UNIVERSITY OF CALIFORNIA

Los Angeles

3D City Modeler with Fuzzy Multiple Layers Perceptron:

Application of Soft Computing in Computer Aided Architectural Design Systems

A dissertation submitted in partial satisfaction of the
requirements for the degree Doctor of Philosophy
in Architecture

by

Yoshihiro Kobayashi

2001
The dissertation of Yoshihiro Kobayashi is approved.

University of California, Los Angeles

2001
# TABLE OF CONTENTS

**TABLE OF CONTENTS**

iii

**LIST OF FIGURES**

viii

**LIST OF TABLES**

xii

**VITA AND PUBLICATION**

xiii

**ABSTRACT OF THE DISSERTATION**

xiv

**CHAPTER I: INTRODUCTION**

1

1.1 Research Areas

1.2 Background in Architecture

   1.2.1 Introduction

   1.2.2 History

   1.2.3 Problem

   1.2.4 Knowledge-based Design Systems

   1.2.5 3D Computer City Models

1.3 Motivation

   1.3.1 Problems between Engineer and Designing

   1.3.2 3+2 Stage in Building Extraction

   1.3.3 Two Limitations

   1.3.4 Suggested Solution

   1.3.5 Methodology

1.4 Contribution

   1.4.1 Three Kinds of Knowledge

   1.4.2 Contribution to Architecture

   1.4.3 Contribution of Neural Networks in Knowledge Building

1.5 Summary

**CHAPTER II: RELATED STUDIES**

17

2.1 CAD (Computer Aided Design)

   2.1.1 Definition
2.1.2 Original CAD System 18
2.1.3 History of CAD System 18

2.2 CAAD (Computer Aided Architectural Design) 21
  2.2.1 Definition 21
  2.2.2 History of CAAD System 21

2.3 AI (Artificial Intelligence) 23
  2.3.1 Definition 23
  2.3.2 History 24
  2.3.3 Applications in Architecture 26
  2.3.4 Theories in Analyzing and Generating Design 26
  2.3.5 Computer Systems 27

2.4 Expert Systems 29
  2.4.1 Original Expert System 29
  2.4.2 History of Expert System 30
  2.4.3 Detail of Expert System 30

2.5 Neural Networks 32
  2.5.1 Definition 32
  2.5.2 Background 32
  2.5.3 Original Neural Network Model 33
  2.5.4 Learning in Neural Networks 35
  2.5.5 Feed-forward Models 36
  2.5.6 Backpropagation 37
    2.5.6.1 Backpropagation Features 39
    2.5.6.2 Nomenclature 40
    2.5.6.3 Backpropagation Learning Algorithm 41
  2.5.7 Self-Organizing Map 44
    2.5.7.1 SOM Features 46
    2.5.7.2 SOM Algorithm 46
  2.5.8 Hopfield Networks 47
    2.5.8.1 Hopfield Network Features 50
    2.5.8.2 Hopfield Network Algorithm 50
  2.5.9 Summary of Neural Networks 52

2.6 Image Processing (Machine Vision) 53
  2.6.1 Definition 53
  2.6.2 Detecting Objects 54
    2.6.2.1 Hough Transform & General Hough Transform 55
    2.6.2.2 Contour Based Algorithms 58
    2.6.2.3 Neural Network Based Algorithms 60
  2.6.3 Other Problems 62
CHAPTER III: LITERATURE REVIEW

3.1 Building Extraction
   3.1.1 History of Building Extraction 64
   3.1.2 Research Projects in Building Extraction 65
3.2 Soft Computing
   3.2.1 Definition 69
   3.2.2 Categories in Neuro-Fuzzy Systems 70
3.3 Fuzzy Multiple Layers Perceptron (MLP)
   3.3.1 Input Data in Fuzzy MLP 72
   3.3.2 Passing Values in Fuzzy MLP 72
   3.3.3 Output Data in Fuzzy MLP 73
   3.3.4 Weight Updating in Fuzzy MLP 74
   3.3.5 Rule Generation and Inferencing in Fuzzy MLP 75
      3.3.5.1 Input Data, Passing Values, and Output Data 76
      3.3.5.2 Querying 78
      3.3.5.3 Justification 79
3.4 Summary 80

CHAPTER IV: PROBLEM STATEMENT

4.1 Problems
   4.1.1 Complex Interfaces 81
   4.1.2 Similar Cases 82
   4.1.3 Reuse of Knowledge 83
   4.1.4 Rule Representation 84
4.2 Causes 85
4.3 Hypothesis 86
4.4 Proposal 87
4.5 Limitation 88
4.6 Summary 89

CHAPTER V: METHODOLOGY

5.1 Introduction
   5.1.1 3D Computer City Models 91
   5.1.2 Segmentation and Interpretation Stages 94
   5.1.3 Neuro-Fuzzy System 96
   5.1.4 Training and Refining Processes 97
   5.1.5 Features 99
   5.1.6 Summary 101
5.2 Inside of the System 102
5.3 Techniques for 3D Model
   5.3.1 Solid Modeling 102
   5.3.2 Texture Mapping 103
5.4 Techniques of Image Processing 104
   5.4.1 RGB Values 104
   5.4.2 Intensity/Gray Value 105
   5.4.3 Convolution 105
   5.4.4 Width and Height 106
   5.4.5 Area 106
   5.4.6 Boundary Searching 107
5.5 Techniques of Fuzzy MLP 109
   5.5.1 Memberships 109
   5.5.2 Mean Square Error and Cross-Entropy 111
   5.5.3 Momentum in Backpropagation Learning 112

CHAPTER VI: PROGRAM 113

6.1 Java 113
6.2 User's Manual 114
   6.2.1 Components 114
      6.2.1.1 Image Panel 115
      6.2.1.2 Data Panel 115
      6.2.1.3 Output Information Panel 119
   6.2.2 Pull Down Menu 120
   6.2.3 Short Tutorial 123
      Step1. Segmentation Stage 124
      Step2. Interpretation Stage 125
      Step3 Generating 3D Computer City Models 126
6.3 I/O File Formats 127
   6.3.1 Sample Data Format 127
   6.3.2 Fuzzy MLP Format 128

CHAPTER VII: CASE STUDIES 130

7.1 Simple Case 130
7.2 Practical Case 141
7.3 Experiments 149
   7.3.1 Changing Sample Number 149
   7.3.2 Changing Neuron Number of Hidden Layer 154
   7.3.3 Applying the Trained Weights to Similar Cases 160
7.4 Summary and System Evaluation 166
CHAPTER VIII: CONCLUSION

8.1 Accomplishments 170
8.2 Questions 171
8.3 Improvement in the Future 173
  8.3.1 3D Computer City Models 174
  8.3.2 Fuzzy Rules 174
  8.3.3 Collaboration with other Knowledge 175
  8.3.4 Training by Communications in Natural Languages 176
  8.3.5 Application in the Internet 176
8.4 Conclusion 177

BIBLIOGRAPHY 180
<table>
<thead>
<tr>
<th>Number</th>
<th>Figure Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-4-1</td>
<td>Relations between Architecture and Computer Science</td>
<td>14</td>
</tr>
<tr>
<td>2-5-1</td>
<td>A simple McCulloch-Pitts neuron Y</td>
<td>34</td>
</tr>
<tr>
<td>2-5-2</td>
<td>A McCulloch-Pitts neuron to perform the logic (Not A or B)</td>
<td>35</td>
</tr>
<tr>
<td>2-5-3</td>
<td>A Hebb Net</td>
<td>36</td>
</tr>
<tr>
<td>2-5-4</td>
<td>Network Structure of Backpropagation</td>
<td>38</td>
</tr>
<tr>
<td>2-5-5</td>
<td>Binary Sigmoid Function, range (0,1)</td>
<td>40</td>
</tr>
<tr>
<td>2-5-6</td>
<td>Network Structure of Self-Organizing Map</td>
<td>45</td>
</tr>
<tr>
<td>2-5-7</td>
<td>Neighborhoods for rectangle grid in SOM</td>
<td>45</td>
</tr>
<tr>
<td>2-5-8</td>
<td>Recurrent Neural Network Structure</td>
<td>48</td>
</tr>
<tr>
<td>2-5-9</td>
<td>Algorithm of Hough Transform</td>
<td>56</td>
</tr>
<tr>
<td>2-5-10</td>
<td>Algorithm of General Hough Transform</td>
<td>57</td>
</tr>
<tr>
<td>2-5-11</td>
<td>Case of interruption</td>
<td>58</td>
</tr>
<tr>
<td>2-5-12</td>
<td>Chain Codes</td>
<td>59</td>
</tr>
<tr>
<td>2-5-13</td>
<td>Process of Detecting Rectangle</td>
<td>60</td>
</tr>
<tr>
<td>2-5-14</td>
<td>Data Structure of Triangle with Texture in MEM</td>
<td>62</td>
</tr>
<tr>
<td>2-5-15</td>
<td>Detecting Process in MEM</td>
<td>62</td>
</tr>
<tr>
<td>3-3-1</td>
<td>Diagram of classification phase of fuzzy MLP</td>
<td>74</td>
</tr>
<tr>
<td>3-3-2</td>
<td>Example rule generation scheme</td>
<td>75</td>
</tr>
<tr>
<td>5-1-1</td>
<td>Satellite Images</td>
<td>91</td>
</tr>
<tr>
<td>Section</td>
<td>Title</td>
<td>Page</td>
</tr>
<tr>
<td>--------------</td>
<td>----------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>5-1-2</td>
<td>Output Images in Building Extraction</td>
<td>92</td>
</tr>
<tr>
<td>5-1-3</td>
<td>3D Computer City Models</td>
<td>93</td>
</tr>
<tr>
<td>5-1-4</td>
<td>Output Images in Segmentation and Interpretation Stages</td>
<td>94</td>
</tr>
<tr>
<td>5-1-5</td>
<td>Diagram of the Process in Segmentation and Interpretation Stages</td>
<td>95</td>
</tr>
<tr>
<td>5-1-6</td>
<td>Network Structure of Fuzzy Multiple Layers Perceptron</td>
<td>97</td>
</tr>
<tr>
<td>5-1-7</td>
<td>Training Process in Neuro-Fuzzy Systems</td>
<td>98</td>
</tr>
<tr>
<td>5-1-8</td>
<td>Refining Process in Neuro-Fuzzy Systems</td>
<td>99</td>
</tr>
<tr>
<td>5-3-1</td>
<td>Winged-edge Data Structure</td>
<td>103</td>
</tr>
<tr>
<td>5-3-2</td>
<td>Multiple Texture Mapping</td>
<td>104</td>
</tr>
<tr>
<td>5-4-1</td>
<td>Process of Convolution</td>
<td>105</td>
</tr>
<tr>
<td>5-4-2</td>
<td>Width and Height of Segmented Region</td>
<td>106</td>
</tr>
<tr>
<td>5-4-3</td>
<td>16 Conditions of 2x2 Pixels and their Searching Directions</td>
<td>107</td>
</tr>
<tr>
<td>6-2-1</td>
<td>Main Program Interface</td>
<td>114</td>
</tr>
<tr>
<td>6-2-2</td>
<td>Interface of Fuzzy MLP Program</td>
<td>120</td>
</tr>
<tr>
<td>6-2-3</td>
<td>Commands in Pull Down Menu</td>
<td>123</td>
</tr>
<tr>
<td>7-1-1</td>
<td>Input Image for Simple Case</td>
<td>131</td>
</tr>
<tr>
<td>7-1-2</td>
<td>Weights Condition of Fuzzy MLP for Segmentation</td>
<td>134</td>
</tr>
<tr>
<td>7-1-3</td>
<td>Categorized Image</td>
<td>134</td>
</tr>
<tr>
<td>7-1-4</td>
<td>Segmented Image with &quot;House&quot; Label</td>
<td>136</td>
</tr>
<tr>
<td>7-1-5</td>
<td>Weights Condition of Fuzzy MLP for Interpretation</td>
<td>138</td>
</tr>
<tr>
<td>7-1-6</td>
<td>Output Image Applied Morphological Functions</td>
<td>139</td>
</tr>
<tr>
<td>7-1-7</td>
<td>3D Computer City Model of Simple Case</td>
<td>139</td>
</tr>
</tbody>
</table>
7-2-1 Input Image for Case 1  
7-2-2 Weights Condition of Fuzzy MLP  
7-2-3 Categorized Image in Practical Case  
7-2-4 Segmented Image with "House" Label  
7-2-5 Weight Condition of Fuzzy MLP for Interpretation  
7-2-6 Output Image Applied Morphological Functions  
7-2-7 3D Computer City Model  
7-2-8 3D Computer City Model with Texture Mapping  
7-3-1 Results of 50 Samples  
7-3-2 Results of 100 Samples  
7-3-3 Results of 200 Samples  
7-3-4 Results of 500 Samples  
7-3-5 Results of 1000 Samples  
7-3-6 Results of 2000 Samples  
7-3-7 Results of 5000 Samples  
7-3-8 Results of 500 Samples with 1(10) Hidden Layer  
7-3-9 Results of 500 Samples with 1(3) Hidden Layer  
7-3-10 Results of 500 Samples with 1(5) Hidden Layer  
7-3-11 Results of 500 Samples with 1(15) Hidden Layer  
7-3-12 Results of 500 Samples with 2(3,3) Hidden Layers  
7-3-13 Results of 500 Samples with 2(5,5) Hidden Layers  
7-3-14 Results of 500 Samples with 2(10,10) Hidden Layers
| 7-3-15 | Image of 5000 Samples | 159 |
| 7-3-16 | Results of Initial Satellite Image | 161 |
| 7-3-17 | Results of Similar Image without Training | 162 |
| 7-3-18 | Results of Similar Image without Training, Missing Some Buildings | 163 |
| 7-3-19 | Results of Similar Image after Refining Process | 163 |
| 7-3-20 | Comparison in the case of Fig. 7-3-16 | 164 |
| 7-3-21 | Comparison in the case of Fig. 7-3-17 | 165 |
| 7-3-22 | Comparison in the case of Fig. 7-3-19 | 165 |
# LIST OF TABLES

<table>
<thead>
<tr>
<th></th>
<th>Table Number</th>
<th>Table Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-4-1</td>
<td>Three Kinds of Knowledge</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>7-1-1</td>
<td>Sample Data for Segmentation</td>
<td>132</td>
<td></td>
</tr>
<tr>
<td>7-1-2</td>
<td>Parameters for Memberships in Segmentation</td>
<td>132</td>
<td></td>
</tr>
<tr>
<td>7-1-3</td>
<td>Memberships for Inputs and Outputs in Segmentation</td>
<td>132</td>
<td></td>
</tr>
<tr>
<td>7-1-4</td>
<td>Generated Fuzzy Rules in Segmentation</td>
<td>135</td>
<td></td>
</tr>
<tr>
<td>7-1-5</td>
<td>Translated Fuzzy Rules</td>
<td>135</td>
<td></td>
</tr>
<tr>
<td>7-1-6</td>
<td>Sample Data for Interpretation</td>
<td>136</td>
<td></td>
</tr>
<tr>
<td>7-1-7</td>
<td>Parameters for Memberships in Interpretation</td>
<td>137</td>
<td></td>
</tr>
<tr>
<td>7-1-8</td>
<td>Memberships for Inputs and Outputs in Interpretation</td>
<td>137</td>
<td></td>
</tr>
<tr>
<td>7-1-9</td>
<td>Generated Fuzzy Rules in Interpretation</td>
<td>140</td>
<td></td>
</tr>
<tr>
<td>7-1-10</td>
<td>Translated Fuzzy Rules</td>
<td>140</td>
<td></td>
</tr>
<tr>
<td>7-2-1</td>
<td>Sample Information for Membership Values</td>
<td>142</td>
<td></td>
</tr>
<tr>
<td>7-2-2</td>
<td>Generated Fuzzy Rules in Segmentation</td>
<td>145</td>
<td></td>
</tr>
<tr>
<td>7-2-3</td>
<td>Parameters for Memberships in Interpretation</td>
<td>146</td>
<td></td>
</tr>
<tr>
<td>7-2-4</td>
<td>Generated Fuzzy Rules in Interpretation</td>
<td>149</td>
<td></td>
</tr>
<tr>
<td>7-3-1</td>
<td>Changes by the Number of Samples</td>
<td>152</td>
<td></td>
</tr>
<tr>
<td>7-3-2</td>
<td>Changes by the Number of Hidden Layers and Neurons</td>
<td>156</td>
<td></td>
</tr>
<tr>
<td>7-3-3</td>
<td>Results of Comparison</td>
<td>166</td>
<td></td>
</tr>
</tbody>
</table>
VITA

July 08, 1968 Born in Kanagawa, JAPAN

1992 B.Arch
    Waseda University (Watanabe, H. Lab.)
    Tokyo, Japan

1994 M.Arch (Sanae Prize)
    Waseda University (Watanabe, H. Lab.)
    Tokyo, Japan

PUBLICATIONS


xiii
ABSTRACT OF THE DISSERTATION

3D City Modeler with Fuzzy Multiple Layers Perceptron:
Application of Soft Computing in Computer Aided Architectural Design Systems

by

Yoshihiro Kobayashi

Doctor of Philosophy in Architecture
University of California, Los Angeles, 2001
Professor Kostas Terzidis, Chair

A computer aided design (CAD) system that can store the design knowledge of users is proposed. Specifically, a computer system for generating 3D city models from satellite images is formulated, implemented and tested. Techniques from neural networks, fuzzy systems, image processing, pattern recognition, and machine learning constitute the methodological foundation of the system. The flexibility and usability of system are evaluated.
This research is about a computer design system to create 3D objects using machine learning technologies. Fuzzy multiple layers perceptron (Pal and Mitra, 1992), one of fuzzy neural network models, is integrated in the system as a main engine for learning. This approach to implement the technologies of fuzzy neural networks in CAD/CAAD systems is very novel, and there have been no CAD/CAAD applications with such an implementation. The contribution of this research was to reduce the labor and time to create 3D models, which is one of the most tedious and slow-down processes in CAD/CAAD systems. Specifically, it is implemented by enabling the computer system to avoid routine works, adapt to similar cases, and support users’ subjective ideas. Technology proposed and tested in this dissertation may ultimately make it possible for CAD/CAAD systems to store design knowledge in architecture and urban planning. One of the main strength of the system is that it can change the relation and logic between the user's requirements and the system's behaviors dynamically and interactively. In other words, it stores the user's approach to extract visual information, and enables the other users to reuse the stored knowledge. In this research, the system is applied in creating 3D computer city models from satellite images as a test case. The system extracts the regions of what the user recognizes as buildings/houses in the images, and generates 3D models by giving the attributes of the buildings' height automatically. The usability and flexibility of the system are evaluated in case studies and evaluation method is reported at the end of
each example/case-study.

I-1. Related Areas

This research is related to two research fields. One is the study of image understanding (IU) in image processing, especially the study of monocular building extraction, which is an application of IU. The other research field is the study of machine learning (ML) in AI, especially the study of how to make computer systems gain architectural and urban design knowledge. Therefore, the direction of this research can be defined as integrating these two technologies to generate 3D computer city models for architectural and urban design.

I-2. Background in Architecture

This section presents the current state of the computer environments in architecture and urban planning, focusing on the difficulty in realizing knowledge-based design systems, and the importance of 3D computer city models.

Introduction

For the last 5 years, computers have become standard tools for accomplishing practical architectural, urban, and landscape design works. For example, design presentation using computer graphics, construction management using computer database, and automatic checking of architectural standards using computer expert systems have been applied in the real world. (Cornick, 1996)
History

Before the 1960's, computers were only used as calculators in architectural fields. Specifically, computers were used in the fields of technology, planning theory, and civil engineering because complete conventional mathematical equations were already being used in these fields, and it was easy to establish the mathematical functions in computers. In the 1970's, the success of Computer Aided Design (CAD) and Computer Graphics (CG) led to the development of Computer Aided Architectural Design (CAAD) systems. However, since the development of CAAD systems was so costly that most architectural firms could not afford them, systems were generally used only for research. In the 1980's, as a result of success in the research of expert systems, the idea of expert systems was transplanted to CAAD systems. Consequently, even people who did not have special knowledge of architecture started to solve various problems using the expert system in computer programs. Also, specialists in architecture became more proficient in explaining their thoughts logically by using expert systems. In the 1990's, the price of computers dropped so drastically that more designers, planners, and engineers started to use computers and applied software to their work as just another common design tool.

Problem

However, many designers and planners are not satisfied with what the current computers provide, especially at the beginning stage of their design process. Therefore, they still use pencil and paper to get their design inspiration. The main reason why they are still using these primitive tools is that current computer systems do not allow designers to carry out a
lot of trial and error. In other words, the current computer systems cannot deal with humanistic problems such as image recognition, voice recognition, and flexible thought. Today neural networks are considered to be the most effective methodology to solve these problems. Therefore, it is necessary to study the application of neural networks in architecture most vigorously.

Knowledge-based Design Systems

The field of planning studies includes urban, landscape, and architectural planning. These all have various kinds of computer applications. Most researchers and practitioners in these fields of study use expert systems called "Knowledge Based Design Systems," which solve problems by using object-oriented databases and "knowledge," a collection of \(<\text{If } \sim, \text{ then } \sim>\) rules.

However, Knowledge Based Design Systems have not been practically utilized in planning because they do not have the most useful expert knowledge, which can derive results from given statistical information. As a result, many planners and designers are still unable to use the expert knowledge potential of computers. Instead, they have been spending a lot of effort and time on storing GIS (Geographical Information System) information and statistical data in computers in order to evaluate it manually for their research.

There are two different factors in decision making when planning and designing: a mathematical factor and a humanistic factor. The mathematical factor makes it possible to create functional models by applying the techniques of mathematics or physics. The
humanistic factor deals with human recognition, feeling, and experience. Although the mathematical factor can be implemented in expert systems, it is very difficult to implement the humanistic factor into the systems because it varies depending on the situation and available information. In fact, there needs to be a computer system that can simultaneously manipulate the mathematical and the humanistic factors in designing and planning.

**3D Computer City Models**

It has become important for designers and planners in architecture and urban planning to create computer city models in order to visualize, simulate, and estimate their design plans (Liggett and Jepson, 1995), (Liggett and Jepson, 1995-2), (Alkhoven, 1991). These are the typical examples of applying 3D city models for visualization of virtual worlds. On the other hand, the technology of image processing has also become practical due to the improvement in computer speed and price. Even though the study of building extraction from aerial images has been researched and developed since the 1980's, it has had few applications in practical architectural and urban designing tasks. It is therefore important to implement more practical and applicable systems for architecture and urban planning, since designers and planners currently invest much time and labor in creating 3D computer city models manually. In short, they need computer systems where designers and planners easily store and use their expert knowledge to create 3D computer city models.
I-3. Motivation

Problems between Engineering and Designing

The motivation of this research comes from conflicts between engineering and designing. From the engineer's point of view, the purpose of monocular (non-perspective) building extraction in image understanding is to create optimal systems that can detect city objects such as buildings, streets and houses accurately. Their focus is on the accuracy of the system. From the designer's point of view, the desire is to create computer city models that fulfill their need to manipulate the detected city objects. Their purpose is to give required attributes to the objects such as height, texture and style, because they use the city models for estimation, visualization, and simulation of their design and plans. Therefore, their desires for computer systems focus on the usability of the systems. These different research goals of engineers and designers are due to the fact that designers and planners do not use computer systems that the computer scientists develop. Instead they use 3D commercial software applications to create computer city models manually (Snyder and Jepson, 1999). This method, however, is very time consuming. It is therefore necessary to develop computer systems that fulfill the demands of designers and planners. Moreover, it is indispensable to allow them to control the inside logic of the system in order to adapt the system to their demands.

3+2 Stage in Building Extraction

Most of the research on image processing has three stages. The first stage is to detect geometrical attributes in aerial and satellite images such as edges, lines, contours, and areas.
The second stage is to select required objects by reducing noise and errors comprising unnecessary information in images. The third stage is to optimize the selected objects by specific rules from expert systems. The researchers concentrate on how computer systems should be arranged and what kinds of algorithms are better in order to develop more accurate systems in the above three stages. However, they do not pay enough attention to the usability of system for the end users such as designers and planners. In fact, two more stages seem to be necessary for making the information useful for designers and planners. One is to give attributes to the detected information that the user requires for manipulating detected objects, such as linguistic labels and degrees. Another stage is to give the systems learning ability in order to make it possible for designers and planners to use the systems and manipulate objects easily. This learning ability also makes it possible to customize the system for their own purpose. These two stages should be considered after the first three stages or be integrated into them.

**Two Limitations**

There are at least two limitations in implementing a CAD/CAAD system where the technology of image processing is integrated user-friendly enough to satisfy the expectation of designers and planners. One is the limitation of technology in image processing. It is impossible to create systems that can detect objects in aerial images as accurately as humans even with the current computer science technology. Most of the systems always have varying degrees of errors in image understanding. Another limitation is related to the representation of design knowledge, -how to store design ideas and
concepts in computer systems. It is caused by the difficulty in representing human knowledge in computer systems. Especially, the knowledge in architecture and urban planning is considered as a combination of mathematical knowledge and human knowledge. The former is objective, but the latter is subjective. They have controversial characteristics each other. Therefore, it is necessary to find the way to implement both of them at the same time in computers. This research will attempt to address both of these issues.

**Suggested Solutions**

In order to solve the problems of how to address the above issues, this research will implement the following strategy. First, the main cause of errors in image understanding is due to the robustness of the systems. In other words, since most of the systems apply the technology of expert systems with a set of static <If ~, then ~> rules, the errors will increase in the case of exceptions. For example, some applications of monocular building extraction use the shadows of buildings to detect buildings and their height (Shufelt, 1993). However, this method cannot be applied when buildings do not cast shadows. Therefore, in order to decrease the inflexibility of systems, it is necessary to implement the ability to adapt flexibly according to the situation. In short, the systems should have the ability to change their methodology when the existing rules do not work well. Second, in order to solve the problem of design knowledge representation, this research assumes that design knowledge is the relation between the given situation and the designer's subsequent actions. In other words, the design knowledge is the kind of functions the designer chooses to
manipulate the given images or the detected objects from the images. The design knowledge should also have the ability to adapt flexibly, because the relations vary from designer to designer.

**Methodology**

In order to implement computer systems that can adapt flexibly for various situations and different users (designers and planners), the technology of neuro-fuzzy systems is applied in this research. The study of neuro-fuzzy systems is a form of soft computing in AI. Neuro-fuzzy systems have both the features of neural network systems and fuzzy systems. The main feature of neural network systems is that they have the ability of learning and representing complex relations between inputs and outputs by using weight matrices. The main feature of fuzzy systems is that they can represent ambiguous rules with linguistic terms. Therefore, it is considered more appropriate to use the neuro-fuzzy systems in order to solve the problems of errors in image understanding and representing design knowledge. Neuro-fuzzy systems will be discussed further in Chapter III.

**I-4. Contribution**

This section shows the importance of this research as a coherent body of knowledge. First, three kinds of knowledge are defined. Secondly, the contribution to the morphological analysis in architecture is explained. Finally, the reasons why neural networks can contribute to this area, and some considerable examples are shown.
**Three Kinds of Knowledge**

"Knowledge" can be classified into three types: mathematical knowledge, encyclopedic knowledge, and human knowledge (Russel and Norvig, 1995).

Mathematical knowledge consists of a set of axioms. It is absolutely correct and there is no exception other than truth. When a new theory is proposed among mathematical knowledge, it is examined mathematically. If the theory has mistakes or is the same as the previous knowledge, it becomes worthless. On the other hand, if the theory is correct and novel, it becomes a theorem. Mathematical knowledge is described with universal mathematical symbols such as "+, "-," and numbers, and it is classified by mathematical logic. Mathematical explanation is logical and deterministic so that it is possible to determine whether it is true or false. In mathematical knowledge, a prediction cannot be said to be true or false until it has either been proven or disproved.

