ÓBUDA UNIVERSITY

A Thesis submitted for the degree of Doctor of Philosophy

SOFT COMPUTING METHODS FOR CLASSIFICATION PROBLEMS IN INTELLIGENT SPACE

Balázs Tusor

Supervisor

Prof. Dr. Annamária R. Várkonyi-Kóczy

Doctoral School of Applied Informatics and Applied Mathematics

March 2021

Members of the Comprehensive Examination Committee:

Members of the Defense Committee:

Date of the defense:

Acknowledgements

First and foremost I would like to express my deepest gratitude to my Supervisor Prof. Dr. Annamária R. Várkonyi-Kóczy for her generous support and guidance. I highly appreciate and acknowledge the ideas, enthusiasm, expertise, time and especially the enormous patience with which she supported me to complete my Ph.D. Thesis.

I gratefully acknowledge the support of the Doctoral School of Applied Informatics and Applied Mathematics, Óbuda University.

I am also thankful to the Hungarian Scientific Research Fund (OTKA K105846 and K78576), for providing the funding which allowed me to undertake parts of this research, as well as the Research & Development Operational Program for the project "*Modernization and Improvement of Technical Infrastructure for Research and Development of J. Selye University in the Fields of Nanotechnology and Intelligent Space*", ITMS 26210120042, co-funded by the European Regional Development Fund.

I would like to sincerely thank the support of my colleagues throughout the years of my research at Óbuda University and J. Selye University, and especially my good friends, Dr. Adrienn Dineva and Dr. Gabriella Simon-Nagy, for their invaluable help during my Ph.D. course.

Lastly, I would like to express my thanks to my family for all their love and support.

Contents

Chapter 1

Introduction

In recent decades, with the spreading of machine intelligence, *"smart environments"* have become a popular tool for humans to collect information, form the environment, get assistance, etc. The Intelligent Space (iSpace) [1] is an intelligent environmental system that offers various services for improving the comfort and safety of everyday life as well as for achieving personalized healthcare and independent living for disabled persons. The main and ultimate goal of such systems is to build an environment that is human centered, comprehends human interaction and satisfies them [2].

The iSpace, which has been developed at the University of Tokyo, is a special intelligent implementation of the Ubiquitous Computing paradigm [3] and a variant of the widely known "Smart Environment" (e.g. "Smart Home") frameworks. Intelligent Space can be any smart area such as a room, railway station, underpass, road crossing, or even a whole town, that is equipped with intelligent sensors and agents. The main feature of the iSpace is that the intelligence itself is not present in the agents but it is distributed in the whole space. The main advantages of this component based architecture are that the iSpace can easily be constructed or modified by the installation or replacement of so-called Distributed Intelligent Networked Devices (DINDs) responsible for monitoring a part of the space, processing the sensed data, making local decisions, and communicating with other DINDs or agents if necessary. Thus, a further advantage of such a system is that the agents in the space do not have to possess any complex logic.

Any room or area can be converted to an iSpace by installing DINDs into it. Although, when building iSpace into an existing area we have to keep in view that the system should be human centered, should not be disturbing for the people who are using it and the installation should not alter the area overly. Another important feature of the iSpace is that it builds models of the environment; it can observe the events and actions taking place in the room or area and is able to react to them if necessary, in order to achieve some kind of change or

give information to the users, help in orientation or anticipate crisis situations. It can also comprehend human interactions: the user can give commands to the Intelligent Space to use certain services. Therefore, the system should be easy to use for the people in it, without the need for them to spend much time learning about how the system is to be used.

1.1 Research Aims and Their Relevance in the Context of the State of the Art

Machine learning is an important field of computer science that automatically creates models (usually from a priori knowledge) in order to implement a pattern recognition or regression function. It has an important role in iSpace applications as well, classification and pattern recognition is essential to process data from various sensors (e.g. video, voice and other sounds, temperature, CO_2 rate, etc.) in order to provide a service (voice recognition, text parsing etc.), tracking the location of robotic agents (task coordination or ability extension) or human users (danger assessment, hand gesture recognition).

The Intelligent Space is a widely researched framework. Numerous new methods and applications had been developed for it since its establishment. Robot control [4] is one of the central areas in iSpace research. [5] has proposed a mapping method for the robotic agents of the iSpace, [6] has developed a human following mobile robot control using the data of distributed sensors, while [7] has proposed an obstacle avoidance method for car-like mobile robots. [8] has described a new virtual iSpace control framework, while in [9] an intercontinental iSpace setup is described that has been created between the Hashimoto Lab at the University of Tokyo in Japan and the Advanced Diagnosis Automation and Control Lab at North Carolina University in the USA, in which a mobile robot in the former laboratory was controlled by a path-tracking controller in the latter one, etc.

Jeni et al. [10][11] and [12] have developed a mobile agent control system using reinforced learning. [13] has created an image based object localization method, then has proposed facial expression recognition system in [14] using near infrared cameras. Yokoi et al. [15] have developed a hand localization method that uses ultrasonic and inertial sensors, while Niitsuma et al. [16] have introduced the design of spatial memory, in which knowledge is assigned to given areas of 3D physical space, which can be recalled on command (by pointing a finger at it). In [17] they have proposed an observation system that can track human activity regarding given objects, then expanded on the idea to develop a method that localizes objects by observing the physical interaction a human user has with it. In [18] this idea has been expanded upon, incorporating human-object interactions in it as well. Subsequently, [19] has extended it with visual and vibration displays.

[20] has created a new method for ethologically inspired human-robot interaction, which has been expanded on by [21], who proposed an attachment behavior model for social robots in the iSpace. [22] has developed an ethologically inspired nonverbal human-robot communication method for ambient assisted living applications. [23] has proposed a human movement profile classifier using Self Organized Maps [24], then has developed a human activity recognition system using conditional fields [25] and particle swarm optimization [26].

As many of the previously mentioned researches show, classification methods can be very advantageously used in iSpace applications. The goal of my research thus far has also been introducing new methods that can extend the toolkit available for the iSpace. For this, I have developed a new hand posture detection method for a gesture-based man-machine interface (Thesis I) in the iSpace; a new clustering method for reducing neural network training time (Thesis II), a general iSpace control system framework (Thesis III) and various classification methods (Theses Group IV-VI) for pattern recognition.

1.2 Organization of the Thesis and Main Directions

The main objectives of my thesis work are the following:

- **Thesis Group I** presents a new hand posture detection method for a gesture-based man-machine interface.

  To reliably describe human hand postures, I have developed so-called *Fuzzy Hand Posture Models* (**Thesis I.1**) that use 14 distinct features. Furthermore, I have introduced a new neural network structure named *Circular Fuzzy Neural Networks* (CFNNs, **Thesis I.2**) for the processing of the 3D coordinate points. CFNNs are a trimmed and fuzzified variant of Artificial Neural Networks (ANNs, [27]). In order to achieve a fuzzy neural network [28], a pair of ANNs is used to approximate the lower and upper bounds of each fuzzy [29] membership function value. To reduce the increased training time requirement that is caused by the doubled complexity of the network, the number of connections between the input layer and the hidden layer neurons are reduced: each input coordinate value is given to only 3 neighboring hidden layer neurons.

  The first and the last hidden layer neurons are considered neighbors as well, hence it is a circular structure. The output of the method is the label of the hand posture model that is the most similar to the detected one, considering its shape. Since the proposed fuzzy hand posture feature set consists of 14 features, three separate CFNNs are used to map the 15 input coordinate triplets into 5-5-4 outputs. The resulting 14 fuzzy

features (represented by the lower and upper interval limits) are processed with a fuzzy inference machine that finds the closest known hand posture to it. The performance of the method has been shown for 6 different hand postures, on which the conducted experiments have shown a 96% classification precision on average.

- Complexity is a very important factor for neural networks, which typically scales with the number of dimensions of the problem. It is a significant problem for the CFNNs as well: the time requirement for training (with the general neural network back propagation algorithm) scales significantly with the number of input samples (considering the high dimensionality of the hand detection problem), even with the reduced complexity gained from the connection trimming.
  In **Thesis II**, to further enhance the training speed of Artificial Neural Networks (such as the Circular Fuzzy Neural Networks), *I have introduced a new clustering method*.
  The proposed method is based on the k-means method, but instead of creating k clusters and assigning every sample into cluster with the nearest mean value, it groups each sample (in the order the data is received) with the rest of the samples that are closer than an arbitrary similarity factor d, and returns the center of each group. The method has been shown to decrease training time significantly, at the cost of a slight decrease in classification precision, depending on similarity factor δ used in the clustering algorithm. Smaller δ values generally results in less training time reduction but also less loss of precision. For example, in one conducted experiment on the 6 hand postures, the initial 120 samples (20 each) had been reduced to only 42, which has decreased the training by 39% but also reduced the precision from 97% to 95.2%. On the other hand, using a slightly smaller δ value has resulted in a 32.8% reduction of training time while no loss of precision. The same clustering method has also been used to calculate the parameters of Radial Basis Function (RBF, [30]) networks.

