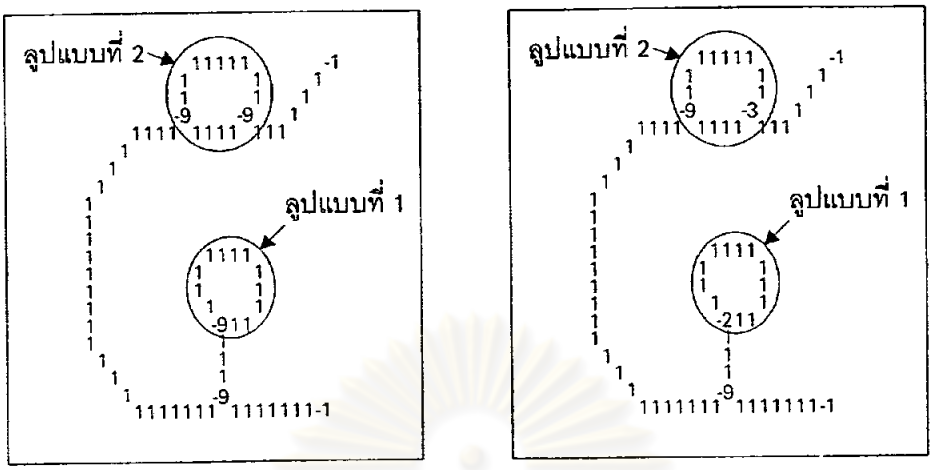
กระบวนการในการค้นหาจุดเริ่มต้นจะทำโดยการแบ่งพื้นที่ของภาพออกเป็น 2 ส่วน คือ ส่วนครึ่งบนของภาพ และส่วนครึ่งล่างของภาพ การสแกนหาจะเริ่มที่ส่วนครึ่งบนของภาพก่อนโดยสแกนในทิศทางดังรูปที่ 3.9(ก) เพื่อหาจุดปลายที่อยู่บนซ้ายสุดซึ่งถ้าพบก็จะใช้จุดนั้นเป็นจุดเริ่มต้น แต่ถ้าหลังจากสแกนเสร็จแล้วไม่ปรากฏมีจุดปลายในส่วนครึ่งบนของภาพ ก็จะทำการสแกนต่อในส่วนครึ่งล่างของภาพโดยมีทิศทางการสแกนดังรูปที่ 3.9(ข) เพื่อหาจุดปลายที่อยู่ขวาล่างสุดและใช้จุดนั้นเป็นจุดเริ่มต้น ส่วนกรณีที่ภาพนั้นไม่มีจุดปลายปรากฏอยู่ก็จะทำการสแกนทั้งภาพในทิศทางดังรูปที่ 3.9(ก) เพื่อหาจุดภาพที่อยู่บนซ้ายสุดและใช้เป็นจุดเริ่มต้นแทน

(ก) (ข)

รูปที่ 3.9 ทิศทางในการสแกน

3.5 <u>การตรวจสอบหาลูป</u> (loop)

รูปแบบของลูป (loop) ที่ปรากฏอยู่ภายหลังจากการทำให้บางจะมีอยู่ 2 ลักษณะดังตัวอย่างเลข ๕ ในรูปที่ 3.10(ก) ลูปแบบที่ 1 จะเป็นลูปที่มีจุดแยกปรากฏอยู่ลูปเพียงจุดเดียว ลูปแบบที่ 2 จะมีจุดแยกปรากฏอยู่บนลูป 2 จุด การตรวจสอบหาลูปทั้งสองแบบดังกล่าวจะทำโดยการตรวจสอบไปตามโครงร่างในแต่ละช่วงของลายเส้น ในกรณีลูปแบบที่ 1 จะปรากฏเป็นช่วงลายเส้นที่มีจุดเริ่มต้นและลงท้ายเป็นจุดเดียวกัน ส่วนกรณีลูปแบบที่ 2 จะ



(ก) (ข)

รูปที่ 3.10 ตัวอย่างผลการตรวจสอบหาลูป
(ก) แสดงรูปแบบของลูปที่ปรากฏบนตัวเลข ๕
(ข) ภายหลังจากทำการค้นหาลูปเรียบร้อยแล้ว