Encyclopedic knowledge is an accumulation of facts such as historical truth and the data from scientific experiments. It is represented as a set of "<X happened in the situation Y>," where X is a result, and Y is a situation, and described as a set of data and information. Usually, statistical methods are used for encyclopedic knowledge. The most probable relation between a situation and a result becomes a theory in this knowledge; in short, mistakes and errors are permitted as far as the theory does not conflict with other theories. Probabilistic and analytical representations are used for encyclopedic explanation while past facts and theories are referenced to make the explanation. Mathematical knowledge is
often applied to the explanation of encyclopedic knowledge in the natural sciences because the results can be predicted easily if the relations between situations and results can be described with mathematical functions. When it is difficult to describe the relation between situations and results mathematically, probability theories are used to predict the results, such as "X will happen with probability of XX% according to the past data."

Human knowledge is a knowledge that is not possible to describe with mathematical symbols nor to describe as \(<X \text{ happened in the situation } Y>\). It is beyond description but understandable for humans. People use human knowledge to recognize objects, sounds, language, feelings, ideas, and so on, and they acquire the knowledge from their experiences and senses. Although languages are often used to describe human knowledge, people can understand it ambiguously and roughly if they have had the similar experiences. If the backgrounds of their experiences are completely different, they cannot understand each other. Human knowledge is too complex to explain or classify by mathematical or statistical methods. Only human brains can understand human knowledge.

Mathematical knowledge that is absolutely correct is applicable to the other two kinds of knowledge. For over 300 years, since the age of Newton, science has tried to represent the relation between situations and results by using mathematical functions because the results become computable if the relations can be represented mathematically. Analytical and statistical functions have been developed and applied to various research areas. These methods are no longer thought to be able to lead to the solution of human knowledge. For instance, current science still cannot explain how the human brain recognizes objects. Even
though neural networks are considered to be the most effective methodology to explain human brain systems, the practical research has only just begun. In short, research on human knowledge like that which is discussed above is still at the beginning stage.

**Contribution to Architecture**

In architecture, especially for morphological analysis: mathematical knowledge and encyclopedic knowledge have been considered to be more important. As a result, the current computer systems were developed to explain theories as mathematical models or statistical data. However, they have not yet developed enough to explain the design concepts, ideas, and various tendencies of designers and architects, which are related to human knowledge and are the most important in morphological analysis in architecture. Therefore, researchers still need to study how human knowledge should be applied to morphological analysis. If computer systems can understand and manipulate it, people can reuse the other designers' concepts and theories and store them in computers. In other words, implementing human knowledge in image recognition is the most desirable technology of today, and it will be an important contribution to architectural analysis if it becomes possible.

The following table summarizes the three kinds of knowledge, mathematical, encyclopedic, and human. Each of them is represented as the expert system, statistical methods, and neural networks.
## Contribution of Neural Networks in Knowledge Building

The practical contribution of human knowledge to morphological analysis is to apply neural network techniques to CAD/CAAD systems as shown by the relations in Fig. 1-4-1. Neural networks are considered to be the most effective methodology to manipulate human knowledge. The following is one proposal of a method for creating a neural network system that can be applied to morphological analysis. It includes three steps: (1) 3D/2D models are generated automatically from images by machine vision techniques, (2) users manipulate the 3D/2D data by using natural language interfaces, and (3) the designers' concepts, ideas, and theories are stocked by learning.
In the past, traditional morphological analysis in architecture was applied manually (Unwin, 1997) (Ching, 1996). Recently computers have been used to create 3D architectural models and drawings as a common tool, but they have not developed enough to help architectural research. Several expert systems have been applied to morphological analysis in architecture (Terzidis, 1994), but their abilities are very limited because the systems cannot manipulate human knowledge such as design concepts, ideas, and theories. Therefore, the combination of expert systems and neural networks for morphological analysis in architecture is a new approach. To be concrete, neural networks can contribute to the application of morphological analysis and recognition in architecture. The followings are examples.
- Auto image recognition of architectural drawings:
  A technology of image processing to understand the meaning of architectural drawings automatically and analyze the architecture from the drawings.

- Auto generation of 3D architectural models from images:
  A computer graphics technology to generate complex 3D objects automatically by combination of systematic functions.

- Realization of generative theories in architecture:
  A technology to implement architectural generative theories by neural network models in computers.

- Storing the designer's concepts and reusing them:
  A technology to represent designers' concept, ideas, and tendencies as a weight matrix of neural networks.

- The collaboration among people in various fields:
  A technology to collaborate design works among designers, planners, and clients by communicating with their own specific domain terms and knowledge in one computer system.

- Real time simulation with domain specific languages:
  A technology to simulate 3D models using natural language interfaces.

- Criticize architectural designs:
  A technology to create artificial architectural critics in computers.

Researchers have proposed many kinds of applications for morphological analysis by using neural networks. However, it is still difficult to create computer systems that can understand human knowledge by only applying simple neural network technologies. Therefore, it is necessary to develop more advanced neural network theories and models for architectural studies.

I-5. Summary

This section introduced the abstract, the main two related study fields, the current situation
of design and computation in architecture, the motivation, and the contribution of this research. This research relates to the study fields of image understanding and machine learning. Currently computers have not been used as expert knowledge but only as a visualizing tool in designing and planning. For example, designers and planners spend a lot of time and labor in creating 3D computer city models manually. The main reason why it is difficult to apply the technologies of AI to architecture comes from the different research directions between engineering and designing. The engineering researchers focus on the accuracy of systems, but designers focus on the usability of systems. Specifically, in monocular building extraction, two more stages -labeling and learning stages- are necessary in addition to the three traditional stages. However, the limitation of current technology and the difficulty in representing human knowledge are obstacles in implementing more user-friendly systems of monocular building extraction. One solution for implementing them is to apply the ability of adapting to various situations, and a neuro-fuzzy system is one of the most suitable methods for implementing the adapting ability. This research contributes to the morphological analysis in architecture and helps to establish human knowledge in designing. Moreover, if this approach can store human knowledge in morphological analysis, it is possible to apply the approach to many problems in architecture and urban planning, such as drawing recognition, generating objects theories, collaboration, and so on.
Chapter II

Related Studies

This chapter explains several study fields applied in this research and their techniques. In fact, this research relates to broad study fields and many applied techniques such as CAD, CG, neural networks, fuzzy systems, and image processing. Here six sections, -computer aided design (CAD), computer aided architectural design (CAAD), artificial intelligence (AI), expert systems, neural networks, and machine vision- are explained. In each section definition, history, and some techniques are represented.

II-1. CAD (Computer Aided Design)

Definition

In order to grant various clients' demands promptly, it is important for designers to economize their time and labor in designing and planning. As a consequence, design and planning companies have been trying to find how to minimize their planning, designing, and manufacturing cycles. For example, the idea of “concurrent designing,” in which planning, designing, and producing processes are made concurrently rather than step by step, has been recently proposed. Computer Aided Design (CAD) is a computer system developed to economize designers' time and labor and has been the most effective tool to accomplish this for many designers.
Original CAD System

The CAD system was created for applying the technology of Computer Graphics (CG) to the design process. The original CAD project was started at Massachusetts Institute of Technology (MIT) in 1959. This project tried to create computer systems in which images and objects could be manipulated. In the beginning of the 1960’s, I. E. Sutherland completed the first graphical computer system called SKETCHPAD (Sutherland, 1963). Although the SKETCHPAD was developed on old and slow computers, it had many of the ideas that the latest CAD systems have today. In 1967, IBM developed a CAD system, called "DAC-1," for General Motors to design automobiles, and Lockheed developed its own commercial CAD system called "CADAM."

History of CAD Systems

The original CAD systems such as SKETCHPAD had a huge server computer and several graphic terminals. There were many communication problems between the server computer and terminals. The greatest problem was the massive workload assigned to the server computer. To solve the problems, microcomputers replaced the terminal devices, and each microcomputer started calculating graphics individually. Recently, architectural firms and construction companies have adopted this advanced server CAD system to reduce the workload of server computers and make the speed of graphic calculations faster.

At the same time that the server CAD systems appeared, a different approach to CAD systems was developed using a personal computer equipped with special turnkeys instead
of having a huge server computer with several terminal devices. In 1968, Applicon Corp. released such a personal CAD system called AGS, and Computer Vision Corp. released CADDS in the following year. In the mid-1970's, personal computer CAD systems became one of the mainstream CAD systems because their installation was much easier than that of the server CAD system.

In the late 1970's, Autodesk Corp. started to sell new two-dimensional CAD application software for the personal computer CAD system, called AutoCAD. Since AutoCAD provided most of the drawing functions that the server CAD systems had with a much lower price, it contributed the diffusion of the personal CAD systems throughout the world. In the 1980’s, people started to use the personal computer CAD system with faster graphic accelerators so that it became possible to display three-dimensional objects in real time. Today, personal computer CAD systems are used in architectural firms much more than the server CAD systems.

As CAD systems spread, the standardization of Computer Graphics (CG) and CAD systems became necessary. In 1984, International Standards Organization (ISO) created GKS (Graphical Kernel System) for two-dimensional objects in computer graphics (GKS, 1984). GKS-3D was established as an official standard for three-dimensional objects in 1987 (GKS-3D, 1987). In 1989, PHIGS (Programmer's Hierarchical Interactive Graphics Standard) that dealt with object data hierarchically became a standard (Shuey, 1987). In 1991, a Windows standard named PEX (PHIGS Extensions to X) was created by the
combination of PHIGS+, an extended version of PHIGS that had new standardization of shading and lighting in computer graphics (PHIGS+ 1988), and X Window System, an operational system developed in MIT (Rost 1988).

Software for CAD systems made it possible to create complex three-dimensional objects in computers. When creating CAD software, the process of modeling and rendering needed to be considered. A technology for three-dimensional shapes, called Computer Aided Geometric Design (CAGD), played an important role in advancing modeling systems. The examples of CAGD are solid modeling, free-form curves, and surfaces. Solid modeling is a technique in which three-dimensional data is treated as a solid instead of a set of points and lines. Free-form curves and surfaces are techniques to create complex curves and surfaces which include B-spline surfaces (Gordon, 1974), Coons surface (Coons, 1964), Bezier surface (Bezier, 1966), and NURBS (Forrest, 1980). Rendering is a technology to visualize the modeled three-dimensional objects realistically on a computer screen. Many rendering techniques, such as scan-line rendering (Wylie, 1967), ray-tracing (Appel, 1968), radiosity rendering (Goral, 1984), and texture mapping (Catmull, 1974), have been developing for decades. Today it is possible to create quite realistic three-dimensional images by using these rendering techniques.

The hardware and software of CAD systems have always developed symbiotically. In the beginning stage of the CAD systems, the hardware researchers concentrated on developing devices such as plotters, mice, and keyboards while the software researchers were studying
how to manipulate shapes and images on the screen using those devices. Today, the researchers of hardware are trying to create graphical accelerators to reduce the rendering time while the researchers of software are focusing on how to make the rendering algorithms faster.

II-2. CAAD

Definition

Computer-aided architectural design (CAAD) is the application of computer aided design (CAD) in architecture. CAAD is used to plan and design buildings and houses in computers. In the beginning, the CAAD technology did not progress as much as CAD technology did in engineering. There were several reasons. First, many architects had strong feelings against using computers. They thought architecture was not engineering but an artistic process. Second, most architects had little knowledge of computers. They did not expect computers to develop so rapidly. Also, the CAAD systems were very expensive. As compared with automobile or aerospace firms, architectural firms usually had much smaller budgets for enhancing their own computer systems.

History of CAAD Systems

Researchers started discussing the potential of CAAD systems in the 1960’s (Dawson, 1961), (Eberhard, 1962), (Sounder, 1963). In 1964, Christopher Alexander published a book, "Notes on the Synthesize of Form," which proposed a systematic methodology to generate architectural plans and designs. It predicted the potential of computer-based
methods in architectural designing and fascinated many CAAD researchers. Several initial CAAD systems were experimentally developed in this period. For example, Souder and Clark implemented COPLANNER in 1964 (Souder, 1964), and Negroponte created URBAN5 in 1970 (Negroponte 1970). During the 1960's, the Department of Civil Engineering at MIT and Pennsylvania State University started extensive research on the engineering applications of architecture. In the 1970's, the institute of Physical Planning at Carnegie-Mellon University started research on building description and space planning issues, and the Architectural Machine Group at MIT started research on artificial intelligence techniques. These universities became the leaders in developing CAAD systems in the U.S.

After I. E. Sutherland announced his SKETCHPAD, researchers in various countries throughout the world started researching CAAD systems. In Britain, CAAD system development was based on large-scale computer-aided design systems. In 1964, a program using the IBIS industrialized component system for housing design was demonstrated at the Industrialized Building Systems and Components Exhibition in London. In the same year, William M. Newman at the Cambridge University Mathematics Laboratory implemented an interactive graphic program into building designs (Newman, 1966). In Australia, much research based on knowledge-based design systems was carried out at the Department of Architectural Science at the University of Sydney. John Gero, a professor at the University of Sydney, is one of the world’s leaders in CAD and CAAD systems that apply the technology of AI. In Asia, especially in Japan, Osaka University and the IBM
Japan Scientific Center were the center of CAAD research in the late 1960's, and Tsuyoshi Sasada, a professor of the Department of Environment Engineering at Osaka University, made considerable research on the environment of collaboration for CAAD systems.

By the middle of the 1970's, as application of CAD technology had become common in many fields of engineering and manufacturing throughout the world, so did the application of CAAD systems in architecture. The study of CAAD appeared in courses of many universities, and many books about CAAD systems were published while many conferences began to be held around the world. For example, "Computer-Aided Architectural Design (1977)" by William J. Mitchell, "Computer Aids to Design and Architecture (1975)" by Negroponte, and "Computer Applications in Architecture (1977)" by Gero are still virtual bibles in this field. CAAD future, ACADIA, ECAADE, AI in Design, and CAADRIA, are the example conferences about CAAD.

The current research on CAAD is focused mainly on computer systems using knowledge-based reasoning, which are called "expert systems." The success of expert systems in various other fields induced its application towards CAAD systems.

II-3. AI

Definition

Artificial intelligence (AI) is one big study area in computer science. In one standard textbook (Russel and Nervig, 1995), AI is defined as a study of computer systems that
think and act like humans: employ rationality. In other words, the purpose of AI is to implement the processes of human minds such as reasoning, perceiving, learning, and decision making in computers, and to apply the processes to several research fields. The textbook also categories AI into four sub fields, which are planning and problem-solving, knowledge and reasoning, machine learning, and communicating, perceiving and acting.

1) Planning and problem-solving is a series of research in how to make computers solve various problems by planning the optimum strategy. Search algorithms, heuristic functions, and general problem solvers are research topics in this field.

2) Knowledge and reasoning is a series of research topics on how to solve specific domain problems logically with knowledge and reasoning. Propositional-logic, first-order-logic, ontology, theorem provers, probability models, expert systems, and knowledge building are research topics in this field.

3) Machine learning is the study of how to implements computer systems that learn. Neural networks, fuzzy theory, and genetic algorithms are research topics in this field.

4) Communicating, perceiving, and acting is a series of research topics on how to make computers behave like humans. Image processing, machine vision, natural language processing, artificial life, and robotics are research topics in this field.

**History**

In 1950, Alan M. Turing, who was an English logician and created the first computer in history, published a paper, "Computing Machinery and Intelligence" (Turing, 1950). He
posed the question of whether machines could think or not and proposed a famous "Turing Test". After a few years, several programs and systems for Turing's proposal were developed. For example, G. P. Dineen and O. G. Selfridge showed computer systems that had the ability to recognize visual patterns (Dineen, 1955), (Selfridge, 1955). Allen Newell published a paper on computer systems that could play chess games (Newell, 1955). In 1956, the term "artificial intelligence" was coined formally when Newell and Herbert Simon (Newell, 1957) reported on the system that could prove some mathematical theorems of Russel and Whitehead's *Principia Mathematica*, a famous textbook about logic.

In the 1960's, the main research in AI was on planning and problem-solving, and the systems using neural networks were considered effective for establishing the ability to perceive. In the 1970's, knowledge and reasoning in computers were major topics, and many expert systems for specific domains were developed. At the same time, interest in neural networks declined because of proven limitation in their learning ability. In the 1980's, AI became an industry because of success in expert systems, and neural networks came back again as a hot topic in AI due to the new learning algorithm, backpropagation. The recent approach to AI is focused on agent-oriented systems that can solve not specific problems but more general problems by combining various AI techniques developed during these two decades.
Applications in Architecture

It is difficult to define the original application of AI in architecture, since there are many systematic theories to analyze or generate architectural plans and shapes in history. Those approaches which develop design principles allowing other people (and machines) to design new models are close to the concepts of AI, especially planning and problem-solving. This section introduces some historical theories in analyzing architecture and urban planning. Several computer applications in architecture and urban planning are presented.

Theories in analyzing and generating design

The morphological analysis in architecture was originally demonstrated by Vitruvius in the book of "The Ten books on Architecture" published in the 1st century. Vitruvius categorized the ancient temples by the geometrical attributes such as symmetry and proportion (Vitruvius, 1960).

L. B. Alberti applied the idea of harmony into architectural design, and developed a theory to design buildings with a series of sacred ratios (Alberti, 1755). His theory is often applied in modern architecture such as the Le Corbusier's modular scheme.

Leonardo da Vinci proposed a systematic generative system in "Elements et Theorie de l'Arcchitecture" (Da Vinci, 1894), and R. Banhan demonstrated the 24 different plans by using Vinci's system (Banham, 1960).
Shape grammar developed by G. Stiny is the application of BNF (Backus-Naur from) in architecture. The syntax rules in architecture are described as a set of spatial relations. Applying his shape grammar, he demonstrated the ability as an analyzing method in architecture by generating several houses that were similar to the houses designed by F. L. Wright (Stiny, 1980).

Architectonics is also the study of analyzing architectural design with modern mathematical theories of group, number, and symmetry (March, 1998).

Pattern Language is a systematic architectural and urban design theory developed by C. Alexander. He considered the design process as a combination of patterns in his famous book, "Pattern Language". (Alexander, 1972).

Currently, many researchers in architecture and urban planning have applied the AI techniques such as fractal geometry (Erickson and Jones, 1997), artificial life (Bonabeau, 1994), cellular automaton (Clarke, Hoppen, and Gaydos, 1997), etc in analyzing designs and plans, generating new designs, and simulating their theories.

Computer Systems

EDM (Engineering Date Model) and EDM-2 introduced by C. M. Eastman proposed a new way of developing databases for engineering design. In the system, the parts of a building are described with the geometrical features and restricted relations (Eastman,
OOMAS (Object-oriented Architectural Modeler and Simulator) developed by Watanabe (Watanabe, 1994) is also an intelligent CAAD system with the technology of object-oriented programming. In the system, a building is considered as a set of objects, and each object changes its attributes automatically corresponding to the given environment. There are many other studies of how to describe the relations of architectural objects in computers (Ullman, 1988), (Turner, 1988), (Bjork, 1992). As a result, the international standards such as STEP (Standard for the Exchange of Product data model) and IFC (Industrial Foundation Classes) have been developed.

SEED (Software Environment for the Early Phases of Building Design) developed by Flemming (Flemming, 1994) and PRECEDENT developed by Oxman (Oxman, 1994) are computer applications using the technology of case-based reasoning, and the systems helped to design buildings in the early design stages.

There are a few research projects on CAAD systems using the technology of neural networks. R. D. Coyne and his associates applied the technique of neural networks in CAAD systems to solve problems of building attribute-form mapping, which is the relation between the forms of plan and their attributes in buildings. (Coyne and Newton, 1990; Coyne 1991). Y. L. Liu developed a computer system to demonstrate the visual phenomena in 2D designing, such as incomplete and ill-processed shape recognition, shape transformation, and emergent sub-shape recognition, by using pattern matching techniques with multiple layers neural networks. (Liu, 1993; 1994; 1996).
II-4. Expert Systems

Expert systems are computer systems using the technology of knowledge engineering, which is one methodology of artificial intelligence in computer science. Expert systems are systems that have the same ability as experts to solve problems in specific domains by storing specific knowledge and heuristics.

The Original Expert Systems

The original expert system, called DENDRAL, was developed at Stanford University in the late 1960's. It was developed to infer the structure of organic chemical compounds (Feigenbaum, 1971) and was famous as the first expert system to use production rules. More practical research on expert systems began in the middle of the 1970's. For example, MYCIN was a medical expert system that could diagnose a patient's illness in place of a doctor, (Shortliffe, 1976), and PROSPECTOR was an expert system that could search for veins, (Duda, 1979). Current research on expert systems has two directions. One is to make expert systems more efficient as an industrial technology, for such systems as medical diagnosis, VSLI design, intelligent CAD systems, and so on. The other is to create theories about the expert system that have wider applicability. Researchers are trying to develop more flexible theories that can realize agent-oriented systems in which the objects called "agents" do anything users request.
**History of Expert Systems**

In the 1950's, the main research areas of artificial intelligence (AI) were concentrating on the study of General Problem Solver (GPS) and its search algorithms. Many kinds of search algorithms were developed, including breadth-first search, depth-first search, best-first search, A* search, and mini-max algorithm. However, it was problematic because those techniques could solve only toy problems such as puzzle games and simple differential equations. More practical methodology was therefore required to realize AI. Because of this problem, the purpose of AI changed from discovering the general theory that was applicable for all kinds of problems to developing the methodology to apply specific domain knowledge for specific problems. In other words, it was recognized that both experts' knowledge and heuristics were indispensable to solve practical problems in the real world. Research on how to represent experts' knowledge and how to apply the experts' knowledge to computers became the center of AI research.

**Detail of Expert Systems**

Expert systems consist of three parts: a knowledge database, an inference engine, and a working memory. The knowledge database contains a set of specific domain facts and experts' heuristics. Knowledge is represented as the rules of \(<IF \neg, \THEN \neg>\). This is called "long term memory" from the point of view of cognitive science. The inference engine is based on mathematical logic, and first-order logic is usually applied. The working memory is part of a calculating process in which a given question is applied to the knowledge database, and the answer is driven by using the inference engine. This is called "short term
memory" in cognitive science. The following is a simple example of the process of expert systems:

The following three rules are stocked in the knowledge database
1) If A happens, then B happens.
2) If E happens, then C happens.
3) If B happens, then D happens.

And the inference engine adopts the following logic rules:
For every x, y, and z, if there are two rules, which are <if x happens, then y happens> and <if y happens, then z happens>, it is true to say that <if x happens, z happens>.
If x happens and there is a rule, which is <if x happens, then y happens>, then y happens.

In this case, when a person asks, "what happens if A happens?" then expert systems answer "B and D happen because of rule 1 and 3" by using the logic rules and knowledge database in the working memory.

There are several problems with expert systems. For example, (1) a set of <If ~, then ~> rules cannot encompass the entire extent of an expert's knowledge, (2) expert systems require a huge working memory space because they must apply the logic rules to all the matching cases of facts in the knowledge database, and (3) the systems do not have the capability to learn. Therefore, new methods such as fuzzy logic, possibility theory, and
case-based reasoning have been researched in order to solve these problems.

II-5. Neural Networks

This section introduces the study of neural networks. Here more time is spent for explaining the techniques because neural networks are the primary algorithms of this research.

II-5-1. Definition

Neural networks or artificial neural networks are computer systems that have a structure that is similar structure of human brain. They have been developed to apply the human ability of recognizing and learning to computers by imitating organic neural functions.

II-5-2. Background

In the last several decades, computers have made remarkable technological progress and solved many problems that humans could never have solved alone. It is said that computers became five times faster every 3 years, while computer prices drop by half each year. Today computers are affordable and have become indispensable for many people.

However, there are still many problems that are almost impossible to solve with recent computer science technology. For example, computers are still unable to see and recognize what is physically happening around them as well as humans do. Unlike a human being, a computer cannot drive a car by itself since it does not have the ability to detect what is
happening on the road and react to circumstances appropriately. Many scientists realized that they had ignored the human ability to recognize objects and to solve problems by using heuristics and other learning methods. They began to study the human nervous systems and how it processes information. This study of neural networks is considered to be one of the most important studies in computer science.

There are three levels of research in the study of neural networks. The first level is experimental research, where scientists analyze the structure of the brain and the nerves. These findings have been applied to biology and medicine. The second level is theoretical research, where scientists try to find the principles behind the brain's method of information processing. This research has been applied to the areas of mathematics, physics, and information science. The third is technical research, where scientists try to apply the principles of neural networks to computer systems. Such research on neural networks has already been applied in the areas of computer science and electrical science.

II-5-3. The original Neural Network Model

Neural network models consist of many artificial neurons just as a human brain has many biological neurons. Each neuron receives signals from other neurons around it, and when the sum of the signals exceeds a threshold, the neuron will activate. McCulloch and Pitts introduced the original idea of neuron models in 1943. (McCulloch and Pitts, 1943) They demonstrated that what they called a "McCulloch-Pitts neuron" could represent Boolean functions by calculating its neuron's threshold and weight sum. The McCulloch-Pitts
neuron is illustrated in Fig. 2-5-1.

\[ f(y_{in}) = \begin{cases} 
1 & \text{if } y_{in} \geq \theta \\
0 & \text{if } y_{in} < \theta 
\end{cases} \]

Fig. 2-5-1. A simple McCulloch-Pitts neuron Y

McCulloch-Pitts neurons have the following characteristics:

- The activation of a McCulloch-Pitts neuron is binary, 0 (the neuron not fired) or 1 (the neuron fired).
- The neurons are interconnected by direct weighted paths.
- A connection path is excitatory if the weight on the path is positive, otherwise it is inhibitory.
- Each neuron has a fixed threshold such that if the input signal is greater than the threshold, the neuron fires.

For example, if the input signal \{X1, X2, X3\} is \{0, 1, 1\}, the neuron Y receives the sum of the weighted elements, which is \(y_{in}\) in Fig. 2-5-1. Therefore, \(y_{in} = X1 \times 2 + X2 \times 2 + X3 \times (-1) = 1\). In this case, the neuron Y will fire, where the threshold value, \(\theta = 0\).
The most influential feature of McCulloch-Pitts neuron is that it can represent Boolean functions instead of logical representation. In other words, it is possible to represent knowledge database with neural networks instead of a set of "If ~, then ~" rules. For example, "If A, then B" is equal to (¬A ∨ B) in logic representation, and (Not A or B) can be represented as the McCulloch-Pitts neurons in Fig. 2-5-2. The ability of McCulloch-Pitts neuron to represent Boolean functions fascinated many scientists because they found the possibility of neural networks to be a new system to store the knowledge base instead of expert systems.

Fig. 2-5-2. A McCulloch-Pitts neuron to perform the logic (Not A or B)

**II-5-4. Learning in Neural Networks**

Neural networks have a learning ability that no other computer systems have. "Learning" means updating "weights" between two neurons in neural networks, also called "synapse strength" in physiological terminology. The original idea of learning in neural networks was proposed by Hebb, and his algorithm is known as the Hebb rule (Hebb, 1949). He proposed that learning occurs by modifying the weights in a way that if two interconnected neurons are both activated at the same time, the weights between those neurons increase. In a Hebb net, which is a neural network trained by using the Hebb rule (Fig. 2-5-3), the
weights are updated when the target neuron activates as $w_i^{\text{new}} = w_i^{\text{old}} + x_i y$, where $w_i$ is the weight from the $i$-th input neuron to the output neuron, $x_i$ the input value from the $i$-th neuron, and $y$ the desired value. Bipolar (1 or -1) data is usually used as a target value instead of binary data. The Hebb rule is still being applied to many neural network models.