- In **Thesis Group III** *a new control system framework is proposed for the Intelligent Space*.
  I have also introduced *a new graph-based knowledge representation method* for storing a priori knowledge. It is designed as a graph-based structure, where each individual node represents a concept or action. The nodes are homogeneous, their identification is done through dictionaries that assign labels to each, thus ensuring language independence. The relations between the concepts are described through edges between their respective nodes, making it possible to store and retrieve complex information (the a priori knowledge) in the system. Based on observed

commands given by the user, it creates hypotheses (a posteriori knowledge). Each hypothesis is bound to one or more conditions and each time the system observes the same command in under very similar circumstances (e.g. around the same time of the day) then the reliability factor of the hypothesis is increased. If the given conditions are met for a sufficiently reliable hypothesis (with reliability factor exceeding an arbitrary threshold) then its command is automatically issued by the system. Its effectiveness has been observed in a virtual Intelligent Space where the daily life of one inhabitant is simulated, who gives the iSpace orders according to his or her needs.

- The focus of **Thesis Group IV** is *a new rule-based classifier called Fuzzy Filter Network* based on a modified RBF neural network architecture (**Thesis IV.1**).
  The base idea is that instead of calculating the activated linear combination of the input data (like in the case of an RBF network), the proposed method realizes simple pattern matching by using the radial basis functions for proximity detection, then simply choosing the class or label associated to the pattern (characterized by the center and width parameters of the basis functions) as output. This classifier has the advantage of being very simple to implement, train (by clustering) and modify the trained knowledge (by adding or removing patterns). I have designed a new clustering approach for the training of the fuzzy filter network: the *Analytical Radial Representative clustering* method. The algorithm implements a so-called bottom-up approach, calculating the appropriate cluster radiuses in a single phase. The efficacy of the thus trained network is shown through experiments. Beside the sequential implementation, *I have designed a parallel realization* as well (**Thesis IV.2**), which reduced the complexity of both the training (down from $O(PNm)$ to $O(Nm)$) and the operation (down from $O(PN)$ to $O(P)$) of the classifier (considering N attributes and P samples).

- The focus of **Thesis Group V** is a new lookup table [31] based classifier that is extended with fuzzy logic, called *Fuzzy Hypermatrices* (FHMs, **Thesis V.1**).
  The method maintains multi-dimensional arrays that store the precalculated class labels and fuzzy membership function values for all input attribute value combinations. The input attribute values are directly used to address the array. It achieves a very fast and simple classification, with the downside that it requires a significant amount of memory to operate (scaling with the number of attributes and the size of their domains), thus its applicability is restricted to problems with low dimensionality, such as color filtering (in HSV or RGB color spaces [32]). *I have also developed a parallel implementation of the FHM method* (**Thesis V.2**) that has been

shown to be able achieve a $\sim$5100-7100% speed increase in image processing compared to the non-parallelized FHMs.

- The focus of **Thesis Group VI** is a fuzzy classifier called Sequential Fuzzy Indexing Tables (constant length SFIT or c-SFIT) that uses a sequence of one 1D and N 2D fuzzy lookup tables in to achieve a more memory-efficient classification than FHMs on problems with a constant number (N) of attributes (**Thesis VI.1**).
  It has a layered structure, where each layer contains a lookup table that combines the value of an attribute belonging to that layer to the combination gained in the previous layers. Thus in each layer, the proposed method reduces the size of the problem space in which the class of the input pattern is. This significantly reduces the arrays that are needed to be stored in the memory, compared to FHMs. The performance of the classifier has been tested on benchmark data sets and compared to that of other state of the art classifiers, such as Ordered Weighted Averaging Operators [33] and Neuro-Fuzzy [34] classifiers. The proposed classifier performs with slightly less precision but faster than the other methods.
  I have extended the Sequential Fuzzy Indexing Tables classifier to work on variable length data as well (v-SFIT, **Thesis VI.2**), at the cost of doubling the structure to hold the class markers separately. The proposed methods have also been successfully applied in an iSpace-based dietary assistant application that uses a c-SFIT for storing food items based on their nutrition data and a v-SFIT for storing the names of food items.

The experimental results and complexity evaluation of each presented method are given at the end of their respective chapter.

1.3 Research Methodology

During the course of my research, the new methods have been developed and tested using different computers, of which the configurations are summed up in Table 1.1. These are referenced in the following chapters whenever test results are presented, by their ID. Since the only thing that is unique for each of these particular computers is the amount of RAM, it is used to easily differentiate between them (as well as emphasize the difference in strength of technical advancement). For PC-16 the graphics card is also detailed, because it is the only device I have used for parallel computing research, which uses the GPU to enhance the computational speed of some of the presented methods. The software used to implement said methods have been either MS. Visual Studio (using C++) or Matlab.

Table 1.1: The computer configurations used during my research.

ID	CPU	RAM	Additional Notes
PC-1	*Intel® Pentium® 4 @3.00 GHz*	*1 GB*	*Desktop PC, MS. Windows XP+SP3*
PC-2	*Intel® Core™ 2 Duo, T5670 @1.80 GHz*	*2 GB*	*HP Compaq 6720s, MS. Windows 7 32-bit*
PC-3	*Intel® Pentium® 4 @3.00 GHZ*	*3 GB*	*Desktop PC, MS. Windows XP 32-bit*
PC-4	*Intel® Core™ i5-2450M @2.50GHz*	*4 GB*	*HP ProBook 4540s, MS. Windows 7 32-bit*
PC-8	*Intel® Core™ i5-4590 @ 3.30 GHz*	*8 GB*	*Desktop PC, Windows 7 64-bit*
PC-16	*Intel® Core™ i5-4590 @ 3.30 GHz*	*16 GB*	*Desktop PC, Gigabyte® GeForce™ GTX 960 graphics card with 4GB GDDR5 RAM*

Chapter 2

Hand Posture Modeling for Intelligent Space Applications

Today, computers play an increasing role in our everyday life. The applications of intelligent systems that aim to improve the living conditions and quality of everyday life are also gaining more and more importance. In ideal case, these systems are realized in such a way that the usage of the systems becomes as easy as possible, while the presence of the system does not bother its user at all. This leads to the basic idea of "*ubiquitous computing*", proposed by [3]. One example for ubiquitous computing applications is the *Intelligent Space* (iSpace) [1], which has been developed at the University of Tokyo. ISpace is a room or area that has built-in intelligence: it can monitor the events and actions taking place in the room or area and it is able to react to them.

It has become an essential expectation towards the Intelligent Space to make communication with the environment possible for the user using simple, natural signs. Hand gestures are among these kinds of communication methods. In this chapter, a new hand posture and gesture modeling and recognition system is presented that is able to determine the shape of the detected hand based on the coordinate model computed by the hand detection and tracking system proposed in [35] . Hence, creating an intuitive interface for the iSpace to make possible for the user to communicate with the environment by using hand gestures.

The comfortableness of the way of communication between humans and the iSpace is of vital importance from the point of view of usability of the system. The more natural the interaction is, the wider the applicability can be. Because of this, I decided to use one of the basic human talents, coordinated complex moving of the hands, for communication. The idea is that after that the sensors of the iSpace detect and separate the hands of the humans, the intelligence of the system determines the postures and movements and translates them

to desired actions. In this chapter, I present a hand posture and gesture modeling and recognition system that is able to determine the shape of the detected hand based on the coordinate model computed by the hand detection and tracking system proposed in [35]. Hence, creating an intuitive interface for the iSpace to make possible for the user to communicate with the environment by using hand gestures.

Numerous systems have been developed for hand gesture recognition. Here, I briefly summarize three characteristic approaches which, similarly to the introduced system, use information directly or indirectly obtained from images made by one or more cameras. The system proposed by [36] is based on the idea of divide and-conquer strategy: it breaks hand gestures to basic hand postures and movements, which can be treated as primitives for syntactic analysis based on grammars. The system consists of a two-level architecture that decouples hand gesture recognition into two stages: the low-level hand posture detection and tracking and the high-level hand gesture recognition and motion analysis. For the low-level hand posture detection, it uses a statistical approach based on Haar-like features [37], while for the high-level hand gesture detection it employs a syntactic approach based on the linguistic pattern recognition technique to fully exploit the composite property of hand gestures. The detected hand postures and motion trajectories are sent from the low level to the high level of the architecture as primitives for the syntactic analysis so that the whole gesture can be recognized according to the predefined grammars.