ปรากฏมี 2 ช่วงลายเส้นที่มีจุดเริ่มต้นและลงท้ายเป็นจุดเดียวกัน เมื่อทำการตรวจสอบพบกรณีดังกล่าวจะทำการทำเครื่องหมายที่จุดแยกเป็นดังนี้

กรณีลูปแบบที่ 1	แทนค่าที่จุดแยกด้วยค่า	-2
กรณีลูปแบบที่ 2	แทนค่าที่จุดแยกที่พบก่อนด้วย	-3
	แทนค่าที่จุดแยกที่พบหลังด้วย	-9

ดังแสดงเป็นตัวอย่างในรูปที่ 3.10(ข)

3.6 <u>การดึงเอา primitive ออกมาจากภาพ</u>

กระบวนการนี้จะประกอบด้วย 3 ขั้นตอนย่อย คือ การแยกส่วน, การแทน Primitive, และการลดจำนวน primitive ในขั้นตอนแรกจะเป็นขั้นตอนที่ทำการแยกส่วนโครงร่างของภาพออกเป็นส่วนย่อย ๆ แล้วจัดเก็บแต่ละส่วนไว้ในบัฟเฟอร์ (Buffer) โดยที่แต่ละส่วนย่อยจะมีการเชื่อมต่อกันในลักษณะของ Tree ซึ่งเป็นดังไวยากรณ์ที่เลือกใช้ การแยกโครงร่างออกเป็นส่วนย่อยที่ใช้ในงานวิจัยนี้ใช้แนวทางที่นำเสนอในเอกสารอ้างอิง[11] มาดัดแปลงเพื่อให้เหมาะสมกับชุด primitive ที่เลือกใช้ โดยมีกระบวนการดังนี้ คือ จะทำการตรวจสอบที่ละช่วงของลายเส้น โดยเริ่มจากจุดเริ่มต้นที่หาได้จากกระบวนการในหัวข้อ 3.4 จากนั้นทำการตรวจสอบที่จุดเริ่มของช่วงนั้นว่ามีเครื่องหมายแสดงว่าเป็นลูป (ดูในหัวข้อ 3.5) ปรากฏ

อยู่หรือไม่ ถ้าไม่มีก็จะทำการแบ่งช่วงของลายเส้นนั้นออกเป็นส่วนย่อยที่มีความยาวประมาณ 5 จุดภาพ แล้วเก็บไว้ในบัฟเฟอร์ตามโครงสร้างที่เชื่อมต่อกัน แต่ถ้าจุดเริ่มต้นของช่วงนั้นมีเครื่องหมายแสดงว่าเป็นลูป ก็จะทำการตรวจสอบหาตำแหน่งและขนาดของลูปแล้วจึงจัดเก็บลงในบัฟเฟอร์ ทำเช่นนี้จนครบทุกช่วงของลายเส้น

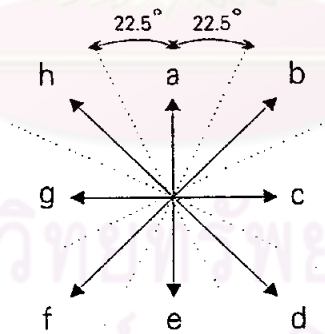
จากนั้นจะเป็นขั้นตอนของการแทน primitive ให้กับแต่ละส่วนย่อยที่หามาได้ โดยแต่ละส่วนย่อยที่ไม่ใช่ลูปจะทำการคำนวณหาทิศทางของเวคเตอร์จากจุดเริ่มของส่วนย่อยไปยังจุดปลาย ซึ่งคำนวณได้โดยใช้สมการที่ 3.2

$$\text{direction} = \arctan \{ (y_2 - y_1) / (x_2 - x_1) \} \qquad (3.2)$$

โดย y_1 และ x_1 แทนตำแหน่งในแนวแกน y และ x ตามลำดับของจุดเริ่มต้นของส่วนย่อย

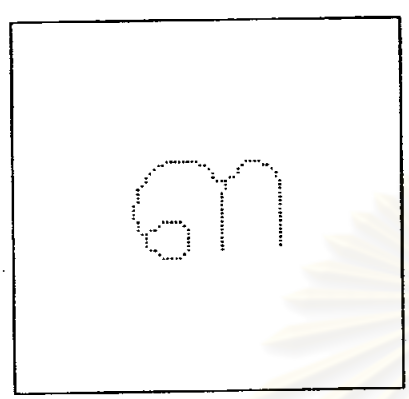
y_2 และ x_2 แทนตำแหน่งในแนวแกน y และ x ตามลำดับของจุดปลายของส่วนย่อย