There are two types of learning in neural networks: supervised learning and unsupervised learning. For supervised learning, a person needs to teach the computer system some expected information manually. For unsupervised learning, the system automatically learns the expected information by itself. Most neural network models use supervised learning because it is usually more accurate than unsupervised learning.

**II-5-5. Feed-forward Models**

Distinguished by their structure, neural network models are classified into two groups: a feed-forward model and recurrent model. The feed-forward model has layer structure. It
was developed and often used for pattern recognition and machine control because the model has the ability to learn the relations between inputs and outputs that correspond to the input.

The original feed-forward model is "Perceptron" developed by Frank Rosenblatt in the 1960s (Rosenblatt, 1958). Although Perceptron was admired as a novel model in artificial intelligence (AI) when it was known to the public, Minsky and Papert proved the limitation of perceptron's learning ability mathematically (Minsky, 1969). As a result, scientists' interest in neural networks declined.

In the middle of the 1980's, David Rumelhart, a psychologist at University of California in San Diego, developed a new feed-forward neural network model, called "backpropagation" (Rumelhart, 1986). Recently many improved feed-forward models based on backpropagation, such as RBF (radial basis function) model, have been developed (Moody, 1989). There have also been proposed learning algorithms other than backpropagation, such as Kohonen's self-organizing map and Learning Vector Quantization (Kohonen, 1989).

II-5-6. Backpropagation

Backpropagation is the best-known neural network model, and various applications of backpropagation have already been developed and used in many research fields. In fact, no better neural network model has yet been developed.
The basic model of backpropagation has three layers: input, hidden, and output layers. Neurons between the input and hidden layers and between the hidden and output layers are interconnected to each other, but the neurons among each layer are not connected (Fig. 2-5-4.).

The learning process of backpropagation is a simple two-step process. First, an input signal is received by each neuron in the input layer, and the output signal is transferred through the hidden layer and output layer. Then, the set of errors between the output signal and the desired target signal are calculated. In order to optimize the network, the weights of each layer are updated by using the delta rule. The expected weights are calculated reversibly from the output layer to the input layer. That back and forth exchange of information is why this network is called backpropagation.
The advantage of backpropagation is that the model has a very simple algorithm with great learning ability. However, there are some disadvantages. For example, it is difficult for this model to find the optimal network structures and understand the meaning of the hidden layer. Also, the learning process takes a lot of training.

II-5-6-1. Backpropagation Features

Backpropagation has the following features.

- It has a three-layer feed-forward structure, which are an input layer, one or more hidden layers, and an output layer.
- The input signal is generally binary.
- Sigmoid function (Fig. 2-5-5) is used as an activation function.
- It has a backpropagation learning algorithm, which is one of the supervised learning algorithms.
- It can be applied to pattern recognition.
- It has a simple algorithm and good learning ability.
- It has difficulty in finding the optimal network structure.
- It has difficulty in understanding the meaning of the hidden layer.
- It takes a long time to learn.
II-5-6-2. Nomenclature (Backpropagation)

The nomenclature in the backpropagation learning algorithm is as follows:

- $x$: Input training vector: $x = (x_1, x_2, ..., x_i, ..., x_n)$.
- $t$: Output target vector: $t = (t_1, t_2, ..., t_k, ..., t_m)$.
- $\delta_k$: Portion of error correction weight adjustment for $w_{jk}$, which is a weight between $Z_j$ and $Y_k$.
- $\delta_i$: Portion of error correction weight adjustment for $v_{ij}$, which is a weight between $X_i$ and $Z_j$.
- $\alpha$: Learning rate.
- $X_i$: Input unit of the neuron $i$. 

Fig. 2-5-5. Binary Sigmoid Function, range $(0, 1)$
(For an input unit, the signal to $X_i$ and output signal of $X_i$ are the same, namely, $x_i$.)

- $Z_j$: Hidden unit $j$.

  The net input to $Z_j$ is denoted $\text{z}_{\text{in},j}$. $\text{z}_{\text{in},j} = v_{0j} + \sum_i x_i v_{ij}$, where $v_{0j}$ is a bias on hidden unit $j$. The output signal of $Z_j$ is denoted $z_j$. $z_j = f(\text{z}_{\text{in},j})$.

- $Y_k$: Output unit $k$.

  The net input to $Y_k$ is denoted $\text{y}_{\text{in},k}$. $\text{y}_{\text{in},k} = w_{0k} + \sum_j z_j w_{jk}$, where $w_{0k}$ is a bias on output unit $k$. The output signal of $Y_k$ is denoted $y_k$. $y_k = f(\text{y}_{\text{in},k})$.

- $f$: Activation function.

  The most typical function is the binary sigmoid function, which has a range of $(0, 1)$ and is defined as $f(x) = \frac{1}{1 + \exp(-x)}$, with $f'(x) = f(x)[1 - f(x)]$.

II-5-6-3. Backpropagation Learning Algorithm

The following section shows the standard backpropagation algorithm.

**Step0.** Initialize weights, $w_{jk}$ and $v_{ij}$. (Set to small random values).

**Step1.** While stopping condition is false, do Steps 2-9.

**Step2.** For each training pair, do Steps 3-8.

**Feedforward Process:**

**Step3.** Each input unit $X_i$ receives input signal $x_i$.

**Step4.** Each hidden unit $Z_j$ sums its weighted input signals, $\text{z}_{\text{in},j} = v_{0j} + \sum_i x_i v_{ij}$. Then,
each hidden unit $Z_j$ sends its output signal, $z_j = f(z_{in_j})$, to output units.

**Step 5.** Each output unit $Y_k$ sums its weights input signals, $y_{in_k} = w_{0k} + \sum_j z_j w_{jk}$. Then, each output unit $Y_k$ computes its output signals, $y_k = f(y_{in_k})$.

**Backpropagation of error:**

**Step 6.** Each output unit $Y_k$ receives a target pattern corresponding to the input training pattern, computes its error information term, $\delta_k = (t_k - y_k)f'(y_{in_k})$, calculates its weight correction term, $\Delta w_{jk} = a\delta_k z_j$, $\Delta w_{0k} = a\delta_k$, and sends $\delta_k$ to units in the layer below.

**Step 7.** Each hidden unit $Z_j$ sums its delta inputs from the units in the layer above, $\delta_{in_j} = \sum_k \delta_k w_{jk}$, multiplies by the derivative of its activation function to calculate its error information term, $\delta_j = \delta_{in_j} f'(z_{in_j})$, and calculates its weight correction term, $\Delta v_{ij} = a\delta_j x_i$, $\Delta v_{0j} = a\delta_j$.

**Update weights and biases:**

**Step 8.** Each output unit $Y_k$ updates its weights, $w_{jk}(new) = w_{jk}(old) + \Delta w_{jk}$.

Each hidden unit $Z_j$ updates its weights, $v_{ij}(new) = v_{ij}(old) + \Delta v_{ij}$.

**Step 9.** Test stopping condition.

The strongest point of backpropagation is that $f'(y_{in_k})$ can be expressed in terms of $y_k$, 42
because $f'(x) = f(x)[1 - f(x)]$ and $y_k = f(y_{in_k})$ leads to the result as follows:

$$f'(y_{in_k}) = f(y_{in_k})(1 - f(y_{in_k})) = y_k(1 - y_k)$$

The backpropagation learning algorithm is based on the optimization technique known as gradient descent. However, as explained above, backpropagation has difficulty in finding the optimal number of neurons in the hidden layer and in understanding the meaning of the weights. Today the fuzzy theory and the genetic algorithm, which will be described below, solve those difficulties. Moreover, it takes much time and labor to train the backpropagation model. For example, some practical backpropagation network models have more than 10,000,000 neurons in the input layers. It is very hard to find the most effective training methods or to select the effective training samples. Therefore, many extended backpropagation models, which have momentum terms or growth factors, have been researched.

Zadeh originally developed fuzzy theory in the 1960's (Zadeh, 1965). He described fuzzy sets in the magazine called "Information and Control". The fuzzy set theory is completely different from the set theory of the past in which all factors are represented as belonging to a certain set or not. In the fuzzy set theory, the degree to which each factor belongs to the set is used rather than the boolean values, true and false.

Genetic Algorithm (GA), developed by J. Holland in 1975, is a computer algorithm that
imitates the process of genetic evolution that biological species are subject to. (Holland, 1975). GA is the algorithm to search better solutions when it is difficult or impossible to find the best solution. In the algorithm, each solution is described as a set of bits called “gene”. Therefore, finding better solutions is selecting the better genes in all genes in GA. There are three basic procedures in GA, selection, crossover, and mutation. Selection is the procedure to define which genes should be selected for the next generation. Crossover is the procedure to generate new genes by replacing some of bits between two genes. Mutation is the procedure to change the value of some bits in genes compulsorily.

II-5-7. Self-Organizing Map

Another feed-forward neural network model is the Kohonen self-organizing map (SOM), which is the most typical unsupervised neural network model. Basically, SOM has a two-layer structure: the feature map layer and the input layer (Fig 2-5-6). SOM is also called a topology-preserving map, meaning that neurons are clustered topologically in the feature map layer. SOM neurons consist of n-dimensional vectors. Neurons that have positively correlating vectors are located more closely on a one or two dimensional feature map layer. When the SOM network receives an input signal, the closest neuron in the feature map layer activates corresponding to the input.
The SOM learning is a winner-take-all process. First, when the network receives an input signal, an error between each neuron's weight and input signal is calculated. Then, only one winner neuron, which has the least error, activates. The winner neuron and its neighbor neurons, then, update their weights to optimize the network by using neighborhood functions. In the neighborhood function, the closer the neurons are to the winner neuron, the more changes the neurons make (Fig. 2-5-7).

Topological neighborhoods at different times as feature maps are formed. NE(t) is the set of nodes considered to be in the neighborhood of node j at time t. The neighborhood starts large and slowly decreases in size over times. In this example, 0<t1<t2.

Fig. 2-5-7. Neighborhoods for rectangle grid in SOM
II-5-7-1. SOM Features

SOM has the following features:

- SOM is a layer structure feed-forward neural network model.
- The learning is unsupervised, therefore, the training process is not necessary.
- It is easy to understand the topological relations among the input signals, because it is possible to visually observe the data mapped on a two-dimensional feature map.
- Less calculation is needed than the other clustering algorithms, because neurons are not interconnected in the feature map layer.
- The algorithm is simple because there is no differential calculation which most other neural networks have.
- This algorithm is biologically correct, which means the self-organizing process is the same as the human brain.

II-5-7-2. SOM Algorithm

**Step0.** Initialize weights, \( w_{ij} \).

Set topological neighborhood function, which is usually used as:

\[
h(j, j^*) = \exp\left(-\frac{|j - j^*|^2}{\sigma^2}\right),
\]

where \( j^* \) is the winner neuron and \( \sigma \) is the distance from the winner neuron.

Set learning rate parameter as: \( a = \eta h(j, j^*) \).

**Step1.** While stopping condition is false, do Step 2-6.

**Step2.** For each input vector \( x \), do Step 3-5.
Step3. For each j, compute \( D(j) = \sum (w_{ij} - x_i)^2 \).

Step4. Find the winner neuron \( j^* \) such that \( D(j^*) \) is a minimum.

Step5. For all units \( j \) within a specified neighborhood of \( j^* \), and for all \( i \), update the weights as: 
\[
    w_{ij}(\text{new}) = w_{ij}(\text{old}) + a[x_i - w_{ij}(\text{old})].
\]

Step6. Update the learning rate \( \eta \).

Step7. Reduce radius, \( \sigma \), of topological neighborhood at specified times.

Step8. Test stopping condition.

Kohonen's SOM has been applied to many fields that cluster data topologically, such as analytic studies. For example, SOM is often used to make language feature maps in natural language processing. Kohonen also developed the extended model of SOM called Learning Vector Quantization (LVQ). LVQ uses a similar algorithm to SOM, but it uses the supervised learning method. In LVQ, the weights are updated to make the correlation more positive if the winner neuron is correctly selected. Otherwise, the weights are updated to make the correlation negative. In contrast, SOM weight updating is automatic. The LVQ has more learning ability than SOM, and Kohonen demonstrated LVQ's ability in voice recognition of Finnish language.

II-5-8. Hopfield Networks (Recurrent Neural Network Models)

The previous section introduced the feed-forward neural network model and its algorithms. The following section describes another neural network model, recurrent networks. In the
feed-forward models, the neurons are connected between different layers and not interconnected among the layers. On the other hand, in the recurrent networks, each neuron is interconnected with all other neurons. This model is usually applied to pattern association and constrained optimization because it has an excellent ability to associate memories and solve optimum problems.

![Fig. 2-5-8. Recurrent Neural Network Structure](image)

As computers store information into memory that has rigid addresses, it is possible to read the information by specifying the address. On the other hand, in the human brain, memories are distributed on the nerve networks and are associated with hints or keywords. Recurrent neural network models are designed to modify that brain memory system. In other words, information is represented as a matrix in networks. The neural network models, which have those memory systems, are called associative models.

There are several associative neural network models: the Associatron (Nakano, 1972), the Hopfield network (Hopfield, 1982), BAM (bi-directional associative memory) (Kosko,
1988), and the Boltzmann machine (Hinton and Sejnowski, 1983). The following section explains the Hopfield network, which is the most famous recurrent and associative neural network model, and its algorithm.

John Hopfield, a Nobel prize-winning physicist, developed the Hopfield network in 1982. The network is the best known recurrent network. The network is also known as an associative memory model. It uses bi-directional connections with symmetric weights, which means $W_{ij} = W_{ji}$ ($i \neq j$, and $W_{ii} = 0$). The reason why this network became famous is that the Hopfield network can solve some difficult optimum problems such as the traveling salesman problem, the 8-queens problem, and the scheduling problem, by using energy or the Lyapunov function. Hopfield defined the following energy function.

$$E = -\frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} w_{ij} y_i y_j + \sum_{i=1}^{N} y_i \theta_i$$

When the network is not on an equilibrium state, the energy decreases. Therefore, the $E$ never increases. For example, when $y_i$ changes from 1 to 0 in the Hopfield network. The difference of $E$ is calculated as:

$$\Delta E_i = E_{y_i=0} - E_{y_i=1}$$

$$= \sum_{j=1}^{N} w_{ij} y_j - \theta_i < 0$$

(Θ See the activation function defined in Step 5 in the algorithm)

As a result, it is possible to say that the Hopfield network is on an equilibrium state when the energy is a local minimum.
The process of the Hopfield network is auto-associative. First, sample patterns are stored in an N by N matrix as energy. When an example-input signal comes to the network, the network tries to find the local minimum point of energy, which is the closest sample pattern corresponding to the input signal. This trial is repeated iteratively by the time it is converged. Finally, the state of the network represents the closest sample pattern by itself.

II-5-8-1. Hopfield Network Features

- The Hopfield network is a recurrent and associative neural network model.
- It is applied for pattern association and constrained optimization.
- It takes a short time to learn sample patterns, because this model does not have training processes.
- The number of patterns, which the network can memorize, is very small. It is at most \(0.14N\). (N is the number of neurons.)

II-5-8-2. Hopfield Network Algorithm

Step 0. Initialize weights to store patterns.

To store a set of binary patterns \(s(p), \ p=1,\ldots, P\), where

\[s(p) = (s_1(p), s_2(p), \ldots, s_i(p), \ldots, s_n(p)),\]

the weight matrix \(W = \{w_{ij}\}\) is given by

\[w_{ij} = \sum_p [2s_i(p) - 1][2s_j(p) - 1] \quad \text{for } i \neq j, \text{and } w_{ii} = 0.\]

While activation of the network is not converged, do Steps 1-7.
**Step1.** For each input vector \( x \), do Steps 2-6.

**Step2.** Set initial activation of network equal to the external input vector \( x \):
\[
y_i = x_i, \quad (i = 1, \ldots, n).
\]

**Step3.** Do Steps 4-6 for each unit \( Y_i \). (Units should be updated in random order.)

**Step4.** Compute network input: \( y_{-in_i} = x_i + \sum_j y_j w_{ij} \)

**Step5.** Determine activation (output signal): \( y_i = \begin{cases} 1 & \text{if } y_{-in_i} > \theta_i \\ y_i & \text{if } y_{-in_i} = \theta_i \quad \text{(Usually, } \theta = 0) \\ 0 & \text{if } y_{-in_i} < \theta_i \end{cases} \)

**Step6.** Broadcast the value of \( y_i \) to all other units.

**Step7.** Test for convergence.

There are several associative neural network models. In the Hopfield network, the process of association is called "autoassociative memory" in which each input vector is the same as the output vector. Another model, BAM is a neural network model of heteroassociative memory, where the network can memorize several patterns corresponding to an input signal. BAM and the Hopfield network has a similar algorithm. Although they can return the solutions in short time, they will not be the optimal solution but one of the local minimums. The Boltzman machine is the extended neural network models of the Hopfield network, and they can return the optimal solution by the annealing technique. In short, the Boltzman machine adopts the following function to compute the probability of acceptance of the change. \( A(T) = \frac{1}{1 + \exp(-\frac{\Delta E}{T})} \). As a result, the Boltzman machine can return the
best solution. However, it usually takes a long time to compute the solution to avoid the energy from staying in local minimums, so it is very difficult to find the appropriate parameter T in the above function.

II-5-9. Summary of NN

In summary, neural networks were developed to solve the problems that traditional mathematical methodology cannot solve. Neural networks have the ability to represent Boolean functions instead of using a set of <If ~, then ~> rules. Also, the systems have the ability of learning. Therefore, neural network systems do not need the big memory space of traditional expert systems. In short, while a new knowledge base is added manually in expert systems, neural network systems learn the new knowledge by updating their weights. There are two types of neural networks -feed-forward and recurrent. Backpropagation is the most famous as a feed-forward model and applied to pattern recognition fields. The Kohonen self-organizing map is also a feed-forward model and famous as an unsupervised learning model. SOM is used in many studies of clustering analysis. On the other hand, the Hopfield network is the most famous model as a recurrent neural network model, and it is applied in the linear programming area. Many extended models of these typical neural networks have been developed and applied in various areas. In addition, fuzzy theory and genetic algorithms have been applied to optimize the neural networks.
II-6. Image Processing (Machine Vision)

II-6-1. Definition

Image processing is a broad study area about images and is related to computer science, electrical engineering, and mechanical engineering. It is called machine vision in artificial intelligence. The purpose of machine vision is to create computer systems in which computers or electric devices automatically recognize and analyze images. In the study of architectural planning, designers usually apply drawing images, satellite images, map images, grid images, and handwritten sketches to communicate their own design concepts; therefore, it is very important to understand what is communicated with those images. However, it is difficult for computers to recognize those images. Computers cannot easily detect what kind of objects exist on those images and where the objects are located. If computers can recognize the meanings of images to the extent that humans can, the technology of machine vision will be very useful and effective for morphological analysis.

The typical process of machine vision has two parts. The first is basic image processing. Images are transferred to digital images, which are represented as a number or a set of numbers. Matrices are often used because they are easy to manipulate by mathematical functions. The matrices of images are computed and then produce different matrices or feature vectors. In the second step of the machine vision process, the feature vectors are analyzed, classified, or explained in order to complete the required tasks. Statistical methods or expert systems are usually used in this second process.
Machine vision has been rapidly developed since the 1980's. Computers have become a practical means to control image data. However, the main applications of image processing are to solve task-based problems, and computers are not developed enough to recognize images as humans do. The task-based problems are those problems that can be controlled in a limited situation. For example, to judge whether the object carried on a conveyer belt is an orange or an apple, or to detect the sick cells in CT scan images of patients in medicine. In the 1990's, neural networks have often been applied to realize more practical computer systems in image processing because neural networks are more useful for distributed computation. There has been much advanced neural network research in the fields of medical, national defense, weather observation, cognitive science, etc.

II-6-2. Detecting Objects

In image processing, detecting objects in images is one of the most difficult tasks. Children learn to detect objects by training everyday for many years, and they can recognize not only primitive shapes such as rectangle, circle, and triangle, but also complex objects such as animals, cars, houses, and so on. There are two difficulties in implementing this human ability in computers. One is how objects should be defined. For example, although humans can recognize a rectangle easily because it has four lines and four corners, computers need definition of what a line is and what a corner is. In the case of primitive shapes, it is possible to define those features using mathematics. But it is very difficult to define what a cat is or what a house is because there is no clear definition about those complex objects. Another difficulty in implementing the human ability of recognition in computers is how
computers should search for objects on images. In short, it is very difficult for computers to recognize an object when it has been rotated and scaled. It takes huge amounts of time for computers to find objects if all possible templates are searched. For instance, when a picture has one rectangle, humans can recognize the size and rotation degree abstractly, but computers must check the degree of rotation from 0 to 360 and the size from 0 to the size of limitation step by step.

Computers must engage in template matching to detect objects in images. As described above, it takes much time for computers to check all possible cases of rotations and scales repeatedly in order to detect the objects on images. However, there are two strategies to avoid repeated computation. One is to create effective templates and another is to develop the effective search methods that use simple templates. The following section introduces some techniques that have been discovered to delete the necessity of rotation and scaling.

**II-6-2-1. Hough Transform & General Hough Transform**

The Hough transform is a technique to detect lines or curves parametrically. It is usually used after the process of edge detection, and an image is transferred into binary data in which 0 means background, and 1 means edge points. To detect lines or curves on the image, each edge point (x, y) is transferred into some parameter space, usually \((p, \theta)\). For example, each point (x, y) on a line represented as \(p = x \cos \theta + y \sin \theta\) in X-Y space are represented as a line \(c = xm + y\) in C-M space, where \(m = \frac{\cos \theta}{\sin \theta}\), and \(c = \frac{p}{\sin \theta}\) (Fig
2-5-9). If there are some points on which several lines are intersected in the C-M space, the intersected point \((c, m)\) means the line whose slope is \(-m\) and whose intersected point of y-axis is \(c\) in X-Y space.

![Image of Hough Transform algorithm](image)

**Fig. 2-5-9. Algorithm of Hough Transform**

General Hough transform (GHT) is the extended technique of the Hough transform, and famous as a method to detect non-analytic arbitrarily complex shapes. A non-analytic shape is a shape that cannot be represented as the form, \(f(x, p) = 0\), where \(x\) is the point in the image and \(p\) is a parameter vector. The template matching technique compensates for the lack of analytic representation for shapes. The template is called R-table in the general Hough transform. In the R-table, a shape consists of a set of several edge points. Each edge point has 3 values, which are the radius \(|r|\) from the edge point \((x, y)\) to reference point \((\chi, \psi)\), the angle \(\alpha\) between x-axis and vector \(r\) from the edge to the reference point, and the gradient \(\phi\) of the edge. R-table sets are represented as:

\[
R(\phi_i) = \{(r, \alpha) : \exists (x, y) \in \delta S \quad \text{s.t.} \quad \phi(x, y) = \phi_i, \quad \alpha(x, y) = \alpha, \quad \text{and} \quad r(x, y) = r\}
\]

i.e. \(R(\phi_i) = \{(r_i, \alpha_i) : 1 \leq i \leq N_i\}\),

where \(N_i\) is the number of the points which has same gradient \(\phi_i\).
After creating R-table, the GHT is proceeded in the following way. First, all edge points and their gradient in the domain are calculated. Secondly, all possible reference points are plotted in the four dimensional parameter space which has x and y location of the reference point, scale number, and angle of rotation. Finally, the reference point plotted more than the defined number in the four parameters space shows the shape's location, scale, and rotation at the same time.

![Diagram of General Hough Transform](image)

Fig. 2-5-10. Algorithm of General Hough Transform

**Pseudo Code of GHT**

for all edge points \((x_i, y_i)\)

**calculate their gradient** \(G_i\) of \((x_i, y_i)\)

for all set of R-table

get the gradient \(\psi_j\) for the set \((r_j, \theta_j)\)

for scale \(s\) (\(min_s \sim max_s\))

**calculate**
rotate degree \( a = G_i - \psi_j \),

temp scale \( S = s \cdot r_j \),

temp rotate degree \( \varphi = \theta_j + \alpha \),

reference x point \( \text{ref}_x = x_i + S \cdot \cos(\varphi) \),

reference y point \( \text{ref}_y = y_i + S \cdot \sin(\varphi) \),

accumulation number
\[
A(\text{ref}_x, \text{ref}_y, s, \varphi) = A(\text{ref}_x, \text{ref}_y, s, \varphi) + 1.
\]

With a set of edge points and their gradients in the R-table, repetition of template matching and creation of the additional templates of all possible cases of scaling and rotating in GHT are avoided. GHT is useful in detecting the location, scale, and rotation of one complex shape in an image. However, if several complex shapes are interrupting each other, GHT often make mistakes because there are several possibilities in solving the combination of shapes. (Fig.2-5-11). Also, noise edge points lead to mistakes. Therefore, effective noise reduction is required before the process of GHT.

![Fig. 2-5-11. Case of interruption](image)

### II-6-2-2. Contour Based Algorithms

GHT is a technique to use edge data directly. On the other hand, the simplest and easiest technique of detecting objects is to use contour data, which is the boundary information of
objects. The process of detecting objects with contour data is as follows. First, an image is transferred into binary data, then the binary data is calculated to produce contours by convolution mask of chain codes. Chain codes are a notation for recording the list of edge points along a contour (Fig. 2-5-12). To avoid repeating calculation and additional templates in template matching, effective templates should be applied to this contour-based algorithm. In short, the template should include the effect of rotation and scale. For example, the template of a rectangle should be defined as the contour that has 4 lines, the line's direction increases or decreases 2 by 2 along the contour. If the set of values consisting of a contour is (2 2 2 4 4 4 6 6 6 8 8 8), then the contour is evaluated as a rectangle because it has 4 lines and each number increases by 2. In this case, the length of rectangle is 3 and the rectangle is rotated 45 degrees, because each line has 3 numbers and the combination order is (2, 4, 6, 8) as shown in Fig. 2-5-13.

![8 Directions Convolution Mask for Chain Codes](image)

Fig. 2-5-12. Chain Codes
This algorithm is useful for detecting primitive shapes, which are rectangles, circles, triangles, and so on, because it is easy to define the attributes of primitive shapes in mathematics. However, it is difficult to apply for complex shapes because it is difficult not only to define the character of shapes but also to detect contours from edge images.

II-6-2-3. Neural Network Based Algorithms

In neural network based algorithms, a network is used to detect objects as a template. When detecting complex shapes, it is difficult to define their characters and attributes mathematically. Therefore, neural networks are applied because they are better at representing the relations using network structure instead of a set of <If ~, then ~> rules or
mathematical notations. Several techniques based on neural networks have been developed. Some of the examples that have similar algorithms are Kohonen self-organizing map (SOM), SNAKES, multiple elastic modules (MEM) (Shams, 1995). The following section introduces the MEM model as a typical network technique.

In the techniques based on neural networks, the objects to be detected are represented as a set of neurons. Each neuron is interconnected with specific relations to represent a specific object as shown in Fig. 2-5-14. The detecting process of the MEM has three parts. First, the neurons representing the features of an object are distributed randomly on an image, and one pixel on the image is selected. Secondly, the similarities between the selected pixel and all neurons are calculated, and the most similar neuron becomes a winner neuron. The similarity is computed as $m_i^* = \min_{j \in A} \{\| m_j - x \| R(F(x), f_j) \}$, where $j$ is the neuron's unit number, $m_i^*$ is the center location of the winner neuron, $m_j$ is the center location of the neuron $j$, $x$ is the location of selected pixel, and $R(x, y)$ is the bounded monatomic nonlinear measure of the similarity between two features $x$ and $y$ having a range $(0, 1)$. Thirdly, the center location of all neurons connecting to the winner neuron is updated by the self-organizing rule $m_i^* = m_i^* + a(x - m_i^*)$. Iterating the second and third processes 1,000 times or more, each neuron finds the most accurate location.
In the MEM model, an object is defined as being composed of elements, and neural networks represent the relations among the elements. As this technique has been proved to be the same process used by humans searching for objects in images, it is possible to detect complex objects such as airplanes and cars (Shams, 1995). The problem is that it takes much time and labor to make templates of the network representation beforehand. However, neural networks searching algorithms are more flexible and practical than GHT or contour based techniques, and many applications using neural networks have been developed, especially in the fields of medicine, national defense, and weather observation.