[38] proposes a fuzzy rule-based approach that uses a data glove to recognize hand gestures for the LIBRAS (*Língua Brasileira de Sinais*, *Brazilian Sign Language*) system. The method classifies the given hand postures based on the angles of the finger joints. Each gesture is broken into a list of monotonic segments, which determine a finite state machine that is used to recognize the posture.

In the system proposed in [39] Pseudo 2D Hidden Markov Models (P2-DHMMs) are used for hand gesture recognition. The basic idea is the real-time generation of gesture models for hand gesture recognition in the content analysis of video sequence from a CCD camera. To avoid the problem caused by the exponential complexity of the algorithm of fully connected two-dimension hidden Markov models, the connectivity of the network has been reduced in several ways, thus gaining P2-DHMMs, which retains all of the useful HMMs features. The models can be trained similarly to neural networks.

The fuzzy neural network architecture proposed by [40] is based on incorporating the idea of fuzzy ARTMAP [41] in feature recognition neural networks [42]. The inputs of the neural network consist of fuzzy membership function values, which are determined from the gray level value of each pixels of the monochrome image. A gesture recognition fuzzy neural network is used (with four layers, excluding the input layer) for feature recognition.

The implemented system offers a flexible framework for gesture recognition, and can be used efficiently in scale invariant systems.

This chapter is organized as follows. Section 2.1 describes the most important features and the architecture of the Intelligent Space. Section 2.2 gives a brief overview of the procedure that detects the human hand and produces the input data for the new recognition system proposed. Section 2.3 presents the Fuzzy Hand Posture Model, an intuitive hand posture model that is used for modeling the human hand. In Section 2.4, an overview of the proposed system is given together with the detailed description of its parts and of the functioning of the system. In Section 2.5 an analysis the performance of the new technique and make comparison to other hand gesture recognition methods is presented in Section 2.6. Finally, Section 2.7 summarizes the new results.

2.1 The Intelligent Space

The *Intelligent Space* (iSpace) is a smart environmental system originally developed at the Hashimoto Laboratory (at the University of Tokyo, Japan). Similarly to other *Smart Home* frameworks, the main goal of the Intelligent Space is to build an environment that comprehends human intentions and satisfies them [2]. This means that the system should be easy to use for the people in it: they should be able to express their will through intuitive actions and there should be no need for them to learn how the system is to be used. Beyond supporting humans, the iSpace should provide an interface also to its artificial agents, e.g. to the robots offering physical services to the humans using the iSpace.

The most characteristic feature of the iSpace is that the intelligence is distributed in the whole space, not in the individual agents. Thus, the agents in the space do not require complex logic and sensors, since the Intelligent Space itself monitors the area, evaluates the computations, and gives instructions to the agents.

The system can also react with its environment and provide information or physical services to its users. Another requirement for the system is that it should be human centered, and should not be disturbing for the people who are using it. Furthermore, the installation of the Intelligent Space into an existing area should not alter that area overly. The most characteristic feature of the iSpace is that the intelligence is distributed in the whole space, not in the individual agents. Thus, the agents in the space do not require complex logic and sensors, since the Intelligent Space itself monitors the area, evaluates the computations, and gives instructions to the agents.

The iSpace is an active information space because information can be requested by the clients and provided by the system through active devices. It is thus a so-called soft

environment: it has the capability to adapt itself to its clients [1].

Any room or area can be converted to an Intelligent Space by installing so-called Distributed Intelligent Networked Devices (DINDs) into it. A DIND consists of three main components (Fig. 2.1): a *sensor* which can monitor a part of the space, a *processing unit* that processes the sensed data and makes decisions, and a *network module* that makes possible for the DIND to communicate with other DINDs or with agents of the iSpace via wireless LAN. The advantages of this component based architecture are that the iSpace can easily be constructed or modified by the installation or replacement of DINDs.

ISpace applications, existing and under development, (see e.g [2] and [43]) aim at such tasks as the monitoring of physiological functions of humans, the positioning and tracking of humans, the localization and control of mobile robots, finding paths for them by using itineraries taken by people, etc. A detailed overview of the iSpace can be found e.g. in [1], [43], and [2].

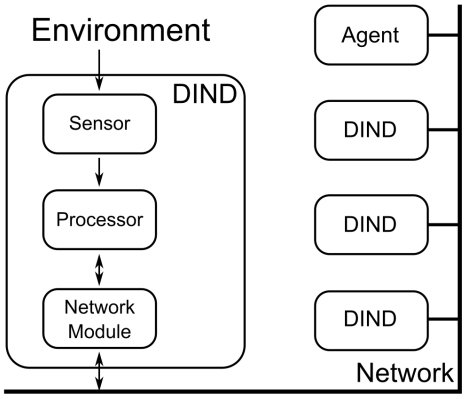


Figure 2.1: The architecture of a Distributed Intelligent Networking Device.

2.2 Hand Detection and Data Preprocessing

This subsection briefly describes the subsystem extracting and classifying various hand postures from the images of a stereo camera pair, which is achieved in two steps: first, the frames of the cameras are processed, yielding a series of three dimensional point locations. Then, these spatial points are processed by a soft computing based intelligent modeling component.

The inputs of the proposed hand posture detection system are produced by the hand tracking and recognition system proposed in [35]. It detects the human hand by using two cameras (Fig. 2.2 *(a)*), which are monitoring the same scene from two different positions and preprocesses the data into a three dimensional coordinate model. The 3D model of the

hand consists of the spatial location of the feature points of the detected hand. The procedure works the following way: Firstly, it locates the areas in the pictures of the two cameras where visible human skin can be detected using histogram back projection [42]:

$$backproj(x, y) = H(hue(x, y)) \tag{2.1}$$

where $backproj()$ means the single channel backprojection image, H is the hue histogram, $hue()$ is the single channel hue plane of the input image and (x, y) are coordinates in the image. This means that to a given color model (histogram) of a given object (human skin), as a result, the probability distribution of skin is obtained.

The next step is the extraction of feature points considering curvature extrema: *peaks* (e.g., fingertips), and *valleys* (such as the roots of the fingers, see [44], [45]). After that, the selected feature points are matched in a stereo image pair. The matching occurs separately for the peaks and the valleys using the fuzzy based matching algorithm proposed by [28]. It works in the following way: First, the candidate pairs (i.e., points lying within a given fuzzy neighborhood of their corresponding epipolar lines) are located. Then, for each candidate point pair detected in the stereo image pair, the sum of the differences of their neighborhoods weighted by a 2-D fuzzy set is calculated:

$$\sum_{x,y \in w, x', y' \in w'} |I(x, y) - I'(x', y')| \cdot \mu_a(x', y') \cdot \mu_b(x', y') \tag{2.2}$$

where $I(x, y)$ and $I'(x', y')$ mean the intensity values in the left and right images, respectively; and μ_a and μ_b denote the membership functions used as weighting functions.

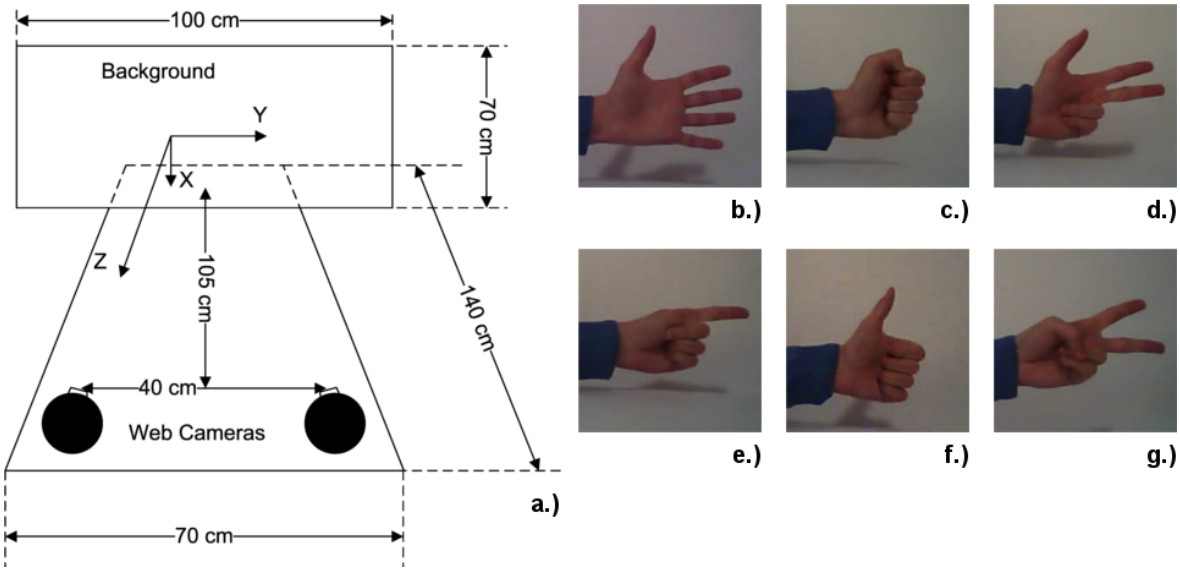


Figure 2.2: The setup of the preprocessing system (a) and the considered hand postures $(b) - (g)$.