นำค่า direction ที่ได้ไปเทียบหา Freeman vector ที่สอดคล้องกัน โดยเกณฑ์การตัดสินจะแบ่งตามเส้นประที่แสดงในรูปที่ 3.11 สำหรับส่วนย่อยที่มีลักษณะเป็นลูปก็จะทำการแทน

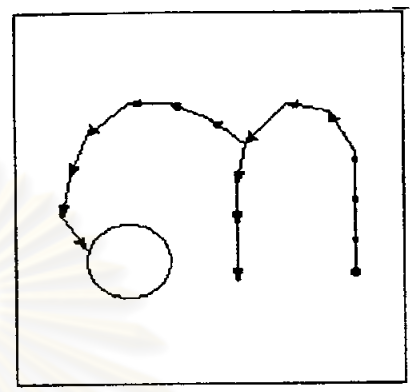


รูปที่ 3.11 เกณฑ์ในการตัดสินหา Freeman vector

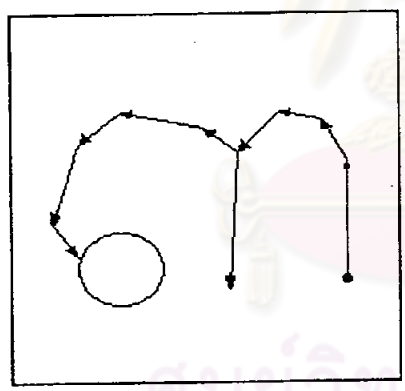
ด้วย primitive ที่ใช้แทนลูปได้เลย ส่วนในขั้นตอนสุดท้ายของกระบวนการนี้จะทำการลดจำนวน primitive ที่ซ้ำกันให้เหลือเพียงตัวเดียว ทำโดยตรวจสอบไปบน primitive ทุกตัวแล้วใช้หลักในการพิจารณาดังนี้ คือ ถ้า primitive ตัวที่เชื่อมต่ออยู่ก่อนหน้าตัวที่กำลังพิจารณาไม่มีการแตกแยกสาขาออกไปและมี primitive เหมือนกับ primitive ของตัวที่กำลังพิจารณาอยู่



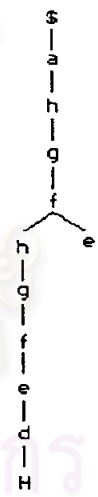
(ก)



(ข)



(ค)



(ง)

รูปที่ 3.12 ตัวอย่างผลของขั้นตอนการดึง primitive ออกมาจากภาพ

(ก) ตัวอักษรที่มีแต่โครงร่าง

(ข) ทำการแยกเป็นส่วนย่อย

(ค) หลังจากยุบ primitive ที่ซ้ำออก

(ง) การเชื่อมต่อของ primitive ในรูปแบบ Tree

ก็จะทำการตัดตัวที่กำลังพิจารณาอยู่นั้นทิ้งไป ในขั้นตอนนี้ทำเพื่อลดการซ้ำซ้อนของ primitive ให้เหลือแต่ตัวที่จำเป็นจริง ๆ เท่านั้น ตัวอย่างการทำตามกระบวนการนี้จะเป็นดังรูปที่ 3.12

สำหรับบัฟเฟอร์ที่ใช้ในการจัดเก็บ primitive แต่ละตัวในโครงสร้างการเชื่อมต่อแบบ tree [17] จะมีลักษณะดังรูปที่ 3.13 โดยแต่ละส่วนมีหน้าที่ดังนี้

ส่วน father ใช้สำหรับชี้ไปยัง primitive ตัวที่เชื่อมต่ออยู่ก่อนหน้านี้

ส่วน prim ใช้ในการเก็บตัวอักษรที่ใช้แทน primitive

ส่วน son ใช้สำหรับชี้ไปยัง primitive ตัวที่เชื่อมต่อถัดจากตัวนี้

ส่วน next ใช้สำหรับชี้ไปยัง primitive ตัวที่เป็นสาขาแยกออกมาจากตัว primitive ก่อนหน้านี้ที่อยู่ถัดไป