**II-6-3. The Other Problems**

There are many problems in the realization of practical computer systems that can detect
objects automatically. The issue of noise reduction is one example. Since all images have some noise, it is necessary to delete the noise completely or to set the maximum noise that is permissible. However, it is still difficult because there are so many different kinds of noise. Also, when some objects interrupt each other on an image, it is almost impossible to detect the objects with the current techniques. Therefore, many researchers are trying to discover more effective detecting techniques. Today the neural networks, fuzzy theory, and genetic algorithms are considered to be the most effective ways, but they have not been developed practically enough to detect objects as humans do.
Chapter III

Literature Review

This chapter reviews the literature in the study areas of building extraction and soft computing. The study area of building extraction is one of the applications of image processing. Soft computing is one of the applications that combined the technologies of neural networks and fuzzy systems, and is often applied to machine learning in AI in order to build human knowledge. This chapter also reviews the theory and algorithms of fuzzy multiple layers perceptron (MLP), which is one of the applications of soft computing and applied to this research project as a main model.

III-1. Building Extraction

III-1-1. History of Building Extraction

Image processing with knowledge-based database is basically a combination of traditional image processing technologies such as edge detection, labeling, pattern classification, and segmentation, and expert systems with a set of \(<\text{If } \sim, \text{ then } \sim>\) rules. The original study of monocular building extraction began in the 1980s. The initial approaches used region growing techniques and simple models of imaging geometry (Conners, 1984), (Harwood, 1987), (Herman, 1986), (McKeown, 1984), (Nagao, 1980), (Nicolin, 1987), (Tavakoli, 1982). Later, shadow analysis and line-corner analysis to derive building structure were considered (Huertas, 1988), (Irvin, 1989), (Liow, 1990), (Mckeown, 1990), (Mohan, 1989), (Venkateswar, 1990). Modern approaches have used elaborate data-driven methods with...
multiple modes of analysis to infer 2D and 3D building structure (Fua, 1996), (Jaynes, 1994), (Lin, 1994), (Lin, 1996), (Shufelt, 1993).

The study of extraction of man-made objects from aerial and space images is broadly categorized into 4 fields according to the book edited by A. Gruen (Gruen, 1995). These are theoretical research, building extraction, road extraction, and map-based extraction. Theoretical research is based on the mathematical methodology in image understanding from aerial and satellite image. Building extraction is the study of developing systems to extract buildings and their attributes in the images. Road extraction is the study of developing systems to extract roads and their attributes. Map-based extraction is the study of developing systems to extract buildings and roads with the help of map information.

Active research topics in building extraction include (1) multi-image and semi-automated techniques using 3D feature extraction (Fisher, 1997), (Hsieh, 1996), (Roux, 1994), DEM analysis (Haala, 1997), (Haala, 1997-2), (Jaynes, 1997), map-based extraction (Roux, 1997), and color image analysis (Henricsson, 1996).

III-1-2. Research Projects in Building Extraction

The following describes typical research projects in monocular building extraction.

Feldman and Yakimovsksy

In the model of J. Feldman and Y. Yakimovsksy (Feldman, 1974), mathematical decision making and heuristic search algorithms are applied for developing a standard scene
analysis system. In the system, the segmented regions are assigned to primitive shapes such as rectangles, squares, circles, etc, by calculating probabilities with several measurements (e.g. color, area). Then, the system uses a heuristic search to merge primitive regions. The goal of this system is only to find the best global interpretation.

**Barrow and Tenenbaum**

IGS and MSYS are the systems developed by H. Barrow and J. Tenenbaum to segment the building areas in satellite images.

In IGS, (Tenenbaum, 1977), various knowledge, which is a set of \(<\text{If } \sim, \text{ then } \rightarrow>\) rules describing the relations between the attributes of pixels and objects, is applied to infer the best interpretation of images. First, an image is partitioned into elementary regions consisting of pixels with the same attributes. Then, the system merges the partitioned regions recursively by using knowledge until all adjacent regions have different interpretations.

In MSYS, (Barrow, 1976), likelihoods based on local attributes of pixels such as color, size, and shape are considered to make regions in IGS. The process of merging is continued until the regions obtain specific values of likelihoods.

**Selfridge**

P. Shelfridge developed a three-level system, (1) model experts in which objects and their expected locations are represented, (2) segment expert in which the best segmentation is
selected for a given situation, and (3) parameter expert in which the best threshold value is searched to extract a region matching the expected appearance (Shelfridge, 1982). The evaluation of success and failure in each expert is repeated automatically until all rules have been applied.

**Hwang**

In SIGMA, (Hwang, 1985), a similar approach to Shelfridge is applied. The system uses blobs and ribbons to extract houses and roads. The shape fixing techniques are first applied, then the locations of other objects are predicted by using the detected blobs and ribbons. For example, if there are similar ribbons near the pre-detected ribbon, the ribbons are combined to the pre-detected ribbon and the integrated ribbons are verified as a road using hypotheses generated by the system.

**McKeown, Enlinger, McGlone, and Shufelt**

The following systems, MACHINESEG, BUILD, SHAVE, SPAM, VHBUILD, PIVOT, and SiteCity, are continuous research projects developed in the Digital Mapping Laboratory at Carnegie Mellon University.

In MACHINESEG, (McKeown, 1984), map knowledge is applied to segment images in the process of region growing. The system calculates the correspondence between maps and images from camera and terrain models. The intensity of pixels and edges are used in the merging process. The merging process is continued by checking the criteria, which is driven from the corresponding maps, until each region is identified as a prototype region.
In BUILD, (McKeown, 1990), a line and corner-based analysis is applied. It uses 2D-range search to find pairs of edges that meet at approximately right angles. The detected edges are defined as candidates of corners. The system then connects the corners if the connection is appropriate to create convex quadrilaterals. Shadows cast from boxes are also considered in determining the boxes as buildings.

McKeown also developed an integrated expert system called SPAM, which is a production system architecture for interpretation of aerial imagery using the OPS5 production system language. In the system, buildings, houses, streets, and the other objects comprising cities are detected and modeled for creating maps.

In SHAVE, (Irvin, 1989), the shadow length and direction are estimated to define the height of buildings for the detected boxes in BUILD. 2D boxes with height information are created in this system. In VHBUILD, an extended system of BUILD, (McGlone, 1994), the vanishing point labels on the verified boxes are applied to determine whether the box is the roof of a flat roof building or a peaked roof building in 3D. In PIVOT, (Shufelt, 1996), horizontal, vertical, and slanted vanishing points are used to improve the building shape detection. Meanwhile VHBUILD assumes only nonhorizontal and nonvertical edges. PIVOT also can apply the camera model and knowledge about solar elevation and azimuth in an object-space based shadow verification test and a surface intensity consistency test in order to produce rectangular and triangular primitive volumes.
SiteCity, (Hsieh, 1996), is a semi-automated 3D site modeling system in which it is possible to construct and manipulate 3D building objects from multiple images.

**Harwood and Chellappa**

PQ (Picture Query), (Harwood, 1987), consists of three subsystems, which are a high-level vision system (HLVS), representing goals and knowledge about objects, and low-level vision system (LLVS). The HLVS has the description of object types, a database of facts giving the current state of interpretation, and parameters for the initial segmentation and search for given object types. In LLVS, the segmented regions are refined into homogeneous connected components by smoothing edges and defining object types. The object types are represented by using attributed graphs with nodes and relational lines.

CD (Change Detection), (Chellappa, 1994), is a system developed for image analysts. The system allows them to specify what are important changes through quick look (QL) profiles, and to select the appropriate IU algorithms for detecting these changes. This system is one project of RADIUS (Gee, 1993).

**III-2. Soft Computing**

**Definition**

Soft computing is a field of study in artificial intelligence (AI). It is the study of implementing the abilities of adapting, learning, and recognizing in computers by using the techniques of neural networks, fuzzy systems and genetic algorithms. Especially, the
The fusion of neural networks and fuzzy systems has been researched and developed well, because it is possible to translate the output values of neurons in neural networks into the membership value in fuzzy systems. Neural networks can represent and calculate the complex relations between inputs and outputs with the weight matrixes. The most salient feature of neural networks is their learning ability. Neural network systems can learn by updating their weights during training sample data. On the other hand, fuzzy systems can represent ambiguous and abstract rules by using linguistic terms such as high, medium, low, very high, etc. Therefore, the fusion of neural networks and fuzzy systems makes it possible to create systems that can learn complex relations between inputs and outputs and represent the relations with a set of \(<\text{If } \sim, \text{ then } \sim>\) rules in linguistic forms at the same time.

**Categories in Neuro-Fuzzy Systems**

Several different techniques of combining neural networks and fuzzy systems have been introduced since the 1990s. In general, the methodologies can be classified as follows (Pal, 1996).

1. Incorporating fuzziness into the neural net framework
2. Designing neural networks guided by fuzzy logic formalism
3. Changing the basic characteristics of the neurons
4. Using measures of fuzziness as the error or instability of a network
5. Making the individual neurons fuzzy
In the model of category 1, the concept of fuzziness is incorporated into neural networks. For example, the input of the input layer is modified as a set of vectors with fuzzy properties such as high, medium, low, etc., and the output of the output layer is fuzzy labeled vectors. This model is often used for classification problems (Keller, 1985). Fuzzy Multiple layer perceptron (MLP) (Pal, 1992), fuzzy logical MLP (Mitra, 1994), and fuzzy Kohonen network (Mitra, 1994-2) all belong to this category.

In the model of category 2, neural networks are used to help calculation of fuzzy models. (Huntsberger, 1990) (Bezdek 1992)

In the model of category 3, the integration and transformation functions at each node in neural networks are replaced by the fuzzy aggregation operations (fuzzy union, intersection, etc) such as the fuzzy layered neural net-based classifier developed by Pedrycz (Pedrycz, 1991), (Pedrycz, 1992) and fuzzy ART and fuzzy ARTMAP by Carpenter (Carpenter, 1991), (Carpenter, 1992).

In the model of category 4, various fuzziness measures are used to determine the instability or energy functions in neural network systems (Ghosh, 1993). This strategy is applied for feature extraction in image processing with self-organizing systems.

In the model of category 5, each neuron behaves as a fuzzy processor. Therefore, the input and output of the neurons are represented as fuzzy sets. The neuron is called a fuzzy neuron,
and was initially introduced by Lee and Lee in 1975 (Lee, 1975). They also proved the possibility of generalizing traditional neurons as fuzzy set theory.

**III-3. Fuzzy MLP**

This section introduces the details of the fuzzy MLP model developed by Pal and Mitra [Pal, 1992]. This model belongs to the category-1 model, which incorporates fuzziness in the input and output layers of neural networks as described in Chapter III-2. The structure of the fuzzy MLP system is based on multiple layers perceptron, which has one input layer, one output layer, and one or more hidden layers.

**III-3-1. Input Data in Fuzzy MLP**

The input data is represented as a set of feature values. Each feature is represented in terms of some combination of membership values in the linguistic property sets low (L), medium (M), and high (H). For example, when the $j$th input feature $F_j$ is low, the feature is represented as, $F_j = \{\mu_{low}(F_j), \mu_{medium}(F_j), \mu_{high}(F_j)\} = \{0.95, 0.5, 0.05\}$. The input is represented as a set of these features, $F = [F_1, F_2, \ldots, F_n]$, where $n$ is the number of features. The number of input neurons must be $3n$, because each feature $F_j$ needs 3 input neurons in fuzzy MLP.

**III-3-2. Passing Values in Fuzzy MLP**

In fuzzy MLP, each neuron in the hidden and output layers receives the total input from all
neurons in the previous layer. Then, the total input $x_{j}^{h+1}$ at the $j$th neuron in the $h+1$th layer is defined as

$$x_{j}^{h+1} = \sum_{i} y_{i}^{h} w_{ji}^{h} - \theta_{j}^{h+1},$$

where $y_{i}^{h}$ is the output of the $i$th neuron in the previous layer $h$, $w_{ji}^{h}$ is the weight of the connection from the $i$th neuron in the $h$th layer to the $j$th neuron in the $h+1$th layer, and $\theta_{j}^{h+1}$ is the threshold of the $j$th neuron in the $h+1$th layer. The output value of each neuron is calculated by a monotonic nonlinear function such as a sigmoid function with the total input described above, and is given as $y_{j}^{h+1} = \frac{1}{1 + e^{-x_{j}^{h+1}}}$.

Only the output of the neurons in the input layer is given as $y_{j}^{0} = x_{j}^{0}$.

### III-3-3. Output Data in Fuzzy MLP

The output data in fuzzy MLP is represented as a set of membership values $(\mu)$, lying in the range $[0,1]$, of each class. The output value of each neuron means the membership value for the input data. For example, if the $k$th output neuron has the value 0.8 for the given input data $F$, $\mu_{Ck}(F)$ is 0.8, where $C_{k}$ represents Class $k$, and $\mu_{Ck}(F)$ is the membership value of Class $k$ for $F$. In short, the input data $F$ belongs to Class $k$ with high probability. The output data is represented as a vector $[\mu_{C1}(F), \mu_{C2}(F), ..., \mu_{Cj}(F)]$, which is a set of the membership values described above, where $j$ is the number of output neurons as shown in Fig. 3-3-1.
III-3-4. Weight Updating in Fuzzy MLP

During the training process, the weight matrices are updated in order to minimize the error between the system's output data and desired output data. The weights are updated from the output layer to the input layer backward repeatedly. In fuzzy MLP, the least mean square error is applied and it is defined as

$$E(w) = \frac{1}{2} \sum_{j,c} (y_j^{\text{H},c}(w) - d_{j,c})^2,$$

where $y_j^{\text{H},c}(w)$ is the output of jth neuron in the output layer in the case c, and $d_{j,c}$ is the desired value of its case. The method of gradient descent is applied to minimize this error $E(w)$, and the set of weights are updated as

$$\Delta w_{ji}^h(t) = -\varepsilon \frac{\partial E}{\partial w_{ji}} + \alpha \Delta w_{ji}^h(t-1)$$

where $\varepsilon$ is a positive constant learning rate, $\alpha$ is the momentum factor lying in the range [0, 1], and $t$ is the number of the iteration in training. From the above formulations, the updating amount is calculated as

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial y_j} \frac{\partial y_j}{\partial x_j} \frac{\partial x_j}{\partial w_{ji}} = \frac{\partial E}{\partial y_j} y_j^h(1 - y_j^h)y_i^{h-1}.$$
In the above equation, the weights in the output layer are updated by using

$$\frac{\partial E}{\partial y_j} = y_j'' - d_j,$$

where $H$ is the index of the output layer. Otherwise, the weights in the other hidden layers are updated by using

$$\frac{\partial E}{\partial y_j} = \sum_k \frac{\partial E}{\partial y_k} \frac{\partial x_k}{\partial y_j} = \sum_k \frac{\partial E}{\partial y_k} \frac{\partial x_k}{\partial y_j} w_{kj},$$

where the $j$th neuron and $k$th neuron lie in the $h$th layer and $h+1$th layer. Then, a backward pass, starting from the output layer, is used to enable weights updating until the pass reaches the input layer.

In conclusion, the methods of weight updating and passing values are the same as those of the standard multiple layer perceptron with backpropagation. Only the input and output data representation in fuzzy MLP is different from the non-fuzzy MLP, because the data is represented as a fuzzy set or a set of membership values.

### III-3-5. Rule Generation and Inferencing in Fuzzy MLP

This section introduces the algorithm of how fuzzy MLP generates and infers rules from its networks. It is based on the study of S. Mitra (Mitra, 1995). The algorithm is to find the paths from the neurons in the output layer to the neurons in the input layer. The connected neurons in the input layer represent the factors of if-clause, and the connected neurons in the output layer represent the factors of then-clause. For example, when the system finds...
the path from Class1 neuron in the output layer, and $F_{1,\text{low}}$ and $F_{2,\text{high}}$ neurons in the input layer as shown in Fig. 3-3-2, the system generates a fuzzy rule, "If $F_1$ is medium AND $F_2$ is high, then Class1." The certainty of Class1 depends on the output value from the Class1 neuron.

![Diagram](image)

Fig. 3-3-2. Example rule generation scheme

### III-3-5-1. Input Data, Passing Values, and Output Data

The input data is represented the same as fuzzy MLP described in Chapter 3.1. However, several additional variables are required in the process of passing values in order to generate fuzzy rules. For each neuron $j$ in the $h+1$ th layer, the following variables should be defined:

1. a confidence estimation factor $conf_{j}^{h+1}$,
2. an ambiguous estimation factor $noin_{j}^{h+1}$ derived from (3)-(5),
3. a unknown amount $unknown_{j}^{h+1}$ for the value of (2),
(4) a known amount $known_{j}^{h+1}$ for the value of (2), and

(5) a certainty factor $unden_{j}^{h+1}$ to calculate the value of (1)

If there is no information about the $i$ th neuron in the input layer, $noin f_{i}^{0} = 1$. Otherwise, $noin f_{i}^{0} = 1$. The variable $unknown_{j}^{h+1}$ is the sum of the weighted information from the preceding ambiguous neurons, and calculated as

$$unknown_{j}^{h+1} = \sum_{i} w_{ji}^{h} y_{i}^{h},$$

$$unden_{j}^{h+1} = \sum_{i} |w_{ji}^{h}|,$$

for all $i$ th neurons having $noin f_{i}^{h} = 1$ in the previous $h$ layer. The variable $known_{j}^{h+1}$ is the sum of weighted information from the preceding non-ambiguous neurons, and calculated as

$$known_{j}^{h+1} = \sum_{i} w_{ji}^{h} y_{i}^{h},$$

for all $i$ th neurons having $noin f_{i}^{h} = 0$ in the previous $h$ layer. The ambiguous estimation factor of $i$ th neuron in the hidden and output layers, $noin f_{j}^{h}$, is defined as

$$noin f_{j}^{h} = \begin{cases} 1 & \text{if } |known_{j}^{h}| \leq |unknown_{j}^{h}| \\ 0 & \text{otherwise} \end{cases},$$

Finally, the confidence estimation factor of each neuron is defined as

$$conf_{j}^{h} = \begin{cases} \frac{\sum_{i} y_{i}^{h-1} w_{ji}^{h+1}}{unden_{j}^{h}} & \text{if } noin f_{j}^{h} = 1 \text{ and } h > 0 \\ y_{j}^{h} & \text{otherwise} \end{cases}.$$
salient the result is comparing to the other choices, and is defined as

\[ \text{bel}_j^H = y_j^H - \sum_{i \neq j} y_i^H. \]

This value is used to choose linguistic forms among very likely, likely, more or less, not unlikely, and unable to recognize. The chosen forms are applied to modify then-clauses in the generated fuzzy rules. The following shows the decision rules.

1. Very likely for \( 0.8 \leq \text{bel}_j^H \leq 1 \)
2. Likely for \( 0.6 \leq \text{bel}_j^H < 0.8 \)
3. More or Less for \( 0.4 \leq \text{bel}_j^H < 0.6 \)
4. Not unlikely for \( 0.1 \leq \text{bel}_j^H < 0.4 \)
5. Unable to recognize for \( \text{bel}_j^H < 0.1 \)

III-3-5-2. Querying

If there is any neuron \( j \) in the output layer \( H \) with \( \text{noin} f_j^H = 1 \), then this querying phase begins. First, select the most certain neuron \( j^* \) whose \( \text{conf}_{j^*}^H \) is maximum among the ambiguous neurons with \( \text{noin} f_j^H = 1 \). The path from neuron \( j^* \) in layer \( H \) is pursued in a top-down manner, to find the ambiguous neuron \( i^* \) in the previous layer (\( h=H-1 \)) such that

\[ \left| w_{j^*i^*}^h \times y_{i^*}^h \right| = \max_i \left| w_{ji}^h \times y_i^h \right|, \]

where \( \text{noin} f_i^h = 1 \). This process is repeated until \( h \) reaches to the input layer. Then the system queries the user for the value of the corresponding input feature. Once the requested
input variable is supplied by the user, the procedure of passing values described above is followed. Finally, all neurons in the output layer have \( n_{oj}^{H} = 0 \). This process lets the user know what kind of information is lacking in the networks, and also enables the user to make the system more appropriate to his/her needs.

**III-3-5-3. Justification**

Justification is the process to finding the maximum weighted paths in the networks and then generating fuzzy rules. This section explains how to find the paths from the neuron \( j \) in the output layer \( H \). First, select one output neuron \( j \) with \( bel_{j}^{H} > 0 \). Then, the weights \( w_{ji}^{H-1} > 0 \) connected to the neuron \( j \) are selected in the lower layer \( H-1 \). These neurons \( i \) connected to the neuron \( j \) are described as \( m^{H-1} \), a set of indexes \( a_{k}^{H-1} \). The maximum weight \( wet_{i}^{H-1} \) in the layer \( H-1 \) is \( w_{ji}^{H-1} > 0 \). The algorithm to find the maximum weighted paths is carried out in a top-down manner. For the hidden layers \( h \), the maximum weight \( wet_{i}^{h} \) is calculated by the sum of the predefined maximum weight in the upper layer and the weights in the lower layers, and is defined as

\[
y_{i}^{h} > 0.5, \\
wet_{i}^{h} = \max_{a_{k}^{h+1}} \left[ wet_{a_{k}^{h+1}} + w_{a_{k}^{h+i}}^{h} \right],
\]

where \( a_{k}^{h+1} \) indicates the index number of neurons in the upper layer \( h+1 \), \( wet_{a_{k}^{h+1}} \) is the predefine weight in the upper layer, and \( w_{a_{k}^{h+i}}^{h} \) indicates the weights of the neurons \( i \) in the lower layer \( h \). When the maximum weighted paths reach the input layer, the connected
neurons in the input layer become the candidates for if-clauses. Then the fuzzy rules are generated among the candidate neurons $m^0$ until

$$\sum_{i_s}^{wet_s} > 2\sum_{i_n}^{wet_n},$$

where $i_s$ indicates the input neurons selected for the clause and $i_n$ indicates the input neurons remaining from the set of $m^0$ as shown in Fig. 3-3-2. In this case, the generated rule from the system would be "IF F1 is very medium AND F2 is high, THEN likely Class 1."

**III-4. Summary**

In this section, the reviews of the research about building extraction and soft computing are described. The theories and algorithms of fuzzy multiple layers perceptron (MLP), one of the models of soft computing, are explained because this model is applied to this research project. Next section represents the statement of problems in applying AI technologies for CAD/CAAD systems.
IV-1. Problems

IV-1-1. Complex Interfaces

Although recent computer systems for design can operate many complex tasks at the same time, they are not easy to use for designers. When using any recent CAD and CG animation software, it is possible to create very complex three-dimensional houses, buildings, and sculptures, but it is hard to learn and master the use these programs. In order to create complex shapes in computers, designers and planners have to learn a lot of complex functions that are completely different from what they usually use in their daily work.

For example, when a planner wants to set back the buildings in the north side of the planning lot in his/her current computer system, the following steps are required:

1. Activate x and y parameters among the attributes of (x, y, z).
2. Activate an object mode instead of neither a point mode nor a face mode.
3. Manually select objects, which he/she thinks are located in the north, one by one.
4. Set the direction of movement to "south."
5. Calculate the distance of movement.
6. Move the selected objects either by moving the mouse or by entering a real number for required distance.
7. Have the computer systems calculate new three-dimensional data and show its image on the screen.

8. The planner needs to repeat the same manipulation again and again until he/she is satisfied with the change that he/she makes.

These steps are troublesome for many planners and designers because they have to learn and understand all the steps to be made before beginning to design and plan. In fact, every time they buy new software and update their system, they have to learn all new functions and steps all over again.

The main reason why the interfaces of CAD/CAAD systems are complex is that the systems make it possible to create complex objects by combining complex mathematical functions. Though it is easy to update the systems by adding new functions in this approach, the interfaces become so complex with buttons and commands that the user cannot use them easily. Moreover, the user has to define parameters for the complex mathematical functions, which are unfamiliar to him/her. Therefore, it is necessary to develop CAD/CAAD systems in which the users can manipulate objects and data in the way that they are accustomed to.

**VI-1-2. Similar Cases**

The second problem of current CAD/CAAD systems is that they have poor behaviors in similar situations. For example, when designers create objects, they must design them from
the beginning even if they have already created similar objects before. In the case of architecture, when an architect designs a new house that is different only in the shape of the living room from the house designed before, he/she must design the new house without any design concepts applied previous cases. Though most of the CAD/CAAD systems try to solve this problem by adding libraries of objects and functions to change the attributes, the approach makes the systems complex as described above. In addition, the users intend to design the objects such that the systems support a lot of alternatives. In other words, they design some objects not because they want to create them but because the system support the objects. As a result, they try to design not what they want but what the systems support. In other words, the systems do not help the users' designing but restrict their design ideas. This is contrary to the purpose of CAD/CAAD systems. Of course, it is difficult to define what a similar case is in designing, and the definition is different from designer to designer. Therefore, it is necessary to develop the CAD/CAAD systems that can adapt to similar cases that are defined by different users.

VI-1-3. Reuse of Knowledge

The third problem of current CAD/CAAD systems is that they cannot reuse the users' knowledge, ideas, or concepts in designing. For example, when designers create objects, they must apply a set of functions built into the systems. If they apply the same combination of functions to the objects as they used before, they must repeat the same functions with the same orders again. Some graphic software applications such as PhotoShop by Adobe Corp. have the commands to store the process history in the memory
and to apply a set of processes to different cases. This approach saves time in designing by avoiding repetition and allows many trials and errors for the users. However, it is still far from the goal of storing the design knowledge, because the system cannot store the semantic rules. It can only store the syntax rules in designing. Needless to say, it is difficult to define what the semantic rules in designing are. Also, the definition varies from designer to designer. Therefore, it is necessary to develop CAD/CAAD systems that can store semantic rules and allow the users to define the rules flexibly.

VI-1-4. Rule Representation

The fourth problem of current CAD/CAAD system is that it is difficult to represent rules for expert systems in the domain of architecture. It can be said that the three problems described above are more or less caused by difficulty in representing design knowledge, concepts and ideas as a set of rules. This also makes it difficult to develop expert systems in architecture and urban planning. The rules in architecture and urban planning are completely different from the rules applied in expert systems of medical, engineering, and natural scientific domains. The rules in medical, engineering, and natural science are based on mathematical knowledge and encyclopedic knowledge, and the results of inferences from the rules should be always be the same. On the other hand, the rules in architecture and urban planning are a combination of mathematical knowledge, encyclopedic knowledge, and human knowledge, and is effected very heavily at the beginning of the design process by human knowledge. The results of inferences from the rules vary from designer to designer.
The research on human knowledge has been one of the biggest topics in science, and several methods of using neural networks, fuzzy systems, and genetic algorithms have been proposed. However, it is a fact that the effective solution for representing human knowledge has been neither discovered nor developed yet.

Considering the above situation, it is useful to develop computer systems that provide support in establishing the user's own rules because it is still impossible to implement human knowledge in computers.

**IV-2. Causes**

The main reason why the current CAD/CAAD systems have the problems described in the above sections is mainly caused by the fact that they cannot represent, store, or manipulate design knowledge as rules. If they had the abilities of representing, storing, and manipulating the design knowledge, the users could communicate with the systems in natural language the same way users of other domain expert systems without complex interfaces do. The users could also apply the knowledge to similar cases and reuse the knowledge in designing, and each user could establish his/her own rules in designing for the other expert systems. In other words, the user could use the systems as their design and planning partners. Specifically in architecture and urban planning, 3D/2D objects could be generated automatically from images and drawings, and planners and designers could manipulate the objects interactively through the communication with the systems.
IV-3. Hypothesis

Today many researchers believe that the applications of neural network techniques is the most effective way to implement systems for storing human knowledge. If the systems with neural network techniques can be created in reality, it will become possible to (1) implement both the mathematical and humanistic factors in the system at the same time, (2) reuse the methodologies and concepts previously created, and (3) demonstrate simulation and presentation in real time in the system.