The environments of the feature points are denoted by w and w'. The pairing that yielded the smallest sum of the weighted difference is considered to be matching.

This results in a variable number of feature points (depending on the given inputs). Since neural networks in general require a constant number of inputs, further so-called subdivision points are used as well. The subdivision points are determined in the left frame as one or more division points of the sections determined by two consecutive feature points. The subsections have the same length within one section, but not necessarily all of them are of the same length (here, the length of a section means the number of contour points it includes). The number of subdivision points, for each section, is proportional to the length of that section. The total number of subdivision points depends on the number of feature points: These two values, together, should yield a constant number, which equals the number of inputs of the CFNNs. In case of complete lack of feature points, the whole contour is equidistantly subdivided (so the total number of feature and subdivision points would be the desired number of outputs). The subdivision points of the left frame are then matched against contour sections on the right frame, again, using the fuzzy point-matching algorithm [45].

Finally, the three dimensional coordinates of the feature points are calculated using the known camera matrices (for further details see [35]). The result of the procedure consists of 15 spatial coordinate points (a choice made by [35], finding a compromise between using a simple model with too few points (resulting in a lower overall potential classification accuracy) or a more complex model that is more efficient in describing a hand (thus making it easier to differentiate between different postures, but in turn raising the computational complexity as well)). Fig. 2.3 shows an example for the visualized results of the procedure. From the detected spatial coordinate model the system determines the hand posture and recognizes the hand gesture from the identified sequence of hand postures.

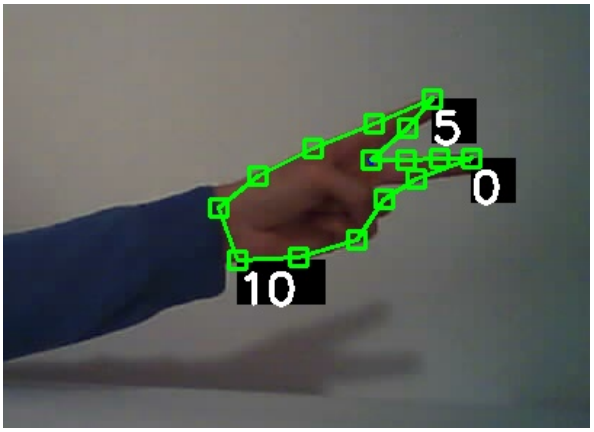


Figure 2.3: The Coordinate Model of a Detected Hand Posture.

It was found that this procedure gives acceptable results if the hand lies apart from the homogeneous background and does not lie too near the camera. (In the first case, the contour might become noisy due to the hand region touching the shadow while in the latter one, peaks and valleys may fail to be detected.)

2.3 Fuzzy Hand Posture Models

In order to efficiently distinguish different hand postures, I have developed *Fuzzy Hand Posture Models* (FHPMs). An FHPM describes the human hand using fuzzy hand feature sets [P. 12].

In order to be able to distinctively identify each individual hand posture, three different types of fuzzy descriptors (each being a fuzzy feature set) had been appointed, each one describing a certain type of features of the given hand posture (Fig. 2.4). The first set consists of four fuzzy features; they describe the distance between the fingertips of each adjacent finger. Both the second and the third sets consist of five-five fuzzy features: former describes how bent is finger, while the latter describes the relative angle between the bottom finger joint and the plane of the palm of the given hand.

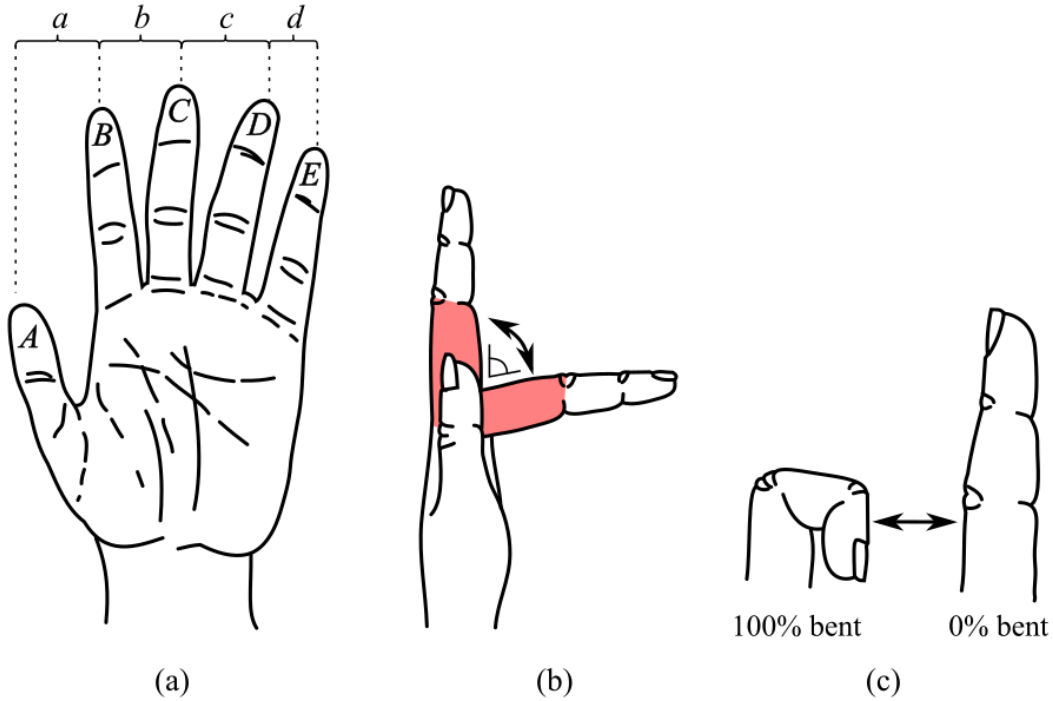


Figure 2.4: The fuzzy descriptors.

Table 2.1: Features for the Hand Posture of "Victory"

Feature group	**Relative distance between adjacent fingers**				**Relative angle between the lowest joint of each finger and the plane of the hand**					**Relative bending angle of each finger**				
Feature	a	b	c	d	A	B	C	D	E	α	β	γ	δ	ϵ
Fuzzy Value	*L*	*M*	*S*	*S*	*M*	*S*	*S*	*L*	*L*	*M*	*L*	*L*	*S*	*S*

Each feature is marked by a linguistic variable that can only have one of the three following values: $small$, $medium$, or $large$. Each feature descriptor group is an answer to a certain question:

(a) How far are the finger X and the finger Y from each other? (4 features: a...d)

(b) How big is the angle between the lowest joint of finger W and the plane of the palm? (5 features: one for each finger (A...E))

(c) How bent is the finger Z? (5 features: one for each finger (α...ϵ))

For each FHPM, 14 linguistic variables (which are used to describe the 14 features introduced above) are stored in the ModelBase. With this model we can distinguish theoretically 3^{14} different hand postures, which is more than enough even for the most complex sign languages.

In the following, an example is presented to demonstrate how a hand posture can be described by the FHPM features: Fig. 2.3 shows the well-known victory sign. The relative position of the fingers from each other and their bentness is clearly visible. From that, we can determine the values of all the 14 linguistic variables. Table 2.1. lists the linguistic values of the features in each of the three feature groups for the hand posture of "victory".