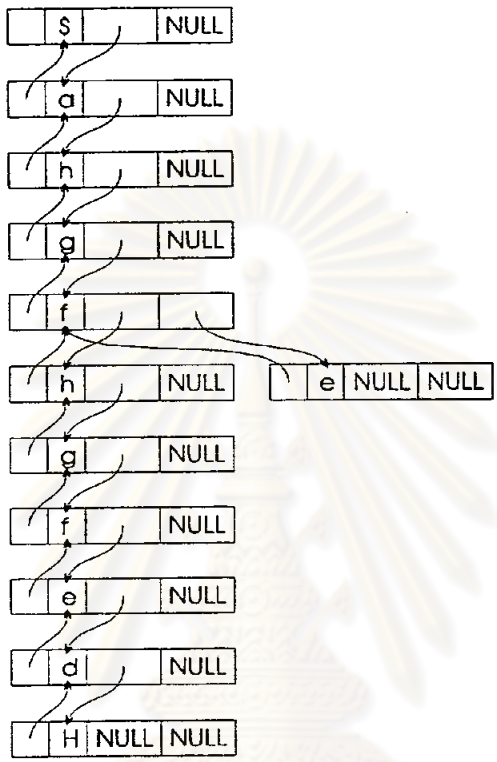
รูปที่ 3.13 รูปแบบของบัฟเฟอร์ที่ใช้ในการเก็บ primitive

ถ้ากรณีที่ primitive ที่จัดเก็บนั้นไม่มีตัวที่เชื่อมต่อถัดลงไป หรือ ไม่มีตัวสาขาที่อยู่ถัดไปที่ต้องชี้ก็จะกำหนดให้ส่วน son หรือ next นั้นมีค่าเป็น NULL (แทนด้วยเลข 0) ซึ่งจากตัวอย่างในรูปที่ 3.12(ง) การจัดเก็บในบัฟเฟอร์จะเป็นดังแสดงในรูปที่ 3.14

3.7 การหาลำดับ Postfix

ก่อนที่จะทำการคำนวณหา distance ระหว่าง 2 trees จะต้องทำการหาลำดับ postfix ของแต่ละ tree เสียก่อน เนื่องจากในการคำนวณหา distance ที่จะกล่าวถึงในหัวข้อต่อไปนี้ จะมีการกำหนดเงื่อนไขที่เกี่ยวข้องกับลำดับ postfix ในแต่ละ Tree ซึ่ง postfix จะกำหนดเป็นดังนี้ [14] คือ

ให้ a และ b เป็น 2 โนดใน D กำหนดตัวกระทำการ $h_\alpha : D_\alpha \rightarrow N$ ที่ใช้ลำดับชุดลำดับของโนดใน D_α ในแบบเครื่องหมาย postfix



(ก)

SEQUENCE NUMBER	:	0	1	2	3	4	5	6	7	8	9	10	11
PRIMITIVE	:	$	a	h	g	f	h	g	f	e	d	e	H
FATHER	:	0	0	1	2	3	4	5	6	7	8	4	9
SON	:	1	2	3	4	5	6	7	8	9	11	0	0
NEXT	:	0	0	0	0	0	10	0	0	0	0	0	0

(ข)

รูปที่ 3.14 การจัดเก็บในบัฟเฟอร์

(ก) โครงสร้างที่แทนด้วยบัฟเฟอร์ของรูปที่ 3.12(ง)

(ข) จากรูป (ก) เมื่อเรียงตามลำดับบัฟเฟอร์ที่ใช้เก็บ

ตัวอย่างที่ 3.1 ให้ D_α = { 0 , 1 , 2 , 1.1 , 1.2 } ดังนั้น

$h_\alpha(1.1) = 1$

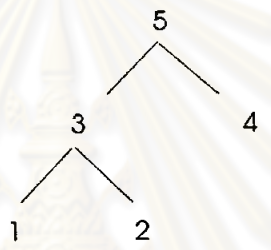
$h_\alpha(1.2) = 2$

$h_\alpha(1) = 3$

$h_\alpha(2) = 4$

$h_\alpha(0) = 5$

ลำดับ postfix ของ D_α จะกำหนดเป็น



วิธีการหาลำดับ postfix ในงานนี้จะใช้อัลกอริทึมที่ปรากฏในเอกสารอ้างอิง [17] ซึ่งจะสอดคล้องกับรูปแบบการจัดเก็บ primitive ที่กล่าวไว้ในหัวข้อ 3.6 โดยมีอัลกอริทึมเป็นดังแสดงใน Algorithm 3.2 ซึ่งเขียนเป็นลักษณะของรูทีนย่อย (subroutine) ในภาษา C ที่ใช้วิธีเรียกซ้ำตัวเองไปเรื่อย ๆ