Therefore, it is useful to apply neural network techniques in CAD/CAAD systems and examine the possibility of how the techniques can contribute to the design and planning processes in CAD/CAAD systems.

When applying neural networks in architecture and urban planning, image processing and natural language processing play important roles because planners and designers evaluate their works on monitors and paper, and discuss their outputs with specific terms. Neural networks have been studied in the fields of image processing and natural language processing. The techniques of neural networks in image processing are useful to create 3D objects automatically from images in computers, and the techniques of neural networks in natural language processing are useful to manipulate 3D objects in computers. Therefore, many researchers are trying to develop CAD/CAAD systems that implement image processing and natural language processing in order to manipulate design knowledge.
However, there are some disadvantages when applying neural network techniques to the CAD/CAAD systems for building knowledge. For example, it takes a lot of time and labor for the users to store their information into neural networks because training data in CAD/CAAD systems requires creating a huge number of example objects. It is also difficult for the users to check whether the neural networks have enough ability and how the networks learn their knowledge because the only way to check the ability is to apply practical cases and observe the results. Moreover, the common difficulties of the neural network systems in defining the optimum network structure, the optimum sample number, optimum training repeating times, etc. can be predicted.

IV-4. Proposal

The main problem in applying neural networks to CAD/CAAD systems is how to train the weights of neural networks, because a lot of sample data is required to train them. There are two training methods. One is training by samples. It is to give the expected output according to the input data one by one manually. The other method is training by communications. It is to train the system through communication in natural languages. The first method is applied to image processing in AI, and the second method is applied to natural language processing in AI. It is reasonable and proper to apply the first method to the CAD/CAAD systems as a proposed system, because it is difficult to represent the design knowledge as a linguistic form in natural language processing. However, the disadvantages of training the weights by giving expected outputs one by one are that it takes a long time to train the weights and it is difficult for the user to understand how they
are trained well. Currently, fuzzy systems and genetic algorithms have succeeded in reducing the labor in training of neural networks. Fuzzy systems help the training process by generating fuzzy rules that the networks learn. Genetic algorithms help to optimize the structures of networks during the training process. In this project, it is reasonable and proper to apply fuzzy systems because designers and planners can understand their uncertain design knowledge if the CAD/CAAD systems produce their ideas as a set of fuzzy rules in linguistic forms.

Considering the above facts, it is possible to develop a new CAD/CAAD system that applies the technologies of neural networks and fuzzy systems for building design knowledge. This project examines the possibilities of the system by applying image processing, specifically in building extraction from satellite images because it is possible to apply the training by samples in image processing. In this case, the design knowledge can be defined as the relations between the given images and applied morphological functions. Moreover, the system will be more useful if the system can generate computer objects from images because it can be applied to create 3D computer city models, recognizing architectural and urban planning drawings, etc. In addition, the generated fuzzy rules can be applied to other expert systems in designing and planning.

IV-5. Limitation

- The purpose is to test the possibility of neural networks in CAD/CAAD systems, not for evaluating the design but for building design knowledge.
The design knowledge is defined as the relations between the given situations and the applied functions by the users. Specifically, this project applies the relations between satellite images and applied morphological functions to generate 3D computer city models.

The goal of the system is to make clear what kind of information is to be stored from the relations in the system and how to reuse the information in designing.

Fuzzy systems are integrated into the system to reduce the cost of training by generating fuzzy rules that the neural networks learn.

The generated fuzzy rules are observed by the users, but the rules are not applied to expert systems in this project.

In order to examine the above issue, this project applies a system of monocular building extraction from satellite images as a test case.

IV-6. Summary

This section represents the current problems in CAD/CAAD systems, which are the complex interfaces, the poor flexibility for similar cases, the difficulty in reusing the user's design knowledge, and the difficulty in representing design rules. The main reasons for the problem is that the systems cannot represent, store, or manipulate design knowledge. Since
neural networks are considered to be effective at building knowledge in AI, this research proposes to apply neural networks for building design knowledge into the CAD/CAAD systems. The proposal is to develop a system in which training by samples and fuzzy systems are integrated in order to reduce the labor in training. In practice, this section proposes a system to generate 3D computer city models from satellite images. In this case, the design knowledge can be defined as the relations between the images and the applied morphological functions of the users.
Chapter V

Methodology

V-1. Introduction

One of the problems in traditional monocular building extraction is that most of the systems cannot be used for different types of satellite images such as those in Fig. 5-1-1, because they use expert systems for specific tasks. Though each strategy should be changed according to the type of satellite images, most expert systems do not have the flexibility to change their strategy depending on the type of satellite image. One way to solve this problem is to prepare different functions for each case, such as city area, country area, residential area, and mountain area. However, it is very difficult to classify the type of images for both humans and computers when one image has a combination of city area, country area, residential area, and mountain area. The target of this program tries to give one solution for this problem by implementing adaptive ability with training and refining processes.

Fig. 5-1-1. Satellite Images
The four images in Fig. 5-1-2 show the typical processes of monocular building extraction with supervised learning. The top left image shows the result of edge detection calculated automatically with the Sobel filter (Nevatia 1982). The top right image shows the contours of buildings. The red lines represent the building boundaries. The bottom left image shows the expected results of monocular building extraction. The bottom right describes the process of labeling for selected pixels. For training neural networks with supervised learning algorithms, it is required to train the networks by giving sample data. In the process of training, the system's users must pick up some sample pixels in the images manually or randomly and define the expected output data corresponding to each selected pixel. In the bottom right image, the red crosses represent the pixels that constitute a building in the satellite image, the green ones represent ground, and the blue ones represent shadow.

Fig. 5-1-2. Output Images in Building Extraction
V-1-1. 3D Computer City Model

As stated, the final goal of this research is to develop a computer system for creating 3D computer city models from satellite images by applying the technology of neuro-fuzzy systems. The example 3D computer city models are shown in Fig. 5-1-3.

Fig. 5-1-3. 3D Computer City Models

The top model shows the 3D boxes and a flat plane. The boxes represent buildings, which are detected from satellite images, and the plane represents the ground of the city. All boxes have the same height. On the other hand, the heights of buildings in the middle image are not the same. As described before, the final goal is to allow the system to handle the required objects by users. This research tries to realize the allowing by labeling objects and learning user's design knowledge. Labeling is when designers and planners train the neuro-fuzzy system by labeling expected results with linguistic terms. Learning design knowledge means updating the weights of neuro-fuzzy networks. The weights store the
relations between the labeled areas in images and the applied functions to the labeled areas. For example, if the image has an ill-defined rectangle labeled as a house, the rectangle will be fixed rigidly and given the information of height as a 2-floor house. As a result, the system’s users can manipulate and design computer city models. Finally, the 3D computer city model is texture-mapped with the satellite image as shown in the bottom image, and designers and planners can evaluate, simulate and visualize their plans by observing the models.

V-1-2. Segmentation & Interpretation Stages

When designers and planners use this system, there are two main stages. The first stage is segmentation. It is to categorize pixels into several classes where the system's users can define the number of classes and their labels in linguistic form such as "small house", "big building", "building's shadow", etc. The left and center images in Fig. 5-1-4 represent samples of the input image and the result. Usually the result image does not have rigid boundaries of buildings, streets, and houses as the center image.

Fig. 5-1-4. Output Images in Segmentation and Interpretation Stages
The second stage is interpretation. Interpretation is to apply functions to manipulate the segmented (labeled) images. Unlike standard image processing, the system of building extraction from satellite images cannot apply global strategies to obtain geometric results by applying one specific mathematical function, because the satellite and aerial images have a lot of different kinds of regions. For example, even if the color of one segmented region is darker than the other regions, it cannot be concluded that the region represents shadow. Therefore, the spatial and attributive relations among the segmented regions should be considered as the key factors in determining what the region means. As a result of using the relations, the segmented regions can be interpreted and then refined by applying functions for fixing their attributes such as shape, direction, etc as the right image in Fig. 5-1-4. The functions are predefined, and the cases of applying the functions to the regions are inferred from the relations, which the system learns. Fig. 5-1-5 shows the diagram of these processes.

Fig. 5-1-5. Diagram of the Processes in Segmentation and Interpretation Stages
V-1-3. Neuro-Fuzzy System

Traditional geometrical techniques of monocular building extraction are limited, because they cannot flexibly adapt to the various situations. Though some projects, (Chellappa, 1994) (Hsieh, 1996) (Mckeown, 1990), have tried to apply expert systems in order to implement the adapting ability with a collection of $\langle \text{If } \sim, \text{ then } \sim \rangle$ rules, this approach is a dead end. The reason being that it is possible for the system developers to adapt the systems by adding rules in the expert systems, but it is impossible for the system's users to adapt the system.

On the other hand, the soft computing approach has the advantage of adapting the systems by the system's users, and several techniques have been proposed and developed since the beginning of the 1990s. In this research, the model of fuzzy multiple layers perceptron (MLP) (Pal, 1992) is used and tested, because it has a good capacity for learning the relations between inputs and outputs and generating fuzzy rules from the networks. The model has three kinds of layers, which are one input layer, one output layer and several hidden layers as shown in Fig. 5-1-6. The backpropagation algorithm is used for learning. The priority of this model is to generate fuzzy rules in linguistic forms from the acquired networks, which most neural networks cannot do. For example, the system can generate the rules such as "If $F_1$ is very low and $F_2$ is high, then the result is very likely Class2," where $F_1$ and $F_2$ are features, and Class2 is one of the categories in classification.
V-1-4. Training & Refining Processes

In developing neural network systems and fuzzy neural network systems, two processes are required. One is a training process to supervise sample data and make the system smart. A set of inputs and the expected outputs corresponding to the inputs must be trained into the networks. The neural networks and fuzzy neural networks learn the relations by updating the weight matrices. At the initial stage, the weight matrices have random variables. After training sample data, the weight matrices can return the most reasonable results from the given inputs as shown in Fig. 5-1-7.

Fig. 5-1-6. Network Structure of Fuzzy Multiple Layers Perceptron
However, if the conditions are changed, the weights should be arranged to adapt to the changed situations. Therefore, an additional process is necessary. It is a refining process. In this process, the networks adapt the weights slightly by changing them from the existing values. Many adapting systems such as LVQ, ART, ARTMAP, etc have been developed. However, the weakness of those systems is the difficulty in knowing how the systems will adapt. There is a need to know what kind of data is necessary to become more adaptive since the relations between inputs and outputs are represented as a set of values whose meaning the system's users can not understand. The fuzzy MLP model, on the other hand, can generate fuzzy rules, which are a set of <If ~, then ~> rules with linguistic terms such as low, high, medium, very, more or less, etc. The system's users can observe what kind of rules the neural networks have and have not learned, and understand why the system makes mistakes by checking these generated rules as shown in Fig. 5-1-8.
V-1-5. Features

This research has three important features in creating 3D computer city models. One is that the system allows the users to label the expected outputs. Due to this allowance, the users can select the required information. For example, it is possible to select only small houses if the user trains the system with different labels discriminating between "small house" and "big building" in satellite images during the training process. In addition, a supervised learning algorithm is applied to the system because it has better ability in classification than unsupervised learning. Most of the applications in image segmentation with these techniques have acquired effective results.

The second feature is the definition of design knowledge as the relations between the segmented areas and the functions applied to them. The segmented output images are only pixel-based information. It is necessary to apply mathematical functions in order to turn
sets of pixels into objects. Example functions are generating rigid squares from ill-defined shapes, and defining the height of buildings from the length and direction of the neighboring shadow area. Therefore, several functions should be implemented in advance.

In addition, the parameters of the functions should be degrees of values or linguistic terms that apply fuzzy MLP. Here, also, is the need for training processes, which make the system learn what kind of functions should be used and to what degree the functions should be applied in the given situations. For example, after training in the relations, the system can recognize what the house area is and what kind of functions should be applied to the house areas by judging from the neighboring areas, the directions of edges, the center locations, etc. Though it is still very difficult to supervise the relations effectively, this approach is considered to have more ability to adapt flexibly than preexisting expert systems. It is also able to integrate with expert systems of the future.

The reason why the technology of soft computing is applied is that the system can adapt to the user's requiring systems gradually, while the traditional expert systems are a dead end. This research focuses on developing systems not for solving specific tasks but for adapting to the user's desires.

The last feature is to generate fuzzy rules automatically from the trained fuzzy MLP. It is difficult to know how much the systems have learned from the array of values in the weight matrices in traditional neural network systems. On the other hand, fuzzy MLP can generate <If ~, then ~> rules in linguistic terms from which users can understand how much the
systems have learned and what kinds of information is lacking. Therefore, it is easier to refine the weight matrices than by using non-fuzzy neural networks.

Though this research can be viewed as an application test of fuzzy MLP to monocular building extraction, it is important to apply and evaluate the most advanced techniques of adaptive ability. This research can also be categorized into a new study of developing inference engines in monocular building extraction.

V-1-6. Summary

- The purpose of this research is to develop a new system to create 3D computer city models.
- Fuzzy multiple layers perceptron (MLP) with the backpropagation algorithm is used as the methodology.
- The approach is to develop a neuro-fuzzy system that allows the system's users to label the pixels in their required manners to generate fuzzy rules in linguistic terms.
- Labeling makes it easy to select users required objects in whole satellite images.
- Generating fuzzy rules automatically makes it easy for the users to refine the weight matrices in the system.
- The system can also learn the relations between the segmented areas and their applied functions. This ability makes it possible to generate 3D city models with linguistic terms.
V-2. Inside the System

In this project, two individual fuzzy MLP networks are used for creating 3D computer city models from satellite images. One is for the segmentation stage, and the other is for the interpretation stage.

For the segmentation stage, the input data for fuzzy MLP is assumed to be a set of attributes of pixels in images such as the R-value, G-value, B-value, intensity, 3x3 neighbors mean, etc. The output data of fuzzy MLP is represented as a set of class membership values. The user defines the number of classes and the labels of the classes.

For the interpretation stage, on the other hand, the input data for fuzzy MLP is assumed as a set of attributes of the segmented regions in the segmentation stage. The attributes are the area, the height, the width, the distance to the next region, the center position, etc. The output data of fuzzy MLP is represented as a set of membership values for applied functions. The functions to manipulate the segmented region should be predefined. The user trains the system by choosing one expected function corresponding to each region in order to refine the shapes and create 3D models.

V-3. Techniques for 3D Models

Solid Modeling

In the process of generating 3D computer city models, this program applies the techniques of solid modeling in which each 3D object is not treated as a set of faces but as a solid
object. The benefits of using solid modeling are that it is easy to implement boolean operations for 3D objects such as combining, separating, and extracting, and to change the attributes of objects such as types of lines, surfaces, and corners. Moreover, it can reduce the calculation of hidden surface rendering not by sorting each pixel in z-buffer rendering but by simply sorting objects according to the distance between the center point of each object and camera position. This program uses one of the most popular solid modeling, winged-edge data structures developed by Baumgart (Baumgart, 1972). In the algorithm, each object has a list of points, edges, and loops, and each loop has a list of edges, and each edge has start and end points. Specifically, each edge has only two wings, which are end-left-edge in the leftloop and start-right-edge in the rightloop as shown in Fig. 5-3-1. As a result, it is possible for the object data to prevent the memory space from rising, though the accessibility to the data becomes more or less inefficient.

![Fig. 5-3-1. Winged-edge Data Structure](image)

**Texture Mapping**

Texture mapping is one of the most popular techniques in computer graphics. It is the rendering technique that maps or projects 2D pictures onto faces of 3D objects. This
program can apply the technique because texture mapped 3D computer city models are more helpful in evaluating the results than surface rendered models alone. Though one texture is mapped on each face of 3D objects in the standard techniques of texture mapping, this program can map multiple textures on each face. The reason why multiple textures are applied is to make it easy for the system users to change, add, or delete textures of buildings and houses. The system users can use the texture mapping after generating 3D computer city models.

![Multiple Texture Mapping](image)

Fig. 5-3-2. Multiple Texture Mapping

**V-4. Techniques of Image Processing**

**RGB value**

RGB is one digital image format that uses a set of red, green, and blue values for one pixel. Standard functions to get each value as an integer are built into the graphic libraries of most programming languages. The following are examples in JAVA.

```java
int red_value=image.getRed();
```
int green_value=image.getGreen();
int blue_value=image.getBlue();

**Intensity/Gray value**

The intensity or the gray value represents the value of combining RGB value calculated above. In this project, the average of R, G, and B values is applied. The following is an example in JAVA.

int intensity=(red_value+green_value+blue_value)/3;

**Convolution**

Convolution is one of the most famous techniques in image processing. It is to apply filters to detect edges, calculate the averages of neighborhoods, and create contours. In this project, the technique is applied to get the average of 3x3 and 5x5 neighborhoods of images. Fig. 5-4-1 shows the process of convolution.

\[
F_{ij} = x0 \cdot P_{(i-j+1)} + x1 \cdot P_{(i-j)} + x2 \cdot P_{(i-j+1)} + x3 \cdot P_{(i+1-j-1)} + x4 \cdot P_{(i+1-j)} + x5 \cdot P_{(i+1-j+1)} + x6 \cdot P_{(i-j+1)} + x7 \cdot P_{(i-j+1)} + x8 \cdot P_{(i+1-j+1)}
\]

Fig. 5-4-1. Process of Convolution
Width and Height

Width and height values of regions are morphological features of segmented regions. In this project, the maximum length values of width and height are applied as shown in Fig. 5-4-2.

Area

The areas of segmented regions are the number of a set of pixels with the same values. Getting the areas of regions is one of the most basic techniques in image processing. The following shows the example function to get the area in JAVA.

```java
int getSegArea(int xPos, int yPos){ // (xPos, yPos)=pixel position in images
    int id=Fij[(yPos+1)*(width+2)+(xPos+1)]; // Fij=array of pixel values in image
    int area=0;
    if(id!=0){
        for(int i=1; i<(height+1); i++) // height = height of image
            for(int j=1; j<(width+1); j++){ // width = width of image
                if(Fij[i*(width+2)+j]==id){
                    area++;
                }
            }
    }
    return area;
}
```
There are mainly three kinds of boundary searching, which are edge-based, corner-based, and pixel-based searching. This program applied corner-based searching because the searching algorithm can get the boundary correctly with one way tracking. The other two algorithms need both clockwise and counterclockwise tracking. Corner-based searching requires preparing 16 conditions of 2x2 binary pixels beforehand. Each condition has one tracking direction among top, bottom, right, and left. The conditions and their tracking directions are shown in Fig. 5-4-3.

The following shows the sample corner-based searching code to get a boundary polygon in Java.

```java
Polygon getSegPolygon(int regionID) { //** regionID = pixel value of the region
    Polygon p = new Polygon(); //** Output Polygon
    boolean firstPicked = true; //** Flag to get first point of region
    int count = 0;
    Point startP = new Point(0, 0);
    int dir = -1; //** Tracking Direction
```
//** Get the first point of the region in the image
while(firstPicked && count<((width+2)*(height+1))){
    if (count%(width+2)==(width+1) ) {count++;} 
    else{
        int tempX=count%(width+2);
        int tempY=count/(width+2);
        int[] tempAij=new int[4];
        tempAij[0]=Fij[count]; //** Fij=pixel values
        tempAij[1]=Fij[count+1];
        tempAij[2]=Fij[count+width+2];
        tempAij[3]=Fij[count+width+2+1];
        //** If matching one of 16 conditions
        if((tempAij[0]!=regionID)&&(tempAij[1]==regionID )||
                (tempAij[2]==regionID )||
                (tempAij[3]==regionID )){
            startP=new Point(tempX, tempY); //** Store the first position
            dir=getAijDirection(tempAij, regionID, -1);
            firstPicked=false;
        }
        count++;}
    } //** Start Tracking counterclockwise
    if(!firstPicked){
        p.addPoint(startP.x, startP.y);
        Point currentP=new Point(startP.x, startP.y);
        Point nextP=new Point(startP.x, startP.y);
        int cond; //** Get the second position
        if(dir==0) nextP=new Point(currentP.x+1, currentP.y);
        else if(dir==1)nextP=new Point(currentP.x, currentP.y-1);
        else if(dir==2)nextP=new Point(currentP.x-1, currentP.y);
        else if(dir==3)nextP=new Point(currentP.x, currentP.y+1);
        while((nextP.x != startP.x || nextP.y != startP.y) && dir !=-1){
            int predir=dir; //** Previous tracking direction
            currentP=new Point(nextP);
            int[] Aij=getAij(currentP.x, currentP.y); //**Create 2x2 neighborhoods
            cond=getAijCondition(Aij, regionID); //**Get the index of 2x2 conditions
            if(cond==1 || cond==2 || cond==4 || cond==5 || cond==7 || cond==8 || cond==10 ||
                    cond==11 || cond==13 || cond==14){
                p.addPoint(currentP.x, currentP.y);
            }
            dir=getAijDirection(Aij,regionID,predir); //**Get the tracking direction
        }
    }
/** Get the next position
if(dir==0) nextP=new Point(currentP.x+1, currentP.y);
else if(dir==1) nextP=new Point(currentP.x, currentP.y-1);
else if(dir==2) nextP=new Point(currentP.x-1, currentP.y);
else if(dir==3) nextP=new Point(currentP.x, currentP.y+1);
}
return p;
}

V-5. Techniques of Fuzzy MLP

Memberships

This program applied fuzzy multiple layer perceptron (MLP), which is one of the
neuro-fuzzy systems. Both the inputs and outputs are represented as a set of membership
values.

In this project, the input values are the sets of features of each pixel such as the red, the
green, the blue, etc. At first, the minimum, the maximum, and the mean values of each
feature are calculated from sample data. Secondly, the means of high and low value sets are
calculated. The mean of high value sets is the mean of the values that are more than the
mean value of each feature. The mean of low value sets is the mean of the values that are
less than the mean value of each feature. Then, the membership values for each feature are
calculated by $\pi$ functions defined as:

$$
\pi(r; c, \lambda) = \begin{cases} 
2 \left( 1 - \frac{\|r - c\|}{\lambda} \right)^2, & \text{for } \frac{\lambda}{2} \leq \|r - c\| \leq \lambda \\
1 - 2 \left( \frac{\|r - c\|}{\lambda} \right)^2, & \text{for } 0 \leq \|r - c\| \leq \frac{\lambda}{2} \\
0, & \text{otherwise}
\end{cases}
$$
where \( r, c \), and \( \lambda \) represent the input feature original value, the mean value, and the bandwidth (the distance between the crossover points of \( \pi \)) respectively. The \( \lambda \)s for medium, high, and low memberships are defined as:

\[
\begin{align*}
C_{med}(F_j) &= \text{mean of all samples in feature } j \\
C_{high}(F_j) &= \text{mean of high sample sets in feature } j \\
C_{low}(F_j) &= \text{mean of low sample sets in feature } j \\
\lambda_{low}(F_j) &= 2(C_{med}(F_j) - C_{low}(F_j)) \\
\lambda_{high}(F_j) &= 2(C_{high}(F_j) - C_{med}(F_j)) \\
\lambda_{med}(F_j) &= \frac{\lambda_{low}(F_j) * (F_j_{max} - C_{med}(F_j)) + \lambda_{high}(F_j) * (C_{med}(F_j) - F_j_{min})}{F_j_{max} - F_j_{min}}
\end{align*}
\]

The output membership values represent the degree of how close to the mean vector of each class they are. At first, the vector of the mean and standard deviation for each class are calculated with sample vectors that are expected to be categorized into the class. The membership value for the class is calculated as:

\[
\mu_k(F_i) = \frac{1}{1 + \left(\frac{Z_{ik}}{f_d}\right)^{fe}}
\]

where \( fd \) and \( fe \) are the denominational and exponential fuzzy generators that control the amount of fuzziness. In this project, initially \( fd \) and \( fe \) are set as 5 and 1 respectively. \( Z_{ik} \) is the weighted distance of training pattern \( F_i \) for the k-th class and defined as:

\[
Z_{ik} = \sqrt{\sum_{j=1}^{2} \left(\frac{F_{ij} - M_{kj}}{V_{kj}}\right)^2}
\]

where \( M_{kj} \) and \( V_{kj} \) are the mean and standard deviation of \( j \)-th feature among the sample.
sets categorized into the $k$-th class, and $F_{ij}$ is the original value of feature $j$ of $i$-th sample input.

In this project, since the expected output should be categorized into only one class, the other membership values are zeros. For example, when the expected output should be categorized into the class 2 among 5 classes, the calculated vector $(0.6, 0.78, 0.2, 0.1, 0.2)$ is transformed into $(0, 0.78, 0, 0, 0)$.

**Mean Square Error and Cross-Entropy**

In order to optimize the weights of neural networks, this program applied the mean square error ($Mse$) and cross-entropy ($Ce$). The mean square error is the average of the square distance between the output vector from the fuzzy MLP and the expected output vector, and defined as:

$$Mse = \left( \sum_{F \in \text{SampleSet}} \sum_{j=1}^{l} (d_j - y_{j}^{H})^2 \right) / (l * \text{samplesNumber}),$$

where $sampleNumber$ is the number of samples, $d_j$ and $y_{j}^{H}$ are the expected value and the output value of $j$-th node in the output layer $H$ respectively.

The cross-entropy is the entropy between the output vector from the fuzzy MLP and the expected output vector, and defined as:

$$Ce = \left( \sum_{F \in \text{SampleSet}} \sum_{j=1}^{l} \left( -d_j * \ln y_{j}^{H} - (1 - d_j) * \ln(1 - y_{j}^{H}) \right) \right) / (\ln 2 * l * \text{sampleNumber}).$$
These two errors are used in order to update the momentum values in the backpropagation learning process. During backpropagation, the index of the momentum array increases when the mean square error or cross-entropy is less than 0.0001 after 10 training epochs. One training epoch means one learning process of all sample data.

**Momentum in Backpropagation Learning**

In this project, the momentum factor is applied in order to optimize the speed of the backpropagation learning processes. The standard backpropagation algorithm requires many epochs to make the weight train well because the factor in updating the weights is constant. Therefore, ill data strongly effects the well-trained weights. In order to reduce the effects of ill data, the momentum factor is developed (Jacobs, 1988). In short, the factor of updating the weights decreases gradually as the weights are trained well. The factor is called momentum and used in updating weights as: $\Delta w_{jk} (t+1) = \partial \delta z_j + u \Delta w_{jk} (t)$, where $u$ is the momentum factor, and the other nomenclatures follow the rules in section II-5-6-2.

In this project, an array of 10 momentum values is applied, and the index of the momentum value applied for backpropagation increases when the mean square error or the cross-entropy is less than the specified threshold. The backpropagation learning stops when the index of the momentum value is updated to 10.
Chapter VI

Program

This chapter explains the details of the program developed for this research. The first section explains what Java is. The second section explains the tutorial of this program. The last section explains the input and output file format.

VI-1. Java

This project uses Java as a programming language. Java was introduced by Sun Microsystems in 1995, and has rapidly become one of the most popular programming languages. The main reason why Java is used in this project is that Java is a cross-platform language. In other words, this program can run on PC, Macintosh, and Unix machines if they have the Java virtual machine (VM) installed. The cross-platform feature is very helpful for researchers because they do not have to pay a lot of attention in setting up libraries and compiling such as with C++ or C languages. Moreover, since network programming through the Internet and server/client systems is easier in Java than in any other language, this program can be collaborated through network computer systems in the future.