2.4 Fuzzy Hand Posture Recognition

For hand posture recognition, the proposed system uses a specialized fuzzy neural network architecture and fuzzy inference. The idea is based on transforming the coordinate model of the detected hand into a Fuzzy Hand Posture Model by *Circular Fuzzy Neural Networks* and then to identify it by applying fuzzy inference. The proposed system consists of six modules:

- ModelBase
- GestureBase
- Target Generator
- Circular Fuzzy Neural Networks (CFNNs)
- Fuzzy Inference Machine (FIM)
- Gesture Detector

Fig. 2.5 shows the architecture of the system. The system receives the coordinate model of the detected hand as input, transforms it into a Fuzzy Hand Posture Model using CFNNs, then determines the shape of the hand from the calculated FHPM with the usage of the Fuzzy Inference Machine (FIM). The Gesture Detector module observes the output of the FIM, searches for matches with predefined patterns in the GestureBase. The hand gesture is identified in case of fuzzy matching with any stored hand gesture or is refused if the hand gesture is unknown for the GestureBase.

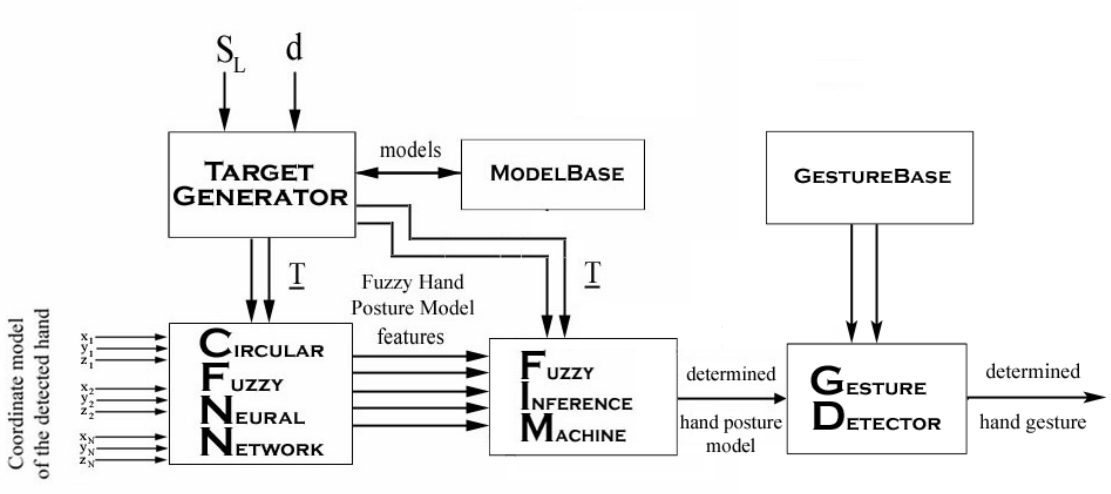


Figure 2.5: Block Diagram of the Hand Posture and Gesture Identification System.

```
<base>
  <model>
    <name> Open_hand </name>
    <value_a> large large large large large </value_a>
    <value_b> small small small small small </value_b>
    <value_c> large large large large large </value_c>
  </model>
  <model>
    <name> Fist </name>
    <value_a> small small small small small </value_a>
    <value_b> medium large large large large </value_b>
    <value_c> medium small small small small </value_c>
  </model>
  <model>
    <name> Three </name>
    <value_a> medium medium medium small </value_a>
    <value_b> small small small large large </value_b>
    <value_c> large large large small small </value_c>
  </model>
  <model>
    <name> Thumb-up </name>
    <value_a> small small small small small </value_a>
    <value_b> small large large large large </value_b>
    <value_c> large small small small small </value_c>
  </model>
  <model>
    <name> Point </name>
    <value_a> medium small small small small </value_a>
    <value_b> large small large large large /value_b>
    <value_c> small large small small small </value_c>
  </model>
  <model>
    <name> Victory </name>
    <value_a> large medium small small small </value_a>
    <value_b> medium small small large large </value_b>
    <value_c> medium large large small small </value_c>
  </model>
</base>
```

```
<base>
     <gesture>
             <name> Classic </name>
             <quantity> 3 </quantity>
             <sequence> Open_hand Fist Three </sequence>
     </gesture>
     <gesture>
             <name> Neo </name>
             <quantity> 3 </quantity>
             <sequence> Point Thumb-up Victory </sequence>
     </gesture>
</base>
```

Figure 2.6: An example for the ModelBase with 6 manually defined models (left) and the GestureBase with 2 manually defined models (right))

The features for each model are stored in the *ModelBase*, as linguistic variables (with possible values: small, medium, large). One of the easiest realizations of the ModelBase can be e.g. using XML files. Fig. 2.6 (left) shows an example for 6 predefined models.

The GestureBase contains the predefined hand gestures. Each hand gesture is identified by its name and is described by up to 5 string values, which are existing Fuzzy Hand Posture Model names. The string values are listed in the sequence parameter. The parameter quantity denotes how many FHPMs the given hand gesture model consists of. Fig. 2.6 (right) shows an example for the realization of the GesturBase.

The *Target Generator* is used to calculate the array of desired target parameters (T) for the Circular Fuzzy Neural Networks and the Fuzzy Inference Machine using the ModelBase. The input parameters of the Target Generator module are:

- d - identification value (ID) of the model in the ModelBase. It can simply be the name of the given model or an identification number.
- S_L - a linguistic variable for setting the width of the fuzzy triangular sets, thus tuning the accuracy of the system.

For faster computation, the implemented version of the system uses triangular shaped fuzzy sets with centers set to the following numerical values: *small* = 0.3; *medium* = 0.5; *large* = 0.7.

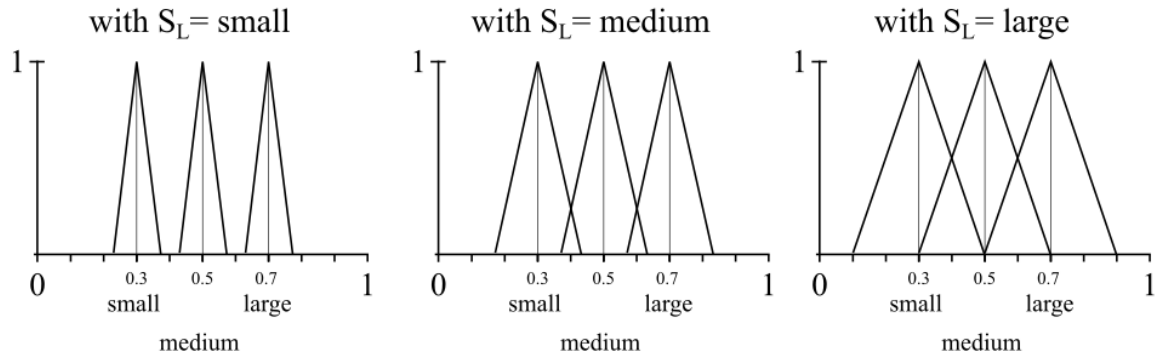


Figure 2.7: The 9 types of considered fuzzy feature sets considering *small*, *medium* and *large* labels for fuzzy set centers and widths.

The other parameter of the fuzzy sets is their *width*. This parameter is adjustable with a linguistic variable S_L. In case of S_L chosen to be small, medium or large, the widths of the fuzzy feature sets equal to 0.33, 0.66, and 1, respectively (Fig. 2.7). Eq. (2.3) is used to compute the ends of the interval that represents the given fuzzy feature set:

$$(a, b) = (center - 0.2 \cdot width, center + 0.2 \cdot width) \tag{2.3}$$

where a, b denotes the lower and upper ends of the interval, $center$ corresponds to the numerical value of the given feature, and $width$ equals to the numerical value of linguistic variable S_L). T is thus a 2×14 array of interval ends (one pair for each feature).

Remark: as it can be seen in Fig. 2.7, not all of the domain is covered with the 3 triangular sets. This introduces a limitation for the system to only recognize models that are similar enough to the known models (the sensitivity of this similarity is set by S_L). Therefore, the output of the system is $unknown$ in cases where the detected hand posture is too different from the stored ones.

For the task of converting the coordinate model of the given hand to a Fuzzy Hand Posture Model, I have developed Circular Fuzzy Neural Networks based on *fuzzy neural networks* with *interval arithmetics* (proposed by [28] (Fig. 2.8), who implemented the fuzzy neural network architecture as a pair of artificial neural networks (ANNs), where one ANN is set to calculate the lower and of the fuzzy intervals while the other one is set for the upper ends).