Algorithm 3.2. Postfix order

```
        p is the pointer of primitive buffer
intrav (p)
{
        if ( p != NULL) {
                intrav (p->son);
                write p to postfix buffer;
                intrav (p->next);
```

ผลของการทำตามอัลกอริทึมนี้กับข้อมูลตัวอย่างเลข ๓ ในรูปที่ 3.13 จะได้ลำดับ postfix เป็นดังรูปที่ 3.15

SEQUENCE NUMBER	:	0	1	2	3	4	5	6	7	8	9	10	11
PRIMITIVE	:	$	a	h	g	f	h	g	f	e	d	e	H
FATHER	:	0	0	1	2	3	4	5	6	7	8	4	9
SON	:	1	2	3	4	5	6	7	8	9	11	0	0
NEXT	:	0	0	0	0	0	10	0	0	0	0	0	0
POSTFIX ORDER	:	11	9	8	7	6	5	10	4	3	2	1	0

รูปที่ 3.15 ผลการหาลำดับ postfix กับข้อมูลตัวอย่างในหัวข้อ 3.6

3.8 การคำนวณหา distance ระหว่าง 2 trees [24]

ให้ α และ β เป็น 2 trees ใด ๆ กำหนดการแปลง (transformation) T เป็นการแม็พจากโดเมน D_β ไปเป็น D_α โดยการแปลง T จะประกอบด้วย 3 รูปแบบต่อไปนี้ คือ ε , ϕ , และ σ. โดยที่

$$\varepsilon(b) \rightarrow \lambda, p$$

$$\phi(\lambda) \rightarrow a, q$$

$$\sigma(b) \rightarrow a, r$$

โดยที่ $b \in D_\beta$, $a \in D_\alpha$, λ เป็นสัญลักษณ์ null และ p , q , และ r เป็น costs ที่สอดคล้องกับ ε , ϕ , and σ ตามลำดับ

การแปลง ε หรือ ϕ จะเป็นผลให้เกิด การตัดออก (deletion) หรือ การแทรกเข้า (insertion) ตามลำดับ ส่วนการแปลง σ จะเป็นผลให้เกิดการแทนที่ (substitution) ในกรณีโนด b และ a ไม่เหมือนกัน [$\beta(b) \neq \alpha(a)$] ถ้า p และ q เป็น cost ที่สอดคล้องกับการแปลง ε และ ϕ ตามลำดับ ดังนั้น p และ q จะเป็น cost ของการตัดออกและการแทรกเข้าตามลำดับ เราจะให้ $r>0$ ถ้าการแปลง σ ก่อให้เกิดการแทนและให้ $r = 0$ ในกรณีนอกจากนี้

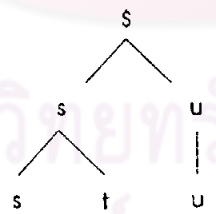
ให้ T^{-1} แทนการแปลงย้อนกลับของ T T^{-1} จะเป็นการแม็พจากโดเมน D_α ไปเป็น D_β ซึ่งจะได้ว่า $\varepsilon^{-1} = \phi$ และ $\sigma^{-1} = \sigma$ distance ระหว่าง 2 trees α และ β เขียนด้วย $d(\alpha,\beta)$ คือ cost ที่น้อยที่สุดที่จำเป็นเพื่อที่จะได้ α จาก β โดยใช้ชุดของการแปลง T ที่สอดคล้องกับเงื่อนไขต่อไปนี้