This project is programmed using JDK 1.2 and can run on any computer supporting a later version than Java 2. In order to run this program, it is necessary only to save the whole files in one directory and type "java Test" in the directory. "Test.class" is the main Java complied

This section explains the tutorial of the program. The first section explains the components of the program interface. The second section explains the commands of the pull down menus in this program. The third section gives a short tutorial for a simple case.

VI-2-1. Components

The menu bar, which is shown on the top bar in the main program window, has one menu "Start Programs". The menu has two submenus, "3D City Model" and "Fuzzy MLP". The "Fuzzy MLP" is a command to open the program window for fuzzy MLP, and "3D City Model" is to show generated 3D computer city models from the output data created in the fuzzy MLP program as shown in Fig. 6-2-1.

![Fig. 6-2-1. Main Program Interface](image-url)
The fuzzy MLP program has three panels, an **image panel**, a **data panel**, and an **output information panel**. The image panel is located in the left top, the data panel in the right top, and the output information panel in the bottom.

1. **Image Panel**

The image panel has three tab panels, an **input image panel**, a **segmented image panel**, and an **interpreted image panel**. The input image panel shows the original satellite image loaded. The segmented image panel shows the image categorized into some classes with fuzzy MLP. Finally the interpreted image panel shows the image with some functions applied to the image in the segmented image panel with fuzzy MLP.

2. **Data Panel**

The data panel in the main program has three tab panels, an **MLP condition panel**, an **SOM panel**, and a **setup information panel**. The MLP condition panel shows the weight conditions of fuzzy MLP. Selecting the command of "Show MLP Condition" in the menu bar, this panel shows the current conditions with white boxes and several kinds of lines. The white boxes represent the nodes/neurons in the input layer, the hidden layers, and the output layer from left to right. The lines represent the weights of the networks. If the weight between two nodes/neurons is between 0.75 to 1, then the weight is described as a red thick line. If it is between 0.5 to 0.75, then as a thick black line. If it is between 0.25 to 0.5, then as a thin black line. If it is under 0.25, no line is between the two nodes/neurons. The SOM
panel in the data panel shows the topological mapping of samples by using the technique of Kohonen's self-organizing map. The setup information panel shows the parameters for fuzzy MLP, and the system user can change them by typing.

**The setup information panel** has three categories, **segmentation parameters**, **interpretation parameters**, and **MLP parameters**. Each category is shown as a labeled boundary box.

**2-1. Segmentation Parameters**

In the category of segmentation parameters, the **features for segmentation** and the **names of classes for segmentation** are defined. The features for segmentation represent the pixel features applied for segmentation such as red, green, and blue values. The system user inputs the identity numbers of features that he/she wants to apply. The identity number of red-value, green-value, blue-value, gray-value, the 3x3 average-value, and the 5x5 average-value are 0, 1, 2, 3, 4, and 5 respectively. The default is set as 0, 1, 2, and 3. The total number of features applied here determines the number of nodes/neurons in the input layer of fuzzy MLP. The second parameter in the category of segmentation parameters, the names of class, represent the labels that the system user wants to assign to categories such as "House", "Shadow" and "Street". He/she input the text of names. The number of words applied here determines the number of nodes/neurons in the output layer of fuzzy MLP.
2-2. Interpretation Parameters

The second category of the setup information panel, interpretation parameters, has three input fields, which are **class identities for interpretation**, **features for interpretation**, and **functions for interpretation**. The class identities for interpretation represent the classes applied in the interpretation stage. The system user inputs the identity numbers of labels defined in the segmentation stage. For example, if the user wants to select the segmented regions labeled as a "House" defined as the first class in segmentation parameters, he/she must input 0. The default is set as 0.

The second input field, **features for interpretation**, represents the regional features applied for interpretation such as area, width, height, and ratio of width and height. The system user inputs the identity numbers of features that he/she wants to apply. The identity number of the area, the width, the height, the ratio, the maximum axis, and the minimum axis are 0, 1, 2, 3, 4, and 5 respectively. The default is set as 0, 1, 2, and 3.

The last input field, **functions for interpretation**, represents the functions that the system user wants to use in the interpretation stage. The user inputs the identity numbers of functions. The functions are predefined. In this case, only three functions, which are the function of deleting region, creating rectangle region, and creating polygonal region, are prepared. The default is set as 0, 1, and 2, which means all predefined functions are applied.
2-3. MLP Parameters

The last category of the setup information panel, MLP parameters, has ten input fields. They are the number of hidden layers, nodes in hidden layers, learning rate, momentum rate, denominational fuzzy generator, exponential fuzzy generator, delta checking interval, maximum learning repeats, and minimum delta for errors.

The input field, **hidden layers number**, determines the number of hidden layers of fuzzy MLP. The default is set to one hidden layer.

The second input field, **nodes in hidden layers**, determines the number of nodes/neurons in hidden layers as a set of integers. For example, if there are three hidden layers, the system user must input three numbers of nodes for each hidden layer such as (10 6 5), which means the first hidden layer, the second hidden layer, and the last hidden layer has 10, 6, and 5 nodes respectively. The default is set as 10 nodes for one hidden layer defined in the field of hidden layers number.

The third and fourth input fields, **learning rates** and **momentum rates**, define the learning rates and momentum rates for backpropagation of fuzzy MLP. The system user can define the number of rates and the values of them. The number of the learning rate and momentum rate should be the same. The defaults are set as (2.0 1.0 0.5 0.3 0.1 0.05 0.01 0.005 0.001) for learning rate and (0.9 0.5 0.5 0.5 0.5 0.5 1.0) for momentum rate.
The fifth and sixth input fields, **denominational fuzzy generator** and **exponential fuzzy generator**, determine the parameters of the fuzzy generator function described in section 5-5. The defaults are set as 5.0 and 1.0 for the denominational fuzzy generator and exponential fuzzy generator respectively.

The eighth input field, **delta checking interval**, determines the interval to check the mean square error and cross-entropy between outputs of MLP and expected outputs. The default is set as 10.

The ninth input field, **maximum learning repeat**, determines the maximum repeat number for backpropagation of fuzzy MLP. The default is set as 1000 times.

The last input field, **minimum delta for errors**, determines the minimum delta value for mean square error and cross-entropy between the outputs from fuzzy MLP and expected outputs. If either mean square errors or cross-entropy errors are less than this delta value during backpropagation, then the learning rate and momentum rate defined above are updated. The default is set as 0.0001. This value should be set between 0 to 0.01.

3. Output Information Panel

The output information panel in the main program is the bottom panel and has four tab panels, which are **sample information panel**, **membership information panel**, **fuzzy MLP information panel**, and **fuzzy rule panel**. The sample information panel shows the
values of features for each sample pixel or each sample region that the system user picked. After the user picks each sample, one text line with the feature values of the picked sample is added to this panel. The membership information panel shows the statistical information about all samples after the user selects the command of "Show Membership Info" from the pull down menu. The fuzzy MLP information panel shows the weights of all fuzzy MLP after the user loads a file or selects the command of "Create New MLP" from the pull down menu. The fuzzy rule panel shows a set of fuzzy rules that the fuzzy MLP obtains after the user selects the command of "Show Fuzzy Rules" from the pull down menu.

Fig. 6-2-2. Interface of Fuzzy MLP Program

VI-2-2. Pull Down Menu

This section explains the commands in the pull down menu.

Main

This program has a desktop frame, and the frame has a main menu bar at the top. The menu
bar has one pull down menu named "Start Program". The menu has two sub menus, "3D City Model", and "Fuzzy MLP". The 3D City Model is a command to open a viewer frame, which shows 3D computer city models created in the main program. The other menu, "Fuzzy MLP", is the command to open a fuzzy MLP frame.

**File**

The fuzzy MLP frame also has a menu bar at the top. The bar has four menus, which are "File", "Run", "Test", and "View". In the "File" menu, there are three sub menus, which are "Open", "Save", and "Exit". In the "Open" menu there are four commands, "Open Image File", "Open Samples (Segmentation)", "Open Samples (Interpretation)", and "Open Fuzzy MLP". "Open Image File" is a command to load any gif/jpg formatted image file, "Open Sample(Segmentation)" is to load sampling data file for segmentation, "Open Sample(Interpretation)" is to load sampling data file for interpretation, and "Open Fuzzy MLP" is to load the weight data of fuzzy MLP. In the "Save" menu there are also four commands, "Save Samples (Segmentation)" to save sampling data for segmentation, "Save Samples (Interpretation)" to save sampling data for interpretation, "Save Fuzzy MLP" to save the weight data of fuzzy MLP that is active in the program, and "Save Output Image" to save a gif formatted image file that is the latest output image in the program. The "Exit" menu is a command to close the fuzzy MLP frame.

**Run**

In the "Run" menu, there are 5 sub menus, which are "Create Memberships", "Create New MLP", "Supervise MLP", "Get Output Image", and "Update Parameters". The
"Create Memberships" menu is a command to calculate membership values of samples for each feature defined in the setup information panel and to show the values in the sample information panel. The "Create New MLP" menu is a command to create a new fuzzy MLP for either segmentation or interpretation and to show all weights in the fuzzy MLP information panel. The "Supervise MLP" menu is a command to make the MLP weights learn samples with the technique of backpropagation. The "Get Output Image" menu is a command to calculate the output image from the input image with the supervised MLP and show the output image in the interpretation panel. The "Update Parameters" menu is to update the parameters of the setup information panel when the system user changes them.

Test

In the "Test" menu, there are two sub menus, which are "Get Accuracy of Samples" and "Get Fuzzy Rules". The "Get Accuracy of Samples" menu is a command to show how many percents of the outputs calculated by the fuzzy MLP agree with the expected outputs. The "Get Fuzzy Rules" menu is a command to generate the fuzzy rules from the current fuzzy MLP and show them in the fuzzy rule panel.

View

In the "View" menu, there are two sub menus, which are "Show Current MLP Weight" and "Show SOM Data". The "Show Current MLP Weight" menu is a command to show the structure and conditions of the current fuzzy MLP in the MLP condition panel. The "Show SOM Data" menu is a command to show the topological mapping conditions in the SOM panel.
VI-2-3 Short Tutorial

This section presents a simple tutorial on how to use this program.

Since this application is programmed in Java, the user has only to set up the Java Running Environment on his/her computer. After setting it up, typing "java Test" will make this program start.

The user can see the desktop frame with no inner frame. If he/she chooses a command, "Fuzzy MLP", in the menu "Start Program" from the menu bar at the top of the frame, an inner panel, the fuzzy MLP frame, comes up inside the desktop frame. The following are the processes of the segmentation stage.

1) Choose the command "File" -> "Open" -> "Open Image File", and select any gif/jpg formatted image file, then the image file can be seen in the image panel at the left top of frame.
Step 1. Segmentation Stage

2) Click the mouse at any point inside the image. A red-cross mark comes up at the pressed point.

3) Push one button corresponding to the class that the picked pixel should be classified as, then the red-cross mark changes color from red to the corresponding class color. For example, if the user pushes "House", then the red cross mark changes to a blue cross mark. The information of the picked pixel is shown in the sampling information panel at the bottom of the frame.

4) Continue this sampling process.

5) After picking enough pixels, choose the command, "Run" -> "Create Memberships", then the membership values are shown in the membership information panel at the bottom of the frame.

6) Choose the command, "Run" -> "Create New MLP", then new fuzzy MLP with one hidden layer are created and all weights are shown in the fuzzy MLP information panel at the bottom of the frame.

7) Choose the command, "Run" -> "Supervise MLP", then the fuzzy MLP is trained by sample data.

8) Choose the command, "Test" -> "Get Accuracy", then the percentage of how the fuzzy MLP can get correct outputs is shown in the fuzzy MLP information panel at the bottom of the frame.

9) If the percentage calculated in 8) is lower than expectation, choose the command, "Test" -> "Get Fuzzy Rules". Then, the fuzzy rules that the fuzzy MLP has already learned are
shown in the fuzzy rule panel at the bottom of the frame. Otherwise, go to 11).

10) Observe the fuzzy rules generated in 9). Go to 2) and add more samples that the user expects but the system has not learned or if the system generates wrong rules.

**Step2. Interpretation Stage**

11) Choose the command, "Run" -> "Get Output Image". The system then calculates all pixels in the input image and categorizes them. The classified image is shown in the segmented image panel, and the segmented image of the first category class is shown in the interpreted image panel at the left top of the frame.

12) The segmented image has two colors, black and orange. The orange parts are the results classified into the first category. Click the mouse inside any orange regions, then a red crossmark comes up in the image.

13) Push one button corresponding to the function that the system should apply to the selected region in 12), then the red cross mark changes color from red to white if the picked point is inside the region.

14) Continue this sampling process.

15) After picking enough regions, choose the command, "Run" -> "Create Memberships", then the membership values are shown in the membership information panel at the bottom of the frame.

16) Choose the command, "Run" -> "Create New MLP". A new fuzzy MLP with one hidden layer is created and all weights are shown in the fuzzy MLP information panel at the bottom of the frame.
17) Choose the command, "Run" -> "Supervise MLP". The fuzzy MLP is trained by sample data.

18) Choose the command, "Test" -> "Get Accuracy". The percentage of how the fuzzy MLP is getting correct outputs is shown in the fuzzy MLP information panel at the bottom of the frame.

19) If the percentage calculated in 18) is lower than expectation, choose the command, "Test" ->"Get Fuzzy Rules". Then, the fuzzy rules that the fuzzy MLP has already learned are shown in the fuzzy rule panel at the bottom of the frame. Otherwise, go to 21).

20) Observe the fuzzy rules generated in 19). Go to 13) and add more samples that the user expects but the system has not learned or if it has generated wrong rules for.

21) Choose the command, "Run" -> "Get Output Image", then the system calculates all regions in the interpreted image and applies corresponding functions to them. Then, the regions are transformed in the same panel.

22) Continue the process from 12) to 21) until the regions in the output image are transformed as well as the user expects.

Step3. Generating 3D Computer City Model

23) Choose the command, "Start Program" ->"3D City Model" in the main desktop frame, then a new inner frame comes up inside the desktop. The 3D computer city models are shown in the frame. Dragging and moving the mouse inside the panel, the generated computer models rotate and the user can observe the models from any points of view.
VI-3. I/O format

This section explains the file format for saving and loading the data files of samples and weights of fuzzy MLP. Both files are based on a text format. Any text files that agree with the following structure can apply to this program as a simple fuzzy MLP system.

Sample Data Format

The sample data file has two parts: a header part and a content part. The header part has three lines: the number of samples, features, and classes. The number of samples represents the total number of samples applied to this program and should be written as "sampleNum XX" in the first line of the file. The number of features is the input feature number for fuzzy MLP and should be written as "featureNum XY" in the second line of the file. The number of the class is the number of output category classes and should be written as "classNum YY" in the third line. The content part of the data file should follow these three lines. Each sample data should be written in one line. The line starts with the expected class identity number and follows the value of each feature. For example, the line of "3 1.2 2.5 3.4" means that class3 is the expected class identity and the values of feature1, feature2, and feature3 are 1.2, 2.5, and 3.4 respectively. If the number of samples and the total lines of contents are not the same, the system cannot load the file. The following is a simple example:

```
sampleNum 6
featureNum 4
classNum 3
0  244.0  249.0  232.0  244.0
1  154.0  156.0  153.0  154.0
```
Fuzzy MLP Format

The fuzzy MLP data file consists of two parts: a header part and a content part. The header part has two text lines: The first line represents the number of weight matrices in the fuzzy MLP, and the second line represents a set of the node numbers in each layer. For example, when the fuzzy MLP has four layers with 4, 5, 10, and 6 nodes in the input, the first hidden, the second hidden, and the output layers respectively, the number of weight matrices is 3, and the set of node numbers are a string of "4 5 10 6". In this case, the system recognizes that the fuzzy MLP has three weight matrices: 1) 4x5 matrix between the first and second layer, 2) 5x10 matrix between the second and third layer, and 3) 10x6 matrix between the third and forth layer. The lines of the header part should be "WeightNum 2" and "Nodes 4 5 10 6"

The content part of the data file consists of each set of weight matrix data. Each set of weight matrix data has one line of information about the matrix in the first line. The line represents the enumerated identity number, the input node number, and the output node number. For example, "Weight 0 Input 5 Output 6" means that the weight matrix represents the weights between the first layer with 5 nodes and the second layer with 6 nodes. If the fuzzy MLP applies bias nodes, the input nodes number is one more than the node number defined in the header part. After the line, the weight values are described in the following rules. One text line represents the weight values as a vector whose dimension is the number
of the input nodes. For example, the weight matrix is 5x6 dimension, there are 6 lines and each line has 5 weight values. The following is a simple example of the fuzzy MLP data file.

<table>
<thead>
<tr>
<th>WeightNum</th>
<th>Nodes</th>
<th>2</th>
<th>2</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Weight 0</td>
<td>Input</td>
<td>3</td>
<td>Output</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-0.3252</td>
<td>-0.1113</td>
<td>0.2052</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.5881</td>
<td>0.6993</td>
<td>0.8055</td>
</tr>
<tr>
<td>Weight 1</td>
<td>Input</td>
<td>3</td>
<td>Output</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.28041</td>
<td>0.5776</td>
<td>0.8198</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.16826</td>
<td>0.7121</td>
<td>0.7376</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.08543</td>
<td>0.3246</td>
<td>0.0097</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.80879</td>
<td>-1.362</td>
<td>1.1279</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-0.1129</td>
<td>-0.023</td>
<td>1.0688</td>
</tr>
</tbody>
</table>
Chapter VII
Case Studies

This chapter presents a set of case studies using the program developed in this research. In the first section, the results of a case study in which one simple painting image is analyzed are presented. In the second section, the system is applied to a real satellite image. These two sections primarily show the type of information the system can produce step by step. In the third section, the results of several experiments that vary system parameters are presented. The final section evaluates the system based on these experimental results. The final section focuses on evaluating the capabilities of the system as well as its shortcomings from the results and performance of the system.

VII-1. Simple Case

This section shows the results of a simple case. The input image representing the plan of a city was drawn manually using commercial photo-retouch software as shown in Fig. 7-1-1.
The size of the image is 300x300 pixels. The number of sample pixels was 30. Each sample pixel has a set of attributes, red, green, blue and gray as shown in Table 7-1-1. After the sampling process, the sampling set is calculated into a membership value set. The results are shown in Table 7-1-2 and 7-1-3.

![Fig. 7-1-1. Input Image for Simple Case](image)

<table>
<thead>
<tr>
<th>Sample ID#</th>
<th>Sample Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>House : (x,y)=(147, 143) : (r, g, b)=(153, 155, 62) : gray=153</td>
</tr>
<tr>
<td>2</td>
<td>House : (x,y)=(50, 52) : (r, g, b)=(153, 155, 62) : gray=153</td>
</tr>
<tr>
<td>3</td>
<td>House : (x,y)=(47, 135) : (r, g, b)=(123, 125, 48) : gray=123</td>
</tr>
<tr>
<td>4</td>
<td>House : (x,y)=(252, 187) : (r, g, b)=(152, 153, 66) : gray=152</td>
</tr>
<tr>
<td>5</td>
<td>House : (x,y)=(272, 48) : (r, g, b)=(145, 148, 70) : gray=145</td>
</tr>
<tr>
<td>6</td>
<td>House : (x,y)=(123, 236) : (r, g, b)=(148, 149, 59) : gray=148</td>
</tr>
<tr>
<td>7</td>
<td>Street : (x,y)=(176, 105) : (r, g, b)=(183, 183, 183) : gray=183</td>
</tr>
<tr>
<td>8</td>
<td>Street : (x,y)=(227, 105) : (r, g, b)=(184, 184, 184) : gray=184</td>
</tr>
<tr>
<td>9</td>
<td>Street : (x,y)=(55, 116) : (r, g, b)=(182, 183, 181) : gray=182</td>
</tr>
<tr>
<td>10</td>
<td>Street : (x,y)=(32, 96) : (r, g, b)=(137, 164, 103) : gray=137</td>
</tr>
<tr>
<td>11</td>
<td>Street : (x,y)=(268, 239) : (r, g, b)=(183, 183, 183) : gray=183</td>
</tr>
<tr>
<td>12</td>
<td>Street : (x,y)=(277, 103) : (r, g, b)=(184, 184, 184) : gray=184</td>
</tr>
<tr>
<td>13</td>
<td>Green : (x,y)=(190, 38) : (r, g, b)=(88, 146, 5) : gray=88</td>
</tr>
<tr>
<td>14</td>
<td>Green : (x,y)=(214, 82) : (r, g, b)=(105, 160, 18) : gray=105</td>
</tr>
<tr>
<td>15</td>
<td>Green : (x,y)=(219, 85) : (r, g, b)=(87, 148, 5) : gray=87</td>
</tr>
<tr>
<td>16</td>
<td>Green : (x,y)=(238, 86) : (r, g, b)=(79, 127, 16) : gray=79</td>
</tr>
<tr>
<td>17</td>
<td>Green : (x,y)=(194, 174) : (r, g, b)=(92, 156, 12) : gray=92</td>
</tr>
<tr>
<td>18</td>
<td>Green : (x,y)=(39, 197) : (r, g, b)=(88, 149, 6) : gray=88</td>
</tr>
<tr>
<td>19</td>
<td>Ground : (x,y)=(233, 161) : (r, g, b)=(232, 248, 168) : gray=232</td>
</tr>
<tr>
<td>20</td>
<td>Ground : (x,y)=(235, 172) : (r, g, b)=(232, 248, 168) : gray=232</td>
</tr>
<tr>
<td>21</td>
<td>Ground : (x,y)=(237, 234) : (r, g, b)=(232, 248, 168) : gray=232</td>
</tr>
<tr>
<td>22</td>
<td>Ground : (x,y)=(228, 59) : (r, g, b)=(232, 248, 168) : gray=232</td>
</tr>
<tr>
<td>23</td>
<td>Ground : (x,y)=(172, 67) : (r, g, b)=(232, 248, 168) : gray=232</td>
</tr>
</tbody>
</table>
The input membership values are represented as a vector of \((L_{\text{red}}, M_{\text{red}}, H_{\text{red}}, \ldots)\).
L_green, M_green, H_green, L_blue, M_blue, H_blue, L_gray, M_gray, H_gray). L, M, and H mean the membership values of low, medium, and high classes respectively. The output membership values are represented as a expected vector of (class0, class1, class2, class3, class4). In this case, the labels of the classes are "House", "Street", "Green", "Ground", and "Shadow". For example, if one pixel should be classified into class1, the membership value for class1 is calculated using equations in section 5-5 and the membership values for the other classes are zeros. In short, the expected output is represented as (0, 0.78, 0, 0, 0).

In this case, the neural networks have three layers, which are one input layer, one output layer, and one hidden layer. The numbers of nodes are 12, 10, and 5 for the input, the hidden, and the output layer respectively. The momentum parameter was updated when the mean square error or cross-entropy for each epoch was less than 0.001. One epoch means 30 samples in this case. The condition of neural networks after training samples is represented in the Fig. 7-1-2.
Fig. 7-1-2. Weights Condition of Fuzzy MLP for Segmentation

Fig. 7-1-3. Categorized Image
In Fig. 7-1-2, the thick lines represent the strong connections, and no lines between nodes means the weights are less than zero. The percentage of accuracy for the given samples was 100%. Fig. 7-1-3 shows the output-segmented image using the trained neural networks, and the fuzzy rules generated from the neural networks are represented in Table 7-1-4. Table 7-1-5 shows the fuzzy rules translated manually.

<table>
<thead>
<tr>
<th>IF Clause</th>
<th>THEN Clause</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF low F2, medium F1, medium F0, more or less high F3,</td>
<td>THEN likely Class 0.</td>
</tr>
<tr>
<td>IF low F2, more or less low F1, medium F0, medium F3,</td>
<td>THEN more or less likely Class 0.</td>
</tr>
<tr>
<td>IF low F2, medium F1, medium F0, medium F3,</td>
<td>THEN likely Class 0.</td>
</tr>
<tr>
<td>IF low F2, medium F1, medium F0, medium F3,</td>
<td>THEN very likely Class 0.</td>
</tr>
<tr>
<td>IF more or less high F1, high F0, high F3,</td>
<td>THEN likely Class 1.</td>
</tr>
<tr>
<td>IF medium F1, medium F0, medium F3,</td>
<td>THEN not likely Class 1.</td>
</tr>
<tr>
<td>IF medium F1, low F0, low F3,</td>
<td>THEN very likely Class 2.</td>
</tr>
<tr>
<td>IF medium F1, low F0, low F3, low F2,</td>
<td>THEN more or less likely Class 2.</td>
</tr>
<tr>
<td>IF low F0, low F3, more or less low F1,</td>
<td>THEN more or less likely Class 2.</td>
</tr>
<tr>
<td>IF medium F1, low F0, low F3, low F2,</td>
<td>THEN likely Class 2.</td>
</tr>
<tr>
<td>IF high F2, high F3,</td>
<td>THEN very likely Class 3.</td>
</tr>
<tr>
<td>IF medium F0, medium F3, medium F2,</td>
<td>THEN not likely Class 3.</td>
</tr>
<tr>
<td>IF more or less low F2,</td>
<td>THEN likely Class 4.</td>
</tr>
<tr>
<td>IF low F2,</td>
<td>THEN likely Class 4.</td>
</tr>
<tr>
<td>IF more or less low F2, low F1,</td>
<td>THEN likely Class 4.</td>
</tr>
</tbody>
</table>

Table 7-1-5. Translated Fuzzy Rules

The next interpretation stage applied 22 samples for the segmented image as shown in Fig. 7-1-4. Each sample region has a set of attributes, the area, the width, the height, and the ratio of the width and height as shown in Table 7-1-6.
Fig. 7-1-4. Segmented Image with "House" Label

After the sampling process, the sampling set is calculated into a membership value set. The result is shown in Table 7-1-7 and 7-1-8.