The networks have fuzzy numbers in their weights, biases, and in the output of each neuron. Each fuzzy number is represented by the interval of its alpha cut at value 0, i.e. by the base of the triangular fuzzy set. Thus, each fuzzy number is determined by two real numbers: the lower and the upper ends of the given interval.

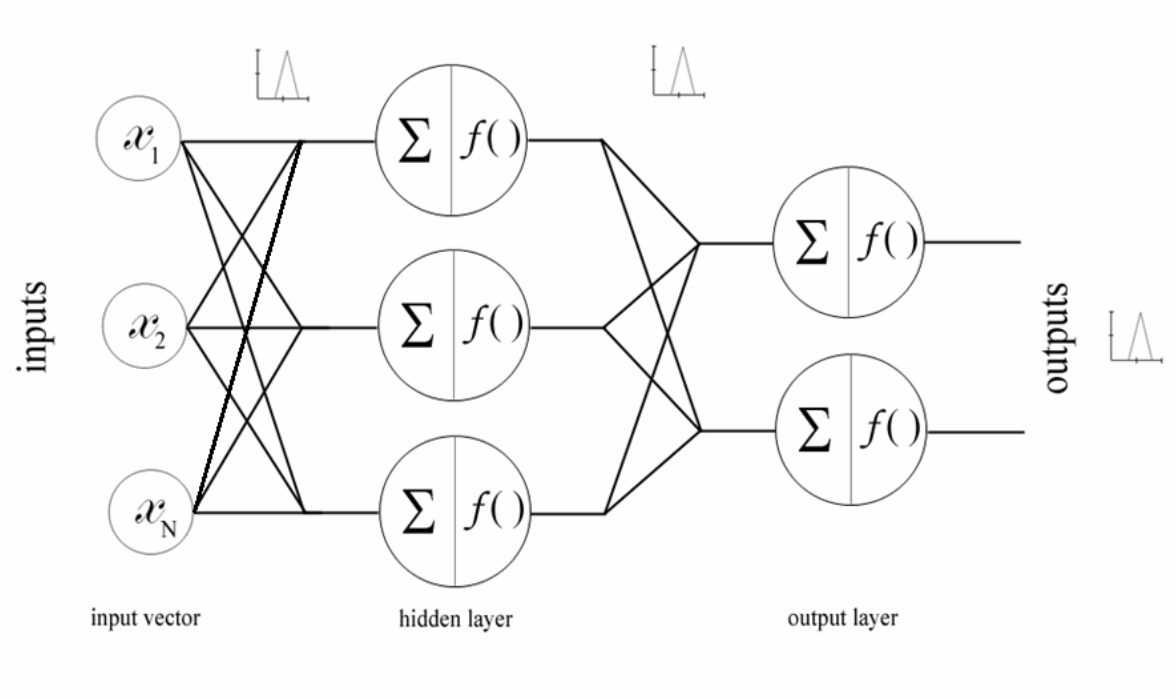


Figure 2.8: The topology of Fuzzy Neural Networks.

The novelty of the CFNN network lies in its modified, circular topology. In order to enhance the speed of the training session and to take advantage of the fact that the input data consists of coordinate triplets, the topology of the network has been modified. Originally, each input was connected to all hidden layer neurons. In the modified network topology only 9 inputs (corresponding to 3 adjacent coordinate triplets) are connected to each. This way every hidden layer neuron processes the data of 3 adjacent coordinates. The topology between the hidden layer and the output layer neurons has not been changed. This way, the network has been realigned into a circular network, see Fig. 2.9. For the sake of better visibility not every connection is shown. In the outer circle layer the input coordinates can be found, in the middle circle the hidden layer neurons are placed, and in the inner circle the output layer neurons of the network are located.

Since the coordinate model consists of 15 three dimensional coordinates, the fuzzy neural networks have 45 inputs. In order to increase the robustness, instead of using one network with 14 output layer neurons, three different networks are used with the only difference in the number of the output layer neurons. The first network computes the first group of fuzzy features, having 4 neurons in the output layer. The second and third networks have 5-5 neurons in the output layer. Furthermore, all the networks consist of one hidden layer with 15–15 neurons.

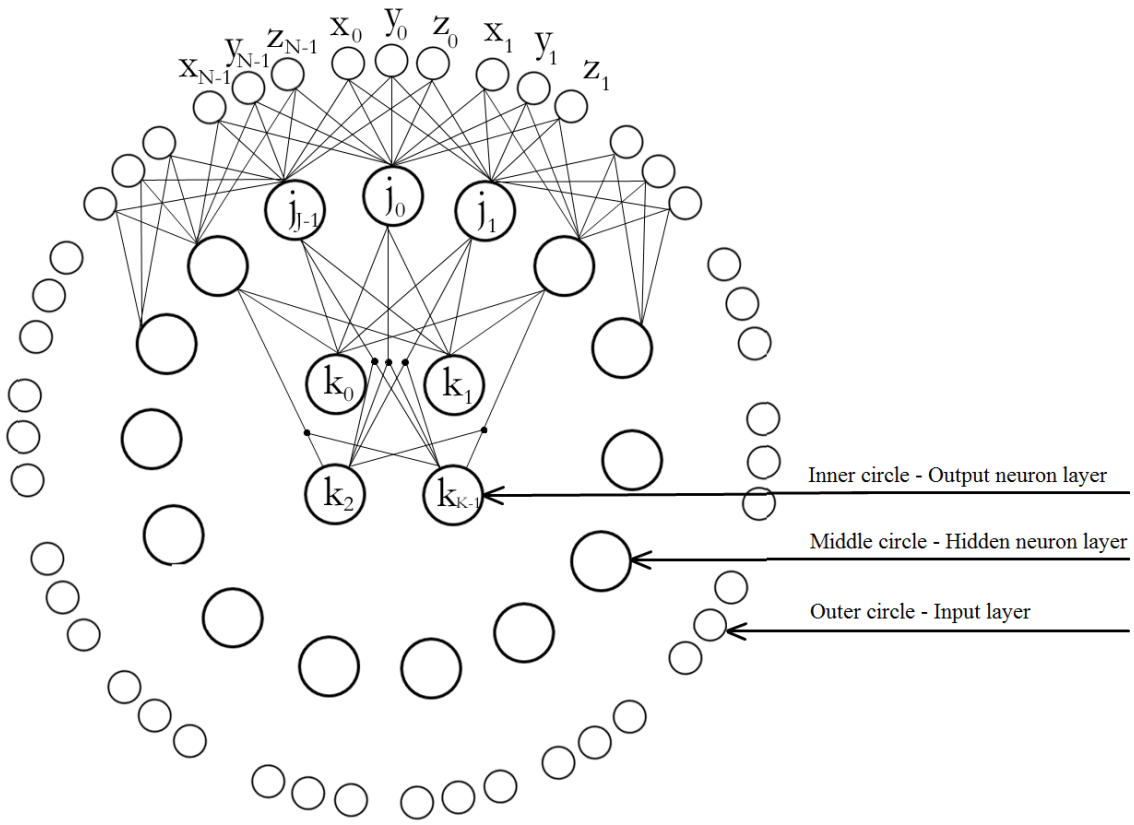


Figure 2.9: The Topology of the Circular Fuzzy Neural Networks.

These reductions cause a dramatic decrease in the required training time, while the precision and accuracy of the networks are not affected.

2.4.1 Fuzzy Reasoning-based Classification

The last step of the hand posture identification procedure is the fuzzy reasoning based classification. This part compares the output of the CFNNs to all fuzzy hand posture models stored in the ModelBase and chooses the model that corresponds the most to the model presented by the CFNNs. The algorithm works as follows: For each model in the ModelBase, the intersection value between their fuzzy feature sets and the fuzzy feature sets of the given FHPM is calculated, thus we gain β_i for each of its features. The minimum of β_i shows the correspondence ratio between each model and the given FHPM. The maximum value of the correspondence ratios indicates which model is the most corresponding to the detected hand posture.

The ID of each recognized hand posture model is put in a queue, which is monitored by the Gesture Detector module. It searches for hand gesture patterns predefined in the

GestureBase, and in case of matching it gives the ID (in my implementation: the name) of the detected hand gesture as the output of the system, i.e. it identifies the gesture. The reliability of the identification system can be improved by applying a "fault tolerant" sign language, i.e. when there is a $d_H > 2$ Hamming distance among the meaningful hand posture series. In this case, a detected unknown hand posture sequence caused by false detection(s) can be corrected to the nearest known hand gesture.

2.4.2 The Training Session

The training of fuzzy neural networks with interval arithmetics is done with the general back-propagation algorithm [46], with slight modifications to account for the decreased connections [P. 13].