1. ถ้า $a < b$ ใน D_β ดังนั้น $T(a) < T(b)$ และถ้า a และ b ไม่สามารถเปรียบเทียบกันได้ ดังนั้น T(a) และ T(b) ก็จะไม่สามารถเปรียบเทียบกันได้
2. ถ้า $a = b$ ใน D_β ดังนั้น $T(a) = T(b)$ และถ้า $a \neq b$ ดังนั้น $T(a) \neq T(b)$
3. ถ้า $h_\beta(a) < h_\beta(b)$ ดังนั้น $h_\alpha(T(a)) < h_\alpha(T(b))$ โดยมีข้อแม้ว่า T(a) หรือ T(b) ต้องไม่เป็นสัญลักษณ์ null

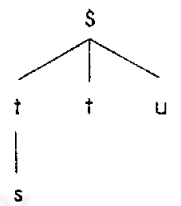
ถ้าสมมุติว่า cost ที่สอดคล้องกับการแปลง ε และ ϕ เป็น 1 และ cost ที่สอดคล้องกับการแปลง σ เป็น 1 ถ้ามีการแทนที่เกิดขึ้น และเป็น 0 ในกรณีนอกจากนี้ ดังนั้น distance จะกลายเป็นจำนวนการตัดออก , การแทรกเข้า และ การแทนที่ที่น้อยที่สุดที่จำเป็นเพื่อที่จะได้อันหนึ่งจากอีกอันหนึ่ง

ตัวอย่างที่ 3.2 กำหนด tree 2 trees เป็น α และ β

β :



α :



พิจารณาเซ็ตของการแปลง T_1 :

T_1 : $D_\beta \rightarrow D_\alpha$

$\sigma(0) \rightarrow 0 , 0$

$\varepsilon(1) \rightarrow \lambda , 1$ (การตัดออก)

$\sigma(1.1) \rightarrow 1.1 , 0$

$\sigma(1.2) \rightarrow 2 , 0$

$\phi(\lambda) \rightarrow 1, 1$ (การแทรกเข้า)

$\sigma(2) \rightarrow 3, 1$ (การแทนที่)

$\varepsilon(2.1) \rightarrow \lambda, 1$ (การตัดออก)

ให้ cost ของการตัดออก , การแทรกเข้า และ การแทนที่ เป็น 1 ทั้งหมด ดังนั้น $d(\alpha,\beta)$ เป็นชุดของ cost ที่น้อยที่สุดที่สอดคล้องกับ เงื่อนไข (1) - (3)

h_β :	(1)		(2)	(3)	(4)	(5)	(6)
D_β :	1.1	λ	1.2	1	2.1	2	0
	↓	↓	↓	↓	↓	↓	↓
h_α :	(1)	(2)	(3)			(4)	(5)
$D_\alpha = T_1(D_\beta)$:	1.1	1	2	λ	λ	3	0

ดังนั้น $d(\alpha,\beta) = 4$

พิจารณาเซ็ตอีกอันหนึ่ง

T_2 : D_β	$\rightarrow$	D_α	
$\sigma(0)$	$\rightarrow$	0 , 0	
$\sigma(1)$	$\rightarrow$	1.1 , 0	
$\sigma(1.1)$	$\rightarrow$	3 , 0	
$\sigma(1.2)$	$\rightarrow$	1 , 0	
$\sigma(2)$	$\rightarrow$	2 , 1	(การแทนที่)
$\varepsilon(2.1)$	$\rightarrow$	λ , 1	(การตัดออก)

แม้ว่า cost ของ T_2 จะน้อยกว่าแต่มันไม่สอดคล้องตามเงื่อนไขข้อที่ 1 คือ

$T_2(1) = 1.1$ และ $T_2(1.2) = 1$

โดยที่ $T_2(1) > T_2(1.2)$ แต่ $1 < 1.2$

สำหรับอัลกอริทึมในการคำนวณหา distance ที่ใช้จะเป็นดังใน Algorithm 3.3 ซึ่งอัลกอริทึมนี้จะใช้วิธี dynamic programming ในการคำนวณหาแต่ละค่าใน distance metrix เขียนแทนด้วย D(i,j) โดยที่ i และ j เป็นลำดับ postfix ของ tree β และ tree α ตามลำดับ กำหนดให้ $a = h_\alpha^{-1}(j)$ และ $b = h_\beta^{-1}(i)$ โดยที่ a และ b เป็นโนดใน tree α และ tree β ซึ่ง $h_\alpha(a) = j$, $h_\beta(b) = i$ นั้นคือว่า D(i,j) จะเป็น cost ที่น้อยที่สุดที่จำเป็นในการหา tree ย่อย α/a จาก β/b ที่เป็นไปตามเงื่อนไข 1, 2, และ 3

Algorithm 3.3 Distance Between Two Trees.