<table>
<thead>
<tr>
<th>Sample ID #</th>
<th>Samp</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(x,y)=(45, 48) : Area=2169 : Width=55 : Height=55 : Center=(36, 54)</td>
</tr>
<tr>
<td>2</td>
<td>(x,y)=(16, 149) : Area=2592 : Width=45 : Height=45 : Center=(24, 156)</td>
</tr>
<tr>
<td>3</td>
<td>(x,y)=(25, 247) : Area=1366 : Width=31 : Height=31 : Center=(23, 248)</td>
</tr>
<tr>
<td>4</td>
<td>(x,y)=(139, 142) : Area=1832 : Width=69 : Height=69 : Center=(141, 145)</td>
</tr>
<tr>
<td>5</td>
<td>(x,y)=(164, 219) : Area=224 : Width=24 : Height=24 : Center=(163, 223)</td>
</tr>
<tr>
<td>6</td>
<td>(x,y)=(127, 202) : Area=536 : Width=29 : Height=29 : Center=(122, 199)</td>
</tr>
<tr>
<td>7</td>
<td>(x,y)=(71, 13) : Area=141 : Width=17 : Height=17 : Center=(71, 12)</td>
</tr>
<tr>
<td>8</td>
<td>(x,y)=(14, 75) : Area=148 : Width=17 : Height=17 : Center=(143, 76)</td>
</tr>
<tr>
<td>9</td>
<td>(x,y)=(286, 251) : Area=154 : Width=16 : Height=16 : Center=(285, 251)</td>
</tr>
<tr>
<td>10</td>
<td>(x,y)=(205, 240) : Area=117 : Width=11 : Height=11 : Center=(206, 241)</td>
</tr>
<tr>
<td>11</td>
<td>(x,y)=(237, 137) : Area=181 : Width=14 : Height=14 : Center=(234, 132)</td>
</tr>
<tr>
<td>12</td>
<td>(x,y)=(239, 76) : Area=103 : Width=18 : Height=18 : Center=(238, 75)</td>
</tr>
<tr>
<td>13</td>
<td>(x,y)=(71, 239) : Area=143 : Width=15 : Height=15 : Center=(69, 238)</td>
</tr>
<tr>
<td>14</td>
<td>(x,y)=(47, 25) : Area=7 : Width=7 : Height=7 : Center=(47, 25)</td>
</tr>
<tr>
<td>15</td>
<td>(x,y)=(188, 8) : Area=5 : Width=4 : Height=4 : Center=(186, 7)</td>
</tr>
<tr>
<td>16</td>
<td>(x,y)=(278, 242) : Area=6 : Width=3 : Height=3 : Center=(278, 241)</td>
</tr>
<tr>
<td>17</td>
<td>(x,y)=(247, 249) : Area=6 : Width=6 : Height=6 : Center=(247, 249)</td>
</tr>
<tr>
<td>18</td>
<td>(x,y)=(132, 242) : Area=20 : Width=3 : Height=3 : Center=(131, 235)</td>
</tr>
<tr>
<td>19</td>
<td>(x,y)=(212, 282) : Area=9 : Width=3 : Height=3 : Center=(212, 281)</td>
</tr>
<tr>
<td>20</td>
<td>(x,y)=(106, 275) : Area=312 : Width=24 : Height=24 : Center=(116, 280)</td>
</tr>
<tr>
<td>21</td>
<td>(x,y)=(277, 42) : Area=2 : Width=1 : Height=1 : Center=(277, 42)</td>
</tr>
<tr>
<td>22</td>
<td>(x,y)=(227, 80) : Area=2 : Width=2 : Height=2 : Center=(226, 80)</td>
</tr>
</tbody>
</table>

Table 7-1-6. Sample Data for Interpretation
<table>
<thead>
<tr>
<th>Input Features</th>
<th>Mean of All Samples</th>
<th>Mean of Low Value Samples</th>
<th>Mean of High Value Samples</th>
<th>Minimum Value</th>
<th>Maximum Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Area</td>
<td>503.36</td>
<td>84.75</td>
<td>1619.67</td>
<td>2.0</td>
<td>2592.0</td>
</tr>
<tr>
<td>Width</td>
<td>18.81</td>
<td>9.13</td>
<td>39.57</td>
<td>1.0</td>
<td>69.0</td>
</tr>
<tr>
<td>Height</td>
<td>21.45</td>
<td>8.8</td>
<td>48.57</td>
<td>1.0</td>
<td>66.0</td>
</tr>
<tr>
<td>Ratio(Width/Height)</td>
<td>1.50</td>
<td>0.89</td>
<td>4.25</td>
<td>0.15</td>
<td>7.0</td>
</tr>
</tbody>
</table>

Table 7-1-7. Parameters for Memberships in Interpretation

<table>
<thead>
<tr>
<th>ID</th>
<th>Membership Values of Features for Input (F1_low, F1_med, F1_high, F2_low, F2_med, F2_high, ...)</th>
<th>Membership Values of Output (func1, func2, func3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.0  0.0  0.87  0.0  0.0  0.72  0.0  0.02  0.99  0.96  0.90  0.35</td>
<td>0.0  0.0  0.79</td>
</tr>
<tr>
<td>2</td>
<td>0.0  0.0  0.62  0.0  0.0  0.96  0.0  0.79  0.94  0.68  0.24</td>
<td>0.0  0.0  0.71</td>
</tr>
<tr>
<td>3</td>
<td>0.0  0.09  0.97  0.0  0.53  0.91  0.0  0.10  0.99  0.91  0.65  0.23</td>
<td>0.0  0.0  0.82</td>
</tr>
<tr>
<td>4</td>
<td>0.0  0.0  0.98  0.0  0.0  0.16  0.0  0.16  0.99  0.50  0.99  0.49</td>
<td>0.0  0.0  0.66</td>
</tr>
<tr>
<td>5</td>
<td>0.0  0.24  0.93  0.10  0.91  0.71  0.0  0.0  0.85  0.65  0.41  0.17</td>
<td>0.0  0.0  0.70</td>
</tr>
<tr>
<td>6</td>
<td>0.42  0.99  0.52  0.0  0.67  0.87  0.01  0.81  0.81  0.99  0.83  0.30</td>
<td>0.0  0.0  0.66</td>
</tr>
<tr>
<td>7</td>
<td>0.99  0.78  0.22  0.67  0.98  0.41  0.79  0.96  0.34  0.98  0.88  0.33</td>
<td>0.0  0.72  0.0</td>
</tr>
<tr>
<td>8</td>
<td>0.98  0.79  0.23  0.67  0.98  0.41  0.91  0.90  0.26  0.86  0.96  0.40</td>
<td>0.0  0.85  0.0</td>
</tr>
<tr>
<td>9</td>
<td>0.98  0.80  0.23  0.74  0.97  0.37  0.96  0.84  0.21  0.73  0.98  0.44</td>
<td>0.0  0.76  0.0</td>
</tr>
<tr>
<td>10</td>
<td>0.99  0.75  0.21  0.98  0.80  0.19  0.96  0.84  0.21  0.99  0.83  0.30</td>
<td>0.0  0.65  0.0</td>
</tr>
<tr>
<td>11</td>
<td>0.97  0.83  0.25  0.87  0.92  0.29  0.91  0.90  0.26  0.98  0.86  0.33</td>
<td>0.0  0.72  0.0</td>
</tr>
<tr>
<td>12</td>
<td>0.99  0.73  0.20  0.58  0.99  0.46  0.96  0.84  0.21  0.50  0.99  0.49</td>
<td>0.0  0.63  0.0</td>
</tr>
<tr>
<td>13</td>
<td>0.99  0.76  0.22  0.81  0.95  0.33  0.83  0.94  0.31  0.99  0.95  0.31</td>
<td>0.0  0.77  0.0</td>
</tr>
<tr>
<td>14</td>
<td>0.98  0.59  0.15  0.97  0.55  0.09  0.81  0.32  0.03  0.0  0.49</td>
<td>0.70  0.0  0.0</td>
</tr>
<tr>
<td>15</td>
<td>0.98  0.59  0.15  0.85  0.33  0.04  0.85  0.37  0.03  0.01  0.88  0.66</td>
<td>0.87  0.0  0.0</td>
</tr>
<tr>
<td>16</td>
<td>0.98  0.59  0.15  0.79  0.27  0.02  0.89  0.43  0.05  0.98  0.88  0.33</td>
<td>0.85  0.0  0.0</td>
</tr>
<tr>
<td>17</td>
<td>0.98  0.59  0.15  0.94  0.48  0.07  0.81  0.32  0.03  0.0  0.79</td>
<td>0.74  0.0  0.0</td>
</tr>
<tr>
<td>18</td>
<td>0.98  0.61  0.16  0.79  0.27  0.02  0.60  0.99  0.44  0.30  0.23  0.12</td>
<td>0.76  0.0  0.0</td>
</tr>
<tr>
<td>19</td>
<td>0.98  0.60  0.15  0.79  0.27  0.02  0.95  0.54  0.07  0.88  0.61  0.22</td>
<td>0.84  0.0  0.0</td>
</tr>
<tr>
<td>20</td>
<td>0.85  0.94  0.34  0.10  0.91  0.71  0.00  0.77  0.83  0.96  0.71  0.25</td>
<td>0.51  0.0  0.0</td>
</tr>
<tr>
<td>21</td>
<td>0.98  0.59  0.15  0.64  0.17  0.01  0.85  0.37  0.03  0.79  0.52  0.20</td>
<td>0.80  0.0  0.0</td>
</tr>
<tr>
<td>22</td>
<td>0.98  0.59  0.15  0.72  0.22  0.01  0.81  0.32  0.03  0.01  0.88  0.66</td>
<td>0.84  0.0  0.0</td>
</tr>
</tbody>
</table>

Table 7-1-8. Memberships for Inputs and Outputs in Interpretation

The input membership values are represented as a vector of (L_area, M_area, H_area, L_width, M_width, H_width, L_height, M_height, H_height, L_ratio, M_ratio, H_ratio). L, M, and H mean the membership values of low, medium, and high classes respectively. The output membership values are represented as an expected vector of (function0, function1, function2). In this case, function0 is to delete the region, function1 is to create a rectangle with the maximum width and height, and function2 is to create a polygon with the shape of the region boundary. The regions to which the function1 is applied are presented in pink, the regions to which the function2 is applied are presented in orange as shown in Fig. 137.
The neural networks for this stage have three layers, which are one input layer, one hidden layer, and one output layer. The numbers of nodes are 12, 10, and 3 for the input, the hidden, and the output layer respectively. The parameters of training the networks are the same as ones applied in the segmentation stage. The condition of the neural networks is represented in Fig. 7-1-5. The percentage of accuracy for the given samples was 95.45%. Fig. 7-1-6 shows the output image, and Fig. 7-1-7 shows the generated 3D computer city models. The 3D object represents buildings and houses in the input image. Each object has the same height in this case. Table 7-1-9 shows the fuzzy rules generated from the neural networks applied for this stage, and the translated rules manually are shown in Table 7-1-10.
IF low F3, high F1, high F2,
THEN likely Class 2.

IF more or less high F1, low F3, high F2,
THEN likely Class 2.

IF high F2, high F0, medium F3,
THEN likely Class 2.

IF medium F1, high F2,
THEN likely Class 2.

IF more or less high F1, low F3, medium F2,
THEN more or less likely Class 2.

IF medium F1, low F3, medium F2, low F0,
THEN likely Class 1.

IF medium F1, medium F2, more or less low F3, low F0,
THEN likely Class 1.
Table 7-1-9. Generated Fuzzy Rules in Interpretation

| IF medium F1, low F3, medium F2, low F0, | THEN more or less likely Class 1. |
| IF medium F1, low F2, medium F3, low F0, | THEN likely Class 1. |
| IF low F0, low F1, low F2, | THEN likely Class 0. |
| IF low F0, low F1, low F2, | THEN very likely Class 0. |
| IF medium F1, low F3, medium F2, | THEN unable to recognize Class 2. |
| IF low F0, low F2, low F1, | THEN very likely Class 0. |

Table 7-1-10. Translated Fuzzy Rules

| If (Area, Width, Height, Ratio)=(?, high, high, low), -> likely "create Polygon from boundary" |
| If (Area, Width, Height, Ratio)=(?, more or less high, high, low), -> likely "create Polygon from boundary" |
| If (Area, Width, Height, Ratio)=(high, ?, high, med), -> likely "create Polygon from boundary" |
| If (Area, Width, Height, Ratio)=(?, med, high, ?), -> likely "create Polygon from boundary" |
| If (Area, Width, Height, Ratio)=(?, more or less high, med, low), -> more or less "create Polygon from boundary" |
| If (Area, Width, Height, Ratio)=(low, med, med, low), -> likely "create Rectangle(Width, Height)" |
| If (Area, Width, Height, Ratio)=(low, med, med, more or less low), -> likely "create Rectangle(Width, Height)" |
| If (Area, Width, Height, Ratio)=(low, med, low, med), -> likely "create Rectangle(Width, Height)" |
| If (Area, Width, Height, Ratio)=(low, low, low, ?), -> very likely "delete the Region" |
| If (Area, Width, Height, Ratio)=(low, low, med, ?), -> likely "delete the Region" |
| If (Area, Width, Height, Ratio)=(?, med, med, low), -> ??? |
| If (Area, Width, Height, Ratio)=(low, low, low, ?), -> very likely "delete the Region" |
VII-2. Practical Case

This section shows the results of a practical case. The inputs are satellite images of Los Angeles from "Los Angeles CityRom" (Small Blue Planet Atlas Company, 1999).

The size of image is 400x400 pixels. The number of samples is 1000. Each sample pixel has a set of attributes: red, green, blue, and gray values. The classes into which the sample pixels are categorized are "House", "Street", "Green", "Ground", and "Shadow". In this case, all parameters are the same as the previous simple case. After sampling process, the sampling set is calculated into a membership value set. The information of the set for getting membership values is shown in Table 7-2-1.

Fig. 7-2-1. Input Image for Case 1
<table>
<thead>
<tr>
<th>Input features</th>
<th>Mean of All Samples</th>
<th>Mean of Low Value Samples</th>
<th>Mean of High Value Samples</th>
<th>Minimum Value</th>
<th>Maximum Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Red Value</td>
<td>138.95</td>
<td>90.21</td>
<td>194.24</td>
<td>31.0</td>
<td>255.0</td>
</tr>
<tr>
<td>Green Value</td>
<td>137.18</td>
<td>89.71</td>
<td>195.69</td>
<td>32.0</td>
<td>255.0</td>
</tr>
<tr>
<td>Blue Value</td>
<td>121.78</td>
<td>76.71</td>
<td>183.64</td>
<td>0.0</td>
<td>255.0</td>
</tr>
<tr>
<td>Gray Value</td>
<td>138.95</td>
<td>90.21</td>
<td>194.24</td>
<td>31.0</td>
<td>255.0</td>
</tr>
</tbody>
</table>

Table 7-2-1. Sample Information for Membership Values

Fig. 7-2-2. Weights Condition of Fuzzy MLP
For the segmentation stage, one hidden layer with ten neurons are applied for the fuzzy MLP. Fig. 7-2-2 shows the weight condition of the fuzzy MLP after training, and Fig.7-2-3 shows the categorized image by using the trained fuzzy MLP. The percentage of accuracy for the given samples is 76.04%. Table 7-2-2 describes the fuzzy rules generated from the fuzzy MLP. The system generates 84 rules from 1000 samples

<table>
<thead>
<tr>
<th>IF Clause</th>
<th>THEN Clause</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF high F0, high F3, high F2, high F1,</td>
<td>THEN more or less likely Class 0.</td>
</tr>
<tr>
<td>IF high F0, more or less high F3, high F2, high F1,</td>
<td>THEN unable to recognize Class 0.</td>
</tr>
<tr>
<td>IF high F0, high F3, high F2,</td>
<td>THEN likely Class 0.</td>
</tr>
<tr>
<td>Rule</td>
<td>Conclusion</td>
</tr>
<tr>
<td>----------------------------------------------------------------------</td>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td>IF high F0, more or less high F3, high F2, high F1,</td>
<td>THEN not likely Class 0.</td>
</tr>
<tr>
<td>IF high F0, high F3, high F2,</td>
<td>THEN more or less likely Class 0.</td>
</tr>
<tr>
<td>IF high F0, high F3, high F2, more or less high F1,</td>
<td>THEN not likely Class 0.</td>
</tr>
<tr>
<td>IF low F2, low F0, low F3,</td>
<td>THEN not likely Class 1.</td>
</tr>
<tr>
<td>IF low F2, more or less low F0, low F3,</td>
<td>THEN more or less likely Class 1.</td>
</tr>
<tr>
<td>IF low F2, medium F0, medium F3,</td>
<td>THEN more or less likely Class 1.</td>
</tr>
<tr>
<td>IF high F0, high F3, medium F3, medium F1,</td>
<td>THEN unable to recognize Class 1.</td>
</tr>
<tr>
<td>IF high F0, high F3, high F2, more or less high F1,</td>
<td>THEN more or less likely Class 0.</td>
</tr>
<tr>
<td>IF high F0, high F3, high F1,</td>
<td>THEN likely Class 0.</td>
</tr>
<tr>
<td>IF low F2, medium F0, medium F3,</td>
<td>THEN more or less likely Class 0.</td>
</tr>
<tr>
<td>IF high F0, high F3, high F2, high F1,</td>
<td>THEN likely Class 0.</td>
</tr>
<tr>
<td>IF high F0, high F3, high F2,</td>
<td>THEN more or less likely Class 0.</td>
</tr>
<tr>
<td>IF high F0, high F3, high F2, high F1,</td>
<td>THEN likely Class 0.</td>
</tr>
<tr>
<td>IF high F0, high F3, high F2, more or less high F1,</td>
<td>THEN more or less likely Class 0.</td>
</tr>
<tr>
<td>IF high F0, high F3, high F2, high F1,</td>
<td>THEN likely Class 0.</td>
</tr>
<tr>
<td>IF low F2, medium F0, medium F3,</td>
<td>THEN more or less likely Class 0.</td>
</tr>
<tr>
<td>IF high F0, high F3, high F2, more or less high F1,</td>
<td>THEN unable to recognize Class 0.</td>
</tr>
<tr>
<td>IF high F0, high F3, high F2, high F1,</td>
<td>THEN more or less likely Class 0.</td>
</tr>
<tr>
<td>IF high F0, high F3, high F2,</td>
<td>THEN more or less likely Class 0.</td>
</tr>
<tr>
<td>IF high F0, high F3, high F2, more or less high F1,</td>
<td>THEN unable to recognize Class 0.</td>
</tr>
<tr>
<td>IF high F0, high F3, high F2, high F1,</td>
<td>THEN likely Class 0.</td>
</tr>
<tr>
<td>IF high F0, high F3, high F2,</td>
<td>THEN more or less likely Class 0.</td>
</tr>
<tr>
<td>IF high F0, high F3, high F2, more or less high F1,</td>
<td>THEN unable to recognize Class 0.</td>
</tr>
<tr>
<td>IF high F0, high F3, high F2, high F1,</td>
<td>THEN likely Class 0.</td>
</tr>
<tr>
<td>IF high F0, high F3, high F2,</td>
<td>THEN more or less likely Class 0.</td>
</tr>
<tr>
<td>IF high F0, high F3, high F2, more or less high F1,</td>
<td>THEN unable to recognize Class 0.</td>
</tr>
<tr>
<td>IF high F0, high F3, high F2, high F1,</td>
<td>THEN likely Class 0.</td>
</tr>
<tr>
<td>IF high F0, high F3, high F2,</td>
<td>THEN more or less likely Class 0.</td>
</tr>
<tr>
<td>IF high F0, high F3, high F2, more or less high F1,</td>
<td>THEN likely Class 0.</td>
</tr>
<tr>
<td>IF high F0, high F3, high F2, high F1,</td>
<td>THEN more or less likely Class 0.</td>
</tr>
<tr>
<td>IF high F0, high F3, high F2,</td>
<td>THEN likely Class 0.</td>
</tr>
<tr>
<td>IF high F0, high F3, high F2, more or less high F1,</td>
<td>THEN more or less likely Class 0.</td>
</tr>
<tr>
<td>IF high F0, high F3, high F2, high F1,</td>
<td>THEN likely Class 0.</td>
</tr>
<tr>
<td>IF low F2, medium F0, medium F3,</td>
<td>THEN more or less likely Class 0.</td>
</tr>
<tr>
<td>IF medium F0, medium F3, medium F1, medium F2,</td>
<td>THEN not likely Class 1.</td>
</tr>
<tr>
<td>IF high F0, high F3, high F2, high F1,</td>
<td>THEN likely Class 0.</td>
</tr>
<tr>
<td>IF high F0, high F3, high F2,</td>
<td>THEN more or less likely Class 0.</td>
</tr>
<tr>
<td>IF high F0, high F3, high F2, more or less high F1,</td>
<td>THEN unable to recognize Class 3.</td>
</tr>
<tr>
<td>IF high F0, high F3, high F2, high F1,</td>
<td>THEN likely Class 0.</td>
</tr>
<tr>
<td>IF high F0, high F3, high F2, more or less high F3,</td>
<td>THEN not likely Class 0.</td>
</tr>
<tr>
<td>IF medium F0, medium F3, medium F1, high F2,</td>
<td>THEN not likely Class 1.</td>
</tr>
<tr>
<td>IF high F0, high F3, high F2, more or less high F1,</td>
<td>THEN unable to recognize Class 0.</td>
</tr>
<tr>
<td>IF high F0, high F2, high F3, more or less high F1,</td>
<td>THEN more or less likely Class 0.</td>
</tr>
<tr>
<td>IF high F0, high F2, high F3,</td>
<td>THEN likely Class 0.</td>
</tr>
<tr>
<td>IF high F0, high F2, high F3, more or less high F1,</td>
<td>THEN more or less likely Class 0.</td>
</tr>
<tr>
<td>IF high F0, high F2, high F3,</td>
<td>THEN likely Class 0.</td>
</tr>
<tr>
<td>IF high F0, high F2, high F3, more or less high F1,</td>
<td>THEN more or less likely Class 0.</td>
</tr>
<tr>
<td>IF high F0, high F2, high F3,</td>
<td>THEN likely Class 0.</td>
</tr>
<tr>
<td>IF high F0, high F2, high F3, more or less high F1,</td>
<td>THEN likely Class 0.</td>
</tr>
<tr>
<td>IF high F0, high F2, high F3,</td>
<td>THEN more or less likely Class 0.</td>
</tr>
<tr>
<td>IF high F0, high F2, high F3, more or less high F1,</td>
<td>THEN likely Class 0.</td>
</tr>
<tr>
<td>IF high F0, high F2, high F3,</td>
<td>THEN more or less likely Class 0.</td>
</tr>
<tr>
<td>IF high F0, high F2, high F3, more or less high F1,</td>
<td>THEN likely Class 0.</td>
</tr>
<tr>
<td>IF high F0, high F2, high F3,</td>
<td>THEN more or less likely Class 0.</td>
</tr>
<tr>
<td>IF high F0, high F2, high F3, more or less high F1,</td>
<td>THEN likely Class 0.</td>
</tr>
<tr>
<td>IF high F0, high F2, high F3,</td>
<td>THEN more or less likely Class 0.</td>
</tr>
<tr>
<td>IF high F0, high F2, high F3, more or less high F1,</td>
<td>THEN likely Class 0.</td>
</tr>
<tr>
<td>IF high F0, high F2, high F3,</td>
<td>THEN more or less likely Class 0.</td>
</tr>
<tr>
<td>IF high F0, high F2, high F3, more or less high F1,</td>
<td>THEN likely Class 0.</td>
</tr>
<tr>
<td>IF high F0, high F2, high F3,</td>
<td>THEN more or less likely Class 0.</td>
</tr>
<tr>
<td>IF high F0, high F2, high F3, more or less high F1,</td>
<td>THEN likely Class 0.</td>
</tr>
<tr>
<td>IF low F2, medium F0, low F3,</td>
<td>THEN more or less likely Class 1.</td>
</tr>
<tr>
<td>IF medium F0, medium F3, medium F1, medium F2,</td>
<td>THEN not likely Class 1.</td>
</tr>
<tr>
<td>IF high F0, high F3, high F2, high F1,</td>
<td>THEN likely Class 0.</td>
</tr>
<tr>
<td>IF high F0, high F3, high F2,</td>
<td>THEN more or less likely Class 0.</td>
</tr>
<tr>
<td>IF high F0, high F3, high F2, more or less high F1,</td>
<td>THEN unable to recognize Class 3.</td>
</tr>
<tr>
<td>IF low F2, low F0, low F3,</td>
<td>THEN more or less likely Class 4.</td>
</tr>
<tr>
<td>IF low F2, low F0, low F3,</td>
<td>THEN more or less likely Class 4.</td>
</tr>
<tr>
<td>IF low F2, low F0, low F3,</td>
<td>THEN more or less likely Class 4.</td>
</tr>
<tr>
<td>IF low F2, low F0, low F3,</td>
<td>THEN more or less likely Class 4.</td>
</tr>
<tr>
<td>IF low F2, low F0, low F3,</td>
<td>THEN more or less likely Class 4.</td>
</tr>
<tr>
<td>IF low F2, low F0, low F3,</td>
<td>THEN more or less likely Class 4.</td>
</tr>
<tr>
<td>IF low F2, low F0, low F3, medium F1,</td>
<td>THEN unable to recognize Class 2.</td>
</tr>
<tr>
<td>IF low F2, low F0, low F3, medium F1,</td>
<td>THEN not likely Class 2.</td>
</tr>
</tbody>
</table>
Table 7-2-2. Generated Fuzzy Rules in Segmentation

The next interpretation stage applied 100 samples for the segmented image as shown in Fig. 7-2-4. Each sample region has a set of attributes, the area, the width, the height, and the ratio of the width and height. The applied functions for the outputs are the same as those of the simple case in the previous section. The information of samples is shown in Table 7-2-3.
Table 7-2-3. Parameters for Memberships in Interpretation

The neural networks for the interpretation has one hidden layer with 10 neurons, and Fig. 7-2-5 shows the weight condition of the networks after training. The applied parameters are the same as the previous simple case. Fig. 7-2-6 shows the output refined image. Fig. 7-2-7 and 8 show the output 3D computer city model with simple rendering and with texture mapping render respectively.
Fig. 7-2-5. Weight Condition of Fuzzy MLP for Interpretation

Fig. 7-2-6. Output Image Applied Morphological Functions
Fig. 7-2-7. 3D Computer City Model

Fig. 7-2-8. 3D Computer City Model with Texture Mapping
Table 7-2-4 shows the generated fuzzy rules in the interpretation stage.

<table>
<thead>
<tr>
<th>IF Clause</th>
<th>THEN Clause</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF medium F3, high F2,</td>
<td>THEN likely Class 2.</td>
</tr>
<tr>
<td>IF medium F1, medium F3, medium F2, medium F0,</td>
<td>THEN more or less likely Class 2.</td>
</tr>
<tr>
<td>IF high F0, high F1, medium F3,</td>
<td>THEN very likely Class 2.</td>
</tr>
<tr>
<td>IF high F1, high F0, medium F3,</td>
<td>THEN very likely Class 2.</td>
</tr>
<tr>
<td>IF high F1, low F2, medium F0,</td>
<td>THEN more or less likely Class 2.</td>
</tr>
<tr>
<td>IF medium F3, high F1, medium F0, medium F2,</td>
<td>THEN likely Class 2.</td>
</tr>
<tr>
<td>IF high F1,</td>
<td>THEN more or less likely Class 2.</td>
</tr>
<tr>
<td>IF medium F1, low F0, medium F3,</td>
<td>THEN more or less likely Class 1.</td>
</tr>
<tr>
<td>IF medium F1, low F0, low F2,</td>
<td>THEN likely Class 1.</td>
</tr>
<tr>
<td>IF medium F1, low F0, low F2,</td>
<td>THEN very likely Class 1.</td>
</tr>
<tr>
<td>IF medium F1, high F0, medium F3, medium F2,</td>
<td>THEN likely Class 2.</td>
</tr>
<tr>
<td>IF medium F1, medium F0, medium F2,</td>
<td>THEN more or less likely Class 2.</td>
</tr>
<tr>
<td>IF medium F1, medium F2, medium F3,</td>
<td>THEN likely Class 2.</td>
</tr>
<tr>
<td>IF medium F1, medium F3, medium F2, medium F0,</td>
<td>THEN unable to recognize Class 2.</td>
</tr>
<tr>
<td>IF medium F1, low F0, medium F3, low F2,</td>
<td>THEN likely Class 1.</td>
</tr>
<tr>
<td>IF more or less high F1, high F3, medium F0,</td>
<td>THEN likely Class 1.</td>
</tr>
<tr>
<td>IF more or less high F1, medium F3, medium F2, medium F0,</td>
<td>THEN very likely Class 2.</td>
</tr>
<tr>
<td>IF low F0, low F2, low F3,</td>
<td>THEN very likely Class 2.</td>
</tr>
<tr>
<td>IF low F0, low F0, low F2,</td>
<td>THEN likely Class 0.</td>
</tr>
<tr>
<td>IF low F0, low F0,</td>
<td>THEN likely Class 0.</td>
</tr>
<tr>
<td>IF low F0, low F0, high F2,</td>
<td>THEN more or less likely Class 1.</td>
</tr>
<tr>
<td>IF more or less low F3, low F0,</td>
<td>THEN likely Class 0.</td>
</tr>
<tr>
<td>IF more or less low F3, low F0,</td>
<td>THEN very likely Class 0.</td>
</tr>
<tr>
<td>IF medium F3, low F0,</td>
<td>THEN very likely Class 0.</td>
</tr>
<tr>
<td>IF medium F3, low F0, low F2,</td>
<td>THEN very likely Class 0.</td>
</tr>
<tr>
<td>IF more or less low F3, low F0,</td>
<td>THEN likely Class 0.</td>
</tr>
<tr>
<td>IF high F3, low F0, low F2,</td>
<td>THEN very likely Class 0.</td>
</tr>
<tr>
<td>IF high F3, low F0, low F2,</td>
<td>THEN likely Class 0.</td>
</tr>
<tr>
<td>IF medium F3, low F0, low F2,</td>
<td>THEN likely Class 0.</td>
</tr>
<tr>
<td>IF low F0, high F1, low F2,</td>
<td>THEN more or less likely Class 0.</td>
</tr>
<tr>
<td>IF medium F2, low F3, low F0,</td>
<td>THEN very likely Class 0.</td>
</tr>
<tr>
<td>IF medium F2, low F0, low F3,</td>
<td>THEN likely Class 0.</td>
</tr>
<tr>
<td>IF low F3, low F0, low F3,</td>
<td>THEN likely Class 0.</td>
</tr>
<tr>
<td>IF low F3, low F0,</td>
<td>THEN likely Class 0.</td>
</tr>
<tr>
<td>IF low F0, high F2,</td>
<td>THEN likely Class 0.</td>
</tr>
<tr>
<td>IF low F3, medium F2, low F0,</td>
<td>THEN more or less likely Class 0.</td>
</tr>
<tr>
<td>IF medium F2, low F0, low F3,</td>
<td>THEN very likely Class 0.</td>
</tr>
<tr>
<td>IF medium F1, high F2, medium F0, low F3,</td>
<td>THEN more or less likely Class 1.</td>
</tr>
<tr>
<td>IF high F1, medium F3, medium F0, medium F2,</td>
<td>THEN very likely Class 2.</td>
</tr>
<tr>
<td>IF low F0, high F1, low F2,</td>
<td>THEN more or less likely Class 0.</td>
</tr>
</tbody>
</table>

Table 7-2-4. Generated Fuzzy Rules in Interpretation

VII-3. Experiments

VII-3-1. Experiments with Different Sample Number

This section shows the results of experiments resulting through changing parameters in the system. In each case, different number of samples (50 -5000) is applied. Each case is
defined by the number of samples, the number of epochs for the neural networks to converge, the number of total updating in the neural networks, and the percentage of samples categorized correctly as shown in Table 7-3-1. In all cases, the neural networks have one hidden layer with ten neurons. Three kinds of graphical information are represented in each case. The left image shows Kohonen's self-organizing map (SOM) of sample data. The SOM data has 100 (10x10) nodes, which are the 100 representative vectors normally distributed among samples as explained in Section II-5-7. The middle image shows the weights condition of the neural networks. The right shows the image classified by the neural networks.