Furthermore, in order to increase the speed of the training of the CFNNs, I have developed a new training procedure. In order to decrease the quantity of the training samples, instead of directly using the training data in the training phase, they are first clustered and the CFNNs are trained by the centers of the obtained clusters (that are still 45 real numbers or 15 spatial coordinate points).

The clustering procedure is based on the k-means method with a simple modification: comparing a given sample to the clusters one by one, the sample gets assigned to the first cluster with the distance of its center to the given sample less than an arbitrary value. If there is no such cluster then a new cluster is appointed with the given sample assigned to it. The clustering is used on only one type of hand model at one time, so the incidental similarity between different type of hand models will not cause problems in the clustering phase. The algorithm for this phase is described in more detail in Chapter 3.

2.4.3 FHPM-based Hand Recognition

The experimental options for the training session of the CFNNs were the following:

- Learning rate (back-propagation): 0.8
- The coefficient of the momentum method (back-propagation): 0.5
- The error threshold (back-propagation): 0.1
- S_L (target generator): small
- Clustering distance: 0.5

Table 2.2: The results of training 6 different type of hands using preclustered training data

Name of the Hand Posture Model	Quantity of Chosen Samples	Cardinality of the Clustered Training Set	Cardinality of the Training Dataset	Correctly Identified	%
Open hand	*20*	*10*	*80*	*75*	*93.75*
Fist	*20*	*12*	*80*	*78*	*97.5*
Three	*20*	*4*	*80*	*78*	*97.5*
Thumb-up	*20*	*7*	*80*	*78*	*97.5*
Point	*20*	*4*	*80*	*78*	*97.5*
Victory	*20*	*5*	*80*	*68*	*85*

2.5 Experimental Results

During the conducted experiments, the effectiveness of the training was investigated on PC-2, with the implementation done in Microsoft Visual Studio 2005, using C++. Table 2.2 summarizes the identification results of the experimental system which is able to differentiate among six hand models (Open hand, Fist, Three, Thumb-up, Point, and Victory) [P. 5][P. 1][P. 14]. For better visualization, Fig. 2.2 shows these hand postures. The set of the generated coordinate model samples has been split into two separate, training and testing, sets. The training set consists of 20 samples from each of the three FHPMs.

The clustering procedure reduced the quantity of the training sets, thus instead of 120 only 42 coordinate models were trained to the CFNNs. The networks are trained with choosing S_L = small. The FIM could identify the models in the testing set (with 80 samples of each models) with an average accuracy of 94.8%. The testing set does not contain the training set.

While the CFNNs were trained with S_L = small value, in the inference phase S_L = large has been used, which increased the accuracy of the Fuzzy Inference Machine.

2.6 Evaluation and Applicability

As it was shown, the correct identification rate proved to be 85-97.5%, with the '*Victory*' sign being the hardest to learn (85%), and the other considered hand postures being recognized at an even higher rate. While the classification accuracy is high, the method has two main disadvantages: even with the reduced complexity, the CFNNs take up to 6.5 hours to train (considering the experimental equipment). This can be reduced with the added clustering step, though that at best halved the necessary time (further analysis on the effects of the clustering step on the speed of the training is done in the next chapter, in Subsection 3.3.2). One solution to this problem is the usage of parallel computing techniques, which have became more advanced since this research have been done, and thus constitutes a possible future upgrade of the system. Even so, the training phase can be done offline and thus the training time is not a significant problem considering the main purpose of the system (that is hand posture identification).

It is worth discussing the computational complexity of the CFNNs and the FIM, as they are the main processes in the proposed hand posture recognition method (excluding the preprocessing system developed by [35]). As it has been already mentioned, the reason for the cropped connections between the input and the hidden layer, aside from aiming for a more localized data processing, is complexity reduction and subsequently, making the training and evaluation phases of the networks faster, compared to standard MLPs. The time complexity of the operation of MLPs (trained with the Backpropagation (BP) algorithm) can be derived from their general architecture [47]: For each sample, the weighted sum of the inputs (N) are taken for all hidden layer neurons (J), then the same is done with the output of all hidden layer neurons in the output layer. Thus, the time complexity can be expressed as $O(P \cdot (N \cdot J + J \cdot K))$. Whereas, the CFNNs have $O(P \cdot (n \cdot J + J \cdot K))$ complexity, where n is the number of inputs connected to the hidden layer neurons. Since this n is small by design ($n = 3 << N = 45$), the complexity is reduced to $O(P \cdot (3 \cdot J + J \cdot K)) \subseteq O(P \cdot J \cdot K)$.

The training (from the BP algorithm [46]) uses an evaluation phase for all samples, for as many epochs (ϱ) as necessary to reach the desired error threshold. Thus, the complexity is $O(\varrho \cdot P \cdot J \cdot K)$, in contrast to the $O(\varrho \cdot P \cdot (N \cdot J + J \cdot K))$ of the MLPs.

Finally, the time complexity of the FIM is $O(P \cdot N \cdot m)$, where m is the number of hand posture models, as it is simply doing a linear search through the models to see which one is the most similar to the model generated by the CFNN from the inputs.

A significant advantage of the presented method overall is that it does not require expensive equipment like data gloves (unlike systems like [38]), though it does need two

cameras (which are still cheaper than a data glove) set up in a rig and precalibrated for stereo vision (which can be automatized). Another advantage is that it works with few training samples, as the experiments have shown where the system could use 20 random samples from each hand (which was further reduced by the clustering step) was able to learn to correctly classify ~85-97.5% of the rest of the samples.

The advantage of using FHPMs as shape descriptors is that they are intuitive, making it easy for a human administrator to define hand models. Unlike other widely used region-based descriptors (derived from geometrical moments of the hand region, e.g. *Hu invariants* [48] and Zenike moments [49]), FHPMs are contour-based (similarly to *Fourier Descriptors* [50], where the features are derived from the contour using Fast Fourier Transform). Although FHPMs are theoretically rotation invariant like Zenike moments and Hu invariants, the proposed itself system is not (meaning that instances of the same hand with different orientation seem to be different models), due to the preprocessing step (the generation of the 3D point sequence). On the other hand, the system is scale invariant considering the hand models, as the preprocessing step scales the detected 3D point-series into a uniform frame. Another significant disadvantage of the system comes from the limitations of the image processing system that only works before homogeneous backgrounds, and it is presumed that the hand is fully visible on both input images or not visible at all, and that the hand is being steady. A further assumption is that stretched fingers are well separated and that the hand stays parallel to the cameras. The dependence on the rotation is also the consequence of using CFNNs, as each input has a fixed position in the network (as classical neural networks are generally not independent on the ordering of the input parameters).

2.7 Summary of New Results

In this chapter, I have presented **a new hand posture detection method** that can be used in a gesture-based man-machine interface. In order to be able to distinctively identify each individual hand posture, I have proposed a **new fuzzy feature set** that can describe hand postures with 14 features that can be designated into 3 feature groups.

The method receives the data from an image processing system that uses two cameras and a Fuzzy Matching Algorithm to calculate the 3D coordinates of the shape of the hand and outputs 15 spatial feature points. The set of these feature points are processed with a **new neural network structure named Circular Fuzzy Neural Network**, mapping the 45 numerical values onto 14 intervals (the 14 fuzzy features). The output of the neural networks is identified using fuzzy inference.

The output of the proposed hand posture recognition method is the label of the hand posture model that is the most similar to the detected one, or $unknonw$ if there is none that is similar enough. The performance of the method has been shown for 6 different hand postures, where **the correct identification rate proved to be 85-97.5%**, with the '*Victory*' sign being the hardest to learn (85%), and the other considered hand postures being recognized at an even higher rate (>93%).

2.7.1 Thesis Group I.

2.7.1.1 Thesis Statement I.1.

I have proposed a new fuzzy hand posture model that uses a fuzzy feature set consisting of 14 features over 3 feature groups. I have also developed a new hand posture detection method for the identification of hand postures. The input of the method is provided by an image processing system that uses two cameras and a Fuzzy Matching Algorithm to calculate the 3D coordinates of the shape of the hand. The 3D coordinates are processed by a set of Circular Fuzzy Neural Networks that map the coordinates into a set of 14 triangular fuzzy sets. The acquired fuzzy sets are processed by a fuzzy inference machine that provides the output of the method: the label of the hand posture model that is the most similar to the detected one. The performance of the method has been shown for 6 different hand postures.