Input : Two trees α and β. The number of nodes in are $N(\alpha) = m$, $N(\beta) = n$.

Output: $d(\alpha,\beta)$, the distance between α and β.

Methods: Let D be an (m+1) x (n+1) array.

Step 1. D(0,0) = 0.

Step 2. Do j = 1, m.

(2.1) $D(0,j) = N(\alpha/a) \times q$, where $a = h_\alpha^{-1}(j)$

Return.

Step 3. Do i = 1, n.

(3.1) $D(i,0) = N(\beta/b) \times p$, where $b = h_\beta^{-1}(i)$

Return.

Step 4. Do i = 1, n.

(4.1) Do j = 1, m, where $a = h_\alpha^{-1}(j)$ and $b = h_\beta^{-1}(i)$

(4.1.1) E(0,0) = 0.

(4.1.2) Do l = 1, t, where t = r(a).

(4.1.2.1) $E(0,l) = E(0,l-1) + N(\alpha/a_l) \times q$.

Return.

(4.1.3) E(0,t+1) = E(0,t) + q.

(4.1.4) Do k = 1, s, where s = r(b).

(4.1.4.1) $E(k,0) = E(k-1,0) + N(\beta/b_k) \times p$.

Return.

(4.1.5) E(s+1,0) = E(s,0) + p.

(4.1.6) Do k = 1, s.

(4.1.6.1) Do l = 1, t.

$$(4.1.6.1.1)\quad E(k,l) = \min \begin{cases} E(k-1,l) + N(\beta / b \cdot k) \times p \\ E(k,l-1) + N(\alpha / a \cdot l) \times q \\ E(k-1,l-1) + D(u,v) \end{cases}$$

where $u = h_\beta(b \cdot k)$ and $v = h_\alpha(a \cdot l)$.

Return.

Return.

(4.1.7) Do l = 1, t.

$$(4.1.7.1)\quad E(s+1,l) = \min \begin{cases} E(s+1,l-1) + N(\alpha / a \cdot l) \times q \\ E(s,l) + p \\ E(0,l-1) + D(i,v) \end{cases}$$

where $v = h_\alpha(a \cdot l)$

Return.

(4.1.8) Do k = 1, s.

$$(4.1.8.1)\quad E(k,t+1) = \min \begin{cases} E(k-1,t+1) + N(\beta / b \cdot k) \times p \\ E(k,s) + q \\ E(k-1,0) + D(u,j) \end{cases}$$

where $u = h_\beta(b \cdot k)$

Return.

$$(4.1.9)\quad D(i,j) = \min \begin{cases} E(s,t+1) + p \\ E(s+1,t) + q \\ E(s,t) + r \end{cases}$$

where $r > 0$ if $\alpha(a) \neq \beta(b)$; $r = 0$, otherwise.

Return.

Return.

Step 5. $d(\alpha,\beta) = D(n,m)$.

Step 6. End.

จากกระบวนการต่าง ๆ ที่ได้กล่าวถึงในบทนี้ จะนำมาใช้พัฒนาเป็นโปรแกรมสำหรับใช้ในการรู้จำตัวเลขไทยในแบบตัวพิมพ์ในงานวิจัยนี้ ซึ่งข้อมูลภาพตัวเลขที่ต้องการรู้จำจะต้องผ่านกระบวนการที่กล่าวมาตามลำดับที่ได้แสดงไว้ในรูปที่ 3.1 จากนั้นจึงนำผลที่ได้สุดท้ายคือ ค่า distance ที่คำนวณได้เทียบกับต้นแบบแต่ละตัวที่จัดเก็บไว้มาทำการตัดสินใจจำแนกว่าภาพข้อมูลตัวเลขนั้นเป็นตัวเลขตัวใด ซึ่งในการทดสอบผลการรู้จำด้วยกระบวนการที่กล่าวมานี้ และ ผลการทดสอบที่ได้จะกล่าวถึงต่อไปในบทที่ 4