**Evaluation Method**

It is beyond the scope of this project to discuss the relation between the optimum sample number used for training neural networks and the structure of the networks. It is very complicated and has yet proved mathematically. Moreover, the data applied here is not objective but subjective. Therefore, this research proposes one visual method for defining the proper sample number. It has three steps. The first step is to observe the segmented image visually. Though more samples have the ability to optimize the neural networks, it is unnecessary to add more samples if the segmented image does not have any improvement. The second step is to observe the SOM data. It shows the topological relations of categories among samples. If the data has clear boundaries between the categories, the effect of ill data is minimal. The third step is to observe the epoch number and the accuracy of samples, which are how many times it takes for the neural networks to converge and how many
samples are categorized correctly by the networks. These three proposed observations are not always appropriate, however, because the user chooses samples subjectively. It is also the purpose of this project to allow the user to manipulate internal logic of the system dynamically. In this case, the internal logic is the relation between the pixel and the category expected by the user, or the relation between the segmented region and the corresponding function expected by the user. As a result, the user can define the system behaviors to be what he/she wants. The following shows one example of finding the proper sample for the case of Fig.7-2-1.

**Evaluation**

By observing the results of segmented images, labeled "c" from Fig. 7-3-1 to Fig. 7-3-7, 200 samples seem to be enough to categorize the pixels of image into 5 classes. It is difficult to see any difference among the images with more than 200 samples. However, the results of SOM images, labeled "a" from Fig. 7-3-1 to Fig. 7-3-7, show that more than 500 samples are required to cluster the samples topologically well. Fig. 7-3-3-a has isolated three green and blue hexagons. Therefore, there could be some ill data among the sample pixels labeled "House" and "Green", and the data could be affecting other correct samples. On the other hand, from Fig. 7-3-4-a to Fig. 7-3-7-a, the samples are topologically mapped clearly. Therefore, 500 samples is a candidate for a suitable sample number. Moreover, the total updating numbers between 200-sample case and 500-sample case are almost same. Considering the three observations, it can be concluded that 500 is a reasonable sample number in this case.
<table>
<thead>
<tr>
<th>Number of Samples</th>
<th>Epoch Number to Converge</th>
<th>Total Update Number</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>440</td>
<td>22000</td>
<td>98.0%</td>
</tr>
<tr>
<td>100</td>
<td>497</td>
<td>48700</td>
<td>92.0%</td>
</tr>
<tr>
<td>200</td>
<td>240</td>
<td>48000</td>
<td>92.0%</td>
</tr>
<tr>
<td>500</td>
<td>225</td>
<td>112500</td>
<td>83.0%</td>
</tr>
<tr>
<td>1000</td>
<td>284</td>
<td>284000</td>
<td>81.5%</td>
</tr>
<tr>
<td>2000</td>
<td>111</td>
<td>222000</td>
<td>71.0%</td>
</tr>
<tr>
<td>5000</td>
<td>129</td>
<td>645000</td>
<td>65.0%</td>
</tr>
</tbody>
</table>

Table 7-3-1. Changes by the Number of Samples

Fig. 7-3-1. Results of 50 Samples: a) SOM data, b) Weight Conditions, c) Segmented Image

Fig. 7-3-2. Results of 100 Samples: a) SOM data, b) Weight Conditions, c) Segmented Image
Fig. 7-3-3. Results of 200 Samples: a) SOM data, b) Weight Conditions, c) Segmented Image

Fig. 7-3-4. Results of 500 Samples: a) SOM data, b) Weight Conditions, c) Segmented Image

Fig. 7-3-5. Results of 1000 Samples: a) SOM data, b) Weight Conditions, c) Segmented Image
VII-3-2. Experiments with Different Numbers of Neurons and Hidden Layers

The results of experimenting with different numbers of hidden layers and neurons per layer are shown in Table 7-3-2 and graphically in figure 7-3-8 through figure 7-3-14. Based on the experimental results in the previous chapter, 500 samples were selected for each case. The first three cases test one hidden layer with 3, 5 and 15 neurons respectively. The second three cases use two hidden layers, again testing 3, 5 and 15 neurons in each hidden layer.
**Evaluation**

It is also beyond the scope of this research to find the optimum number of hidden layer and neurons for neural networks mathematically and statistically. It is, however, an issue of great importance in computer science. Therefore, again a manual method for selecting the proper structure of neural networks by observing the segmented images is suggested. This experiment also demonstrates how easily the user can control and adapt the system to what he/she requires by changing the parameters of hidden layers. This approach is completely different from traditional expert systems in which the user has to actually program complex rules with his/her expert experiments.

Using two hidden layers with 10 neurons each comes closest to replicating the base case (one hidden layer with 10 neurons). The other test cases show considerable noise and incorrect categorizations. None of the test cases produce better quality results than the base case because the system recognizes the pixels of wide street at the right of the image incorrectly as ground (light blue pixels) except in the base case. Therefore, it is safe to say that one hidden layer with 10 neurons is suitable for this kind of images. Needless to say, neural networks with more hidden layers and more neurons in each layer have more ability to categorize the samples. Therefore, it may be possible to get better results by changing the other parameters of neural networks such as minimum delta for errors, momentum rates, etc described in Section VI-2-1-2-3. However, experimentation in this direction was not attempted, as it is not relevant to the main goals of this research.
<table>
<thead>
<tr>
<th>Neurons in Hidden Layer</th>
<th>Number to Converge</th>
<th>Accuracy of Samples</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>225</td>
<td>83.0%</td>
</tr>
<tr>
<td>3</td>
<td>283</td>
<td>83.0%</td>
</tr>
<tr>
<td>5</td>
<td>172</td>
<td>83.2%</td>
</tr>
<tr>
<td>15</td>
<td>187</td>
<td>84.8%</td>
</tr>
<tr>
<td>3.3</td>
<td>167</td>
<td>83.4%</td>
</tr>
<tr>
<td>5.5</td>
<td>333</td>
<td>85.0%</td>
</tr>
<tr>
<td>10.10</td>
<td>208</td>
<td>82.4%</td>
</tr>
</tbody>
</table>

Table 7-3-2. Changes by the Number of Hidden Layers and Neurons

Fig.7-3-8. Results of 500 Samples with 1(10) Hidden Layer:
   a) Weight Conditions, b) Segmented Image

Fig.7-3-9. Results of 500 Samples with 1(3) Hidden Layer:
   a) Weight Conditions, b) Segmented Image
Fig. 7-3-10. Results of 500 Samples with 1(5) Hidden Layer:
   a) Weight Conditions, b) Segmented Image

Fig. 7-3-11. Results of 500 Samples with 1(15) Hidden Layer:
   a) Weight Conditions, b) Segmented Image

Fig. 7-3-12. Results of 500 Samples with 2(3,3) Hidden Layers:
   a) Weight Conditions, b) Segmented Image
Fig. 7-3-13. Results of 500 Samples with 2(5,5) Hidden Layers:
a) Weight Conditions, b) Segmented Image

Fig. 7-3-14. Results of 500 Samples with 2(10,10) Hidden Layers:
a) Weight Conditions, b) Segmented Image

The following image shows the satellite image used in all the above cases with 5000 samples inputted manually.
Fig. 7-3-15. Image of 5000 Samples
VII-3-3. Applying the Trained Weights to Similar Cases

Figure 7-3-15 shows the results of the base case trained with 500 samples, using one hidden layer with 10 neurons. The neural network weights produced from this base case were used to process a similar satellite image without retraining. These results are shown in Figure 7-3-16. These same trained weights were applied to another image (shown in Fig. 7-3-17) with less successful results. Many of the pixels in this new image were not categorized correctly, particularly the houses with red roofs. Adding more samples to the currently trained weights using this new image produced a significantly better result as shown in Figure 7-3-18. In this case, 20 more samples are added to the previous ones.
Fig. 7.3.16. Results of Initial Satellite Image: a) Input Satellite Image, b) Categorized Image, c) Extracted Image, d) Interpreted Image, e) 3D City Model, f) 3D City Model with Texture Mapping
Fig. 7-3-17. Results of Similar Satellite Image without Training:
a) Input Satellite Image, b) Categorized Image, c) Extracted Image, d) Interpreted Image,
e) 3D City Model f) 3D City Model with Texture Mapping
Fig. 7-3-18. Results of Similar Image without Training.
   a) Input Satellite Image, b) Categorized Image

Fig. 7-3-19. Results of Similar Satellite Image after Refining Process:
   b) Categorized Image, c) Extracted Image, d) Interpreted Image,
Evaluation

This section evaluates the above three cases. In each case, the left image shows the system output and the right shows the result expected by the user. In the expected result the blue areas are the objects clearly detectable as buildings or houses, and the pink areas are the objects that are unclear but recognizable as buildings or houses. Those decisions are based on the user’s subjective requirements, and the expected images are drawn by the user.

Fig. 7-3-19. Results of Similar Satellite Image after Refining Process:
e) 3D City Model f) 3D City Model with Texture Mapping

Fig. 7-3-20. Comparison in the case of Fig. 7-3-16:
a) System Output, b) Expected Result
In the base case of Fig. 7-3-16, the expected result has 25 blue and 37 pink objects. Twenty-three objects from the 25 blue ones are detected by the system, and 6 from 37 pink ones. Fourteen objects detected by the system are not overlapped with the expected ones as shown in Fig. 7-3-20.

The second case (Fig. 7-3-17) is that the system generates objects automatically by applying the previous neural networks without training. In this case, the expected result has 29 blue and 9 pink objects. Twenty-eight objects from the 29 blue ones are detected by the system, and 4 from 9 pink ones. Eight objects detected by the system are not overlapped with the expected ones as shown in Fig. 7-3-21.

![Fig. 7-3-21. Comparison in the case of Fig. 7-3-17: a) System Output, b) Expected Result](image)

![Fig. 7-3-22. Comparison in the case of Fig. 7-3-19: a) System Output, b) Expected Result](image)
The third case (Fig. 7-3-19) is that the system generates objects after training the previous neural networks. In this case, the expected result has 33 blue and 48 pink objects. Thirty-one objects from the 33 blue ones are detected by the system, and 28 from 48 pink ones. Twelve objects detected by the system are not overlapped with the expected ones as shown in Fig. 7-3-22.

<table>
<thead>
<tr>
<th>Case</th>
<th>Clear Objects</th>
<th>Ambiguous Objects</th>
<th>Mistaken Objects by System</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fig. 7-3-16</td>
<td>23/25 (92.0%)</td>
<td>6/37 (16.2%)</td>
<td>14</td>
</tr>
<tr>
<td>Fig. 7-3-17</td>
<td>28/29 (96.6%)</td>
<td>4/9 (44.4%)</td>
<td>8</td>
</tr>
<tr>
<td>Fig 7-3-19</td>
<td>31/33 (93.9%)</td>
<td>28/48 (58.3%)</td>
<td>12</td>
</tr>
</tbody>
</table>

Table 7-3-3. Results of Comparison

Table 7-3-3 shows the results of all cases. For clear objects, the system can perform 94.2% correctly. For ambiguous objects, it can perform 39.6% correctly.

VII-4. Summary & System Evaluation

Summary

This section shows how the system behaves in generating 3D objects from images. The first case used just a simple painting while the second applied the system to a real satellite image. Experiments detailed in Section VII-3 shows how the user should control the system in order to adapt the system to his/her requirements. That section also demonstrates how to evaluate and change system parameters such as the number of samples, the number of hidden layers, and the number of neurons in hidden layers. Moreover, the adaptability of the system to similar cases is demonstrated in Section VII-3-3. Specifically, the system could adapt to interpret 94.2% of the objects that are clearly recognizable. However, it did not perform well for ambiguous objects that are not clearly recognizable using only from
the color information (Interpretation of those objects was only 39.6% accurate).

**System Evaluation**

The most important part of this system is that the user has access to the internal logic of the system and can adapt parameters to his/her requirements. In this system the user can define the relations between pixels and their expected categories and between the extracted regions and their corresponding functions by giving sample relations. The system can learn the logic/relation from the given samples to determine the weights of neural networks. To the best of my knowledge, there are no CAD/CAAD systems that have this ability, and it makes possible to re-use the prior knowledge in creating 3D objects.

The potential contributions of this project to the use of CAD/CAAD in architecture and urban planning include 1) reducing routine works, 2) adapting to similar cases, and 3) supporting users’ subjective ideas. Specifically,

1) Once the system learns the user's subjective idiosyncrasies and methodologies, it can apply them to the other cases without having to re-invent the same process all over every new time. Here the user’s requirement is to give the relations between the pixel/region information in images and the applied functions (delete, create rectangles, and create polygons). As a result, the system infers the most suitable functions and applies them to the untrained pixels/regions automatically. This automatic inferring ability is very helpful, especially for CAD/CAAD users who create 3D objects step by step manually, because it can also reduce the labor and time in creating 3D objects as describe in Fig. 7-3-17.
2) When the system's behavior does not match the user's requirements well, the user has only to add some more samples for improving the performance of the system to those cases. For example, if the system does not recognize red objects as buildings, the user has only to add some samples that are sets of red color information and the expected category as a building. The system then can detect red areas as buildings as described in Fig. 7-3-18 and 19. This flexibility is very useful for CAD/CAAD users who have to design objects from the beginning even when dealing with the similar cases because they cannot change the behaviors of built-in functions in CAD/CAAD systems.

3) The user can make the system learn not by programming rules but by giving samples. In other words, the system allows the user to change the internal logic. For example, the system does not have a logic rule, "if the pixel color is red, then it is a part of building" in the case of Fig. 7-3-18. However, it learns the rule without changing other rules after the user adds some samples in the case of Fig.7-3-19. Therefore, it can be said that this system supports the user’s subjective ideas in extracting the visual information. It is a completely different from traditional expert systems, and very helpful for the designers, planners, and artists who have their own ideas because the system can learn their ideas.

Limitation of Evaluation

It is very difficult to define the optimum parameters for neural networks, because the
theories have not been proved mathematically. In this project, the evaluation mostly relies on observing the result images in order to find the best values for the parameters.

Limitation of Accuracy

Though this system demonstrates how 3D buildings can be created from satellite images, the generated objects are not as accurate as the real ones because of the limitation of detecting objects only from satellite images. However, they can be applied usefully in visualizing cities in computers, because there are many cases in which only simple 3D volume objects are needed and there is no time to create detailed objects.
Chapter VIII

Conclusion

VIII-1. Accomplishment

This research shows the possibility of implementing design knowledge in CAD/CAAD systems. The knowledge is defined as the relations between the given situations and the applied functions to the situations. To be concrete, a computer system that can generate 3D computer city models from satellite images is programmed as a test case. In the program, the user teaches the relations between the given satellite images and the applied morphological functions, and the system can generate 3D computer city models with the trained the user's design knowledge. Fuzzy multiple layers perceptron, which is one of the neuro-fuzzy systems, is applied in the system.

The system can support different kinds of images such as paintings and real images. Even though the user has to train the relations by giving the sample data and the expected outputs corresponding to the data, the system has the flexibility of adapting to the different kinds of inputs. This flexibility does not exist in traditional expert systems.

Moreover, the system can store the user's design knowledge and allow him/her to reuse and manipulate the knowledge. As a result, the user can apply his/her prior knowledge to a case that is similar to previous cases. In detail, if the user trained the neural network weights well in the system, it can apply the expected functions automatically. Only when the
outputs are far from the expected results the user have to observe and refine the system. This ability is accomplished by combining both the ability of neural networks and that of fuzzy systems. Neural networks have the ability of learning and adapting, and fuzzy systems have the ability of representing the knowledge as a set of fuzzy rules in linguistic forms. Combining those abilities makes it possible for the user to recognize how well the system is trained and what kinds of information are not trained. The user can adapt the system to the expected conditions by observing the generated fuzzy rules and refining the neural network weights by giving the additional samples that the system has not been trained well. As a result, the system can generate the expected 3D computer city models semi-automatically in this project. This ability helps to heighten the usability of CAD/CAAD systems because the user does not have to create 3D models step by step manually in the system.

VIII-2. Questions

Though this project shows one method to build design knowledge and apply the knowledge in CAD/CAAD systems, there is room for improvement. This section shows some questions considered from the result of this project.

**Question1. Are there some other ways to improve the training process?**

In this project, the neural networks are trained by choosing samples and giving the expected result for each sample. Though this training method is simple, a lot of samples are required. Therefore, it is important to find better ways to reduce the labor and time in
training the neural networks in CAD/CAAD systems.

**Question 2. Are the features of each pixel enough to segment the image?**

In the process of segmentation, several features of pixels such as the red-value, the green-value, blue-value, etc. are applied for the inputs of fuzzy MLP. However, the results are not always what the user expects because of insufficient samples or limitation of segmentation only with the features of each pixel. Therefore, it is important to find additional features to improve the accuracy of segmentation.

**Question 3. How many functions should be pre-defined for the interpretation stage?**

In the process of interpretation, several morphological functions are applied to reduce the ill data, to refine the shapes of regions, and to generate polygons from the refined shapes. In this project, a few simple functions such as deleting the regions, creating rectangles from the width and height of the regions, and creating polygons from the boundary of the regions, etc. are pre-defined. The system does not support enough functions for practical designing and planning cases because it was developed as a test case in building design knowledge. Needless to say, it is important to research what kinds of functions are necessary for practical design cases and how many functions should be built into the system in order to develop a practical system. Therefore, it is necessary to find ways to make this project practical.
Question 4. Were the generated fuzzy rules helpful to recognize the information that the neural networks learn?

In this project, the system can generate fuzzy rules representing how well the neural networks learned, and the rules are used to help the user's training process. Though each generated rule is easy to understand, it is too complicated for users to recognize all relations described in the set of fuzzy rules. Therefore, it is necessary to simplify the relations in order for users to understand the information easily.

Question 5. Was it possible to represent the design knowledge in this project?

This project tries to implement a system, which can build design knowledge to generate 3D computer city models from satellite images. The system generates the models by applying the appropriate functions for segmented regions in the images. Therefore, the design knowledge in generating 3D objects is implemented in the system. However, it is not true that the knowledge of generating objects is the design knowledge. In fact, the knowledge of manipulating objects is the other important factor in design knowledge. Therefore, it is important to integrate the design knowledge of manipulating objects into the system as well.

VIII-3. Improvement in the Future

Considering the questions described in the previous section, this section shows several research directions to improve the system in the future.
8-3-1. 3D Computer City Models

In this research, the design knowledge is defined as the relations between the given situations and the applied functions in the situations. In fact, the technologies of neuro-fuzzy systems are applied to represent the design knowledge in generating 3D computer city models from satellite images because many designers and planners spend a lot of time in creating the models manually. However, though the design knowledge of generating objects is implemented in the system, it will be necessary to implement the design knowledge of manipulating the generated objects in order to make the system more practical.

To implement the design knowledge of manipulating objects into this system, the knowledge can be defined as the relations between a condition of objects and the applied functions to change the conditions. It is more difficult to train the relations in manipulating objects than those in generating objects, because the system must recognize the conditions and represent them. Some technologies of neural networks to recognize and represent the conditions have been proposed (Cloete and Zurada, 2000). Therefore, the next step of this research is to find the methods of recognizing and representing the conditions in the future.

8-3-2. Fuzzy Rules (Proposal for Question4)

In this project, the generated fuzzy rules are used to observe what kind of information the fuzzy MLP has learned and helped users in training and refining the weights of neural networks. However, the generated fuzzy rules are too complicated for users to be able to
recognize the information exactly. The generated rules are too many, and some of them overlap. For example, the two rules, "if feature1 is high, then likely Class2" and "if feature1 is high, then very likely Class2" were recognized as two different rules in the system.

Some technologies to reduce the redundant fuzzy rules by using genetic algorithms (Ishibuchi, 1995), controlling uncertainty margins (Shavlik, 1994), or fuzzy set theories (Sun, 1994) have been researched. Therefore, the next step in improving this research is to use those technologies to find the methods to make the generated fuzzy rules easy to understand.

8-3-3. Collaboration with other Knowledge

The system developed in this research allows the user to store his/her design knowledge. The design knowledge is stored as neural network weight data. As a result, it is possible to apply the weights to the other cases and to generate objects. In short, the weight data is the user's design knowledge in the system. Therefore, researchers in different study fields can store their knowledge as their weight data in one system. This approach makes it possible to design collaboratively among designers and planners of different fields such as designers of construction, environment, and planning. Moreover, the people who do not have any design knowledge such as clients or students can use the expert knowledge by applying the weight data that the experts train. Therefore, it is useful to research methods to use this system collaboratively as the next step in the future.
8-3-4. Training by Communication in Natural Languages

This project applies the training by samples to supervise the neural networks. Since the main objective of the project is to generate 3D computer city models from satellite images, it was possible to train the neural networks by choosing pixels or regions and giving the expected outputs. However, in order to develop practical CAD/CAAD systems, it is necessary to implement the ability of not only generating objects but also manipulating the generated objects as stated in section 8-3-1. Moreover, it will be necessary to apply the training by communication in natural languages in order to implement the ability into the systems, because it takes a lot of time to train the networks by giving samples and it is almost impossible to represent all design conditions. Therefore, the system should be improved to support training by communications. To be concrete, the user gives the message to manipulate the object, and the system applies the expected functions and returns the manipulated objects. If the user is not satisfied with the results, the system applies the next candidate function. During the communications, the system learns the relations between the conditions of objects and the applied functions corresponding to the user's messages. There are many research projects applying training by communication in natural language processing in AI (Sharkey, 1992). Therefore, the next step to improve this research is to try and test the possibility of implementing the technologies of training by communications into the system.

8-3-5. Application in the Internet

This system was programmed in Java, which is one of the object-oriented programming
languages. Java has the advantage in network environments such as the Internet and client/server systems. Therefore, it is possible to use this system through the network environments in order to share common design knowledge in architecture and urban planning. If the system is used on the Internet, the laborious process of training the neural networks can be shared, and the trained data will be closer to the common design knowledge of many designers. This is one application of this system in the future.

**VIII-4. Conclusion**

This research presents the possibility of implementing design knowledge into CAD/CAAD systems in order to solve current system's problems, which are the complex interfaces, the poor flexibility of similar cases, the poor usability in reusing design knowledge, and the difficulty in representing design knowledge. The main reason of those problems is that the current CAD/CAAD system cannot represent, store, or manipulate design knowledge in architecture and urban planning, which is the combination of mathematical knowledge and human knowledge.

In this research, the knowledge is defined as the relations between the given situations and the applied functions to the situations. To be concrete, a computer system that can generate 3D computer city models from satellite images has been programmed as a test case, because designers and planners spend a lot of labor and time to create 3D computer city models one by one manually.
In the program, an implementation of a neuro-fuzzy system into a CAAD system has been demonstrated. The most effective benefit has been that the system's users could adapt the system to what they would expect. Especially, fuzzy Multiple Layers Perceptron (MLP) could allow the users to observe the complex relations between inputs and outputs as a collection of fuzzy rules in linguistic forms. This ability made it possible for the system's users to understand how well the neural networks can be trained and what kinds of relations can be supervised. The training by samples has been applied to make the neural networks learn. As a result, the system has demonstrated its usability and flexibility, which most of the traditional expert systems do not have. In other words, the user can store his/her original design knowledge in the system.

However, there is still room for improvement in the system. Five questions arise from the results of the system. The first question is how to optimize the training process. The second and third questions are whether the functions and features prepared for segmentation and interpretation are effective enough. The fourth question is how to simplify the generated fuzzy rules enough for the users to understand the relations that the neural networks learned. The last question is how to implement not only design knowledge in generating objects but also that in manipulating objects. In addition, neural networks also have some inherent problem in terms of their initial configuration. For example, it is difficult to define the optimum structure such as the number of hidden layers and the number of nodes in each hidden layer.
Therefore, it is necessary to find the solutions for the above questions in the future. In other words, it can be concluded that the next step of this research is to improve the system by combining the technologies of natural language processing, knowledge-based neural computing, design collaboration, and computer network environments, and to develop more practical and effective systems that can represent, store, and manipulate design knowledge in architecture and urban planning.
BIBLIOGRAPHY


Edinburgh, Scotland, Edinburgh University Press, 1971


Henricsson, O., *Analysis of Image Structures using Color Attributes and Similarity Relations*, Institute of Geodesy and Photogrametry, Swiss Federal Institute of Technology (ETH), Zurich, Switzerland, May 1996


Hopfield, J. J., "Neural networks and physical systems with emergent collective computational ability." *Proceedings of the National Academy of Sciences, USA* 79, 2554-2558, 1982


Liu, Y. L., "Encoding explicit and implicit emergent subshapes based on empirical findings about human vision", in *Artificial Intelligence in Design '94*, Eds J. S. Gero, F Sudweeks (Kluwer, Dordrecht) 401-418, 1994


McKeown, D. M., and Denlinger, J. L., "Map-guided feature extraction from aerial imagery", *Workshop on Computer Vision: Representation and Control*, Annapolis, MD, 205-213, 1984


Selfridge, P. G., "Reasoning about success and failure in aerial image understanding, Ph.D. Thesis*, University of Rochester, 1982


Small Blue Planet Atlas Company, “Los Angeles CityRom” (CDROM), 1999