Related publications: [P. 1], [P 5], [P. 12], [P. 13], [P. 14]

2.7.1.2 Thesis Statement I.2.

I have developed a new neural network structure named Circular Fuzzy Neural Network. The structure of the new network is derived from interval-based Fuzzy Neural Networks by trimming the number of connections between the input and hidden layer neurons in order to achieve a faster yet more localized processing (one hidden layer neuron only processes the output of a given number of neighboring input layer neurons). The other significant difference comes from the circular shape of the new network, as the first neuron in each layer is regarded as a neighbor of the last neuron in the same layer. The new classifier is used to convert a set of 3D coordinates into 14 fuzzy features. It has been validated for 6 different hand postures.

Related publications: [P. 1], [P 5], [P. 13], [P. 14]

Number of independent citations as of May 2021, according to Google Scholar: 87

Chapter 3

A New Clustering Method for the Acceleration of the Training of Artificial Neural Networks

Many engineering problems involve some kind of machine learning tasks. *Clustering* is one such task, in which the goal is to find a structure in a collection of unlabeled data. The procedure assigns a set of objects into groups whose members are similar in some way and are dissimilar to the objects belonging to other groups (so called clusters). Clustering can be defined as the probably most important unsupervised learning problem and in most cases it leads to a (usually iterative) multi-objective optimization task. Many difficulties may occur during the clustering, caused by e.g. multi-dimensional spaces, time/data complexity, finding an adequate distance measure, non-unambiguous interpretation of the results, overlapping of the clusters, etc.

Such problems, like probability estimation, feature selection, feature extraction, data mining, statistical data analysis, pattern recognition, general machine learning etc. can strongly be related to clustering. Classification methods like *Artificial Neural Networks (ANNs)* and *Radial Basis Function Neural Networks (RBFNNs)* can also significantly profit from a preclustering step before training, either by gaining a more compact representation of the training data so the necessary training time can be reduced (which is beneficial for ANNs), or even information about the structure of the classifier (e.g. the RBFNNs).

In this chapter, first the problem of structural complexity is stated in Section 3.1 in regards to ANNs and RBFs. Then in Section 3.2 the proposed clustering algorithm is presented, while experimental results are presented in Section 3.3 for three different neural network architectures: Feed-forward Artificial Neural Networks (Subsection 3.3.1), Circular Fuzzy Neural Networks (Subsection 3.3.2), and Radial Basis Function Neural

Networks (Subsection 3.3.3). Finally, in Section 3.4 the applicability and time complexity of the proposed method is presented. Finally, Section 3.5 summarizes the new results.

3.1 The Issue of Structural Complexity in Neural Networks

3.1.1 Feed-forward Artificial Neural Networks

Feed-forward Artificial Neural Networks (*FFANNs* or simply *ANNs*) are systems based on the operation of biological neural networks. They are networks built of many simple processing units (neurons) arranged into two or more layers. Each unit can be connected to all units in the next layer. The most important ability of ANNs is that they are able to find relationship (structure) among the elements of unstructured data sets and can learn complex functions from input/output data-pairs, which means that they can adjust their parameters using iterative training algorithms in order to achieve accurate input-output mapping. Another advantage is that they are relatively easy to implement in any application.

The utilization of ANNs is widespread, covering possibly the whole engineering field. Although, besides the advantages, a big drawback of their usage is that they usually need a significant amount of time to be trained, which scales with the structural parameters of the networks and the quantity of input data. Although, this can be done offline; the training has a non-negligible cost, and further, can cause a delay in the operation. CFNNs proposed in Chapter 2 solve this by reducing the structure itself, but do so at the cost of potentially lowering the learning ability as well if the problem requires a higher number of neurons.

3.1.2 Radial Basis Function Neural Networks

Radial Basis Function Neural Networks (*RBFNNs* or simply *RBFs* [51][30]) are feed-forward artificial neural networks with two active layers, which use the linear combination of radial basis functions that are used as activation functions. Their application areas mainly involve function approximation, time series prediction, control, etc.

The general architecture of RBFs can be seen in Fig. 3.1. The network first calculates the activation value ($g_i(X)$) from the input data (X) using radial basis functions, each function having a center (c_i) and a width (σ_i) parameter of the base function (which usually differ for each neuron). In the output layer, the weighted sum is generated from the values resulted in the previous step and their weight parameters (w_i) (with an optional additional bias value (g_0), which is usually 1), resulting in the response of the network (y).

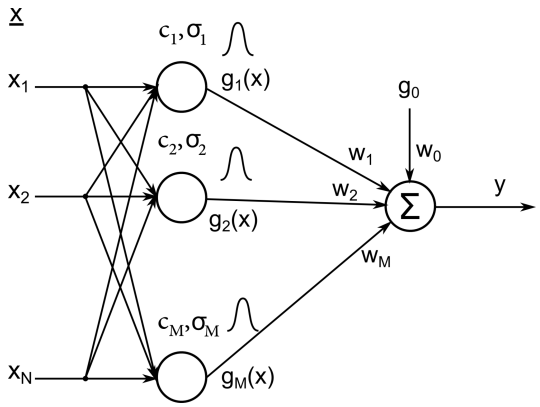


Figure 3.1: The architecture of Radial Basis Function Neural Networks.

In the training phase of RBFs the hidden layer realizes an unsupervised training through nonlinear mapping using radial basis functions, to tune the center and width parameters of each function. A widespread solution for this is using clustering methods, e.g. the k-means method [52]. The output layer realizes a supervised training phase, e.g. the back-propagation algorithm.

Like that of the ANNs, the utilization of RBFNNs is widespread nowadays. And also similarly to ANNs, the determination of the optimum structure for the networks is not trivial, particularly the number of neurons that are required to solve the given problem. Even though the c-means method is effective for determining the center parameters of each neuron, the number of clusters is needed to be set by the users themselves. The clustering algorithm proposed in this chapter is not only able to calculate the center parameters, but it can also give the number of neurons thus the very structure of the network.

3.1.3 The Problem of Structural Complexity

When solving complex problems with neural networks, one of the biggest problems is the approximation of the necessary complexity of the NN model. It is well known that it has to majorate the complexity of the system or problem to be modeled which usually leads to a situation where the complexity of the model significantly exceeds that of the problem. This may result in a complexity explosion and serious problems in the fitting and/or evaluation of the model. The main problem is that there is not any real systematic method for the estimation of the needed (minimum) complexity. Engineers usually apply one of two strategies: The first one starts of a small model and increases its complexity iteratively if it turns out to be insufficient. The other method follows the opposite direction: It starts of a model with overestimated complexity which is reduced step by step by deleting unnecessary parts.

The problem leads back to Hilbert's 13th [53] (and also to his 23th) conjectures ("The existence of a continuous function of three variables which cannot be decomposed as the finite superposition of continuous functions of two variables" and "Further development of the calculus of variations", respectively) [54] and to the works of Kolmogorov [55] and his student Arnold [56], [57]. Arnold disproved Hilbert's 13th conjecture in 1957 by showing a constructive evidence to the problem while Kolmogorov proved a general representation theorem in the same year stating that any continuous real-valued N variable function defined over the [0,1]^N compact interval can be represented with the help of appropriately chosen 1 variable functions and sum operation.

The universal approximation theorem has come into the focus again with the spreading of soft computing, fuzzy, and neural network techniques. It has been proven that if an arbitrary function is approximated by a set of functions of a certain type then the universal approximation property holds only if the number of building functions (fuzzy sets or hidden neurons) is unlimited (see e.g. [58],[59],[60]).

In this chapter, I introduce a new data preprocessing procedure for NNs used for classification that can be used to the training (and to determine the structure) of RBFs. According to the experiences derived from the conducted experiments, the developed method increases the speed of the training of the three considered neural networks: FFANNs, RBFNNs and CFNNs. The technique applies the following idea: instead of directly using the training data in the training phase, the data is clustered first and the ANNs are trained using only the centers of the obtained clusters. This results in that the amount of training data and by this the necessary training time can be decreased significantly, thus speeding up the learning process of these NN models (by reducing the complexity through the creation of a more compact representation of the training data).

3.2 The Clustering Algorithm

The main idea of the new training procedure is that instead of directly using the training data in the training phase of NNs, the data is clustered and the NNs are trained by using the centers of the obtained clusters.

Fig. 3.2 shows the block diagram of the general supervised learning scheme extended with a clustering step. The goal of the training is to tune the model in order to make the output of the model (y) approximate the desired output (d) of the examined unknown system using the value (c) determined by the criteria function (typically a function of the approximation error). The model input is the cluster center (u') of the cluster which the actual input (u) belongs to.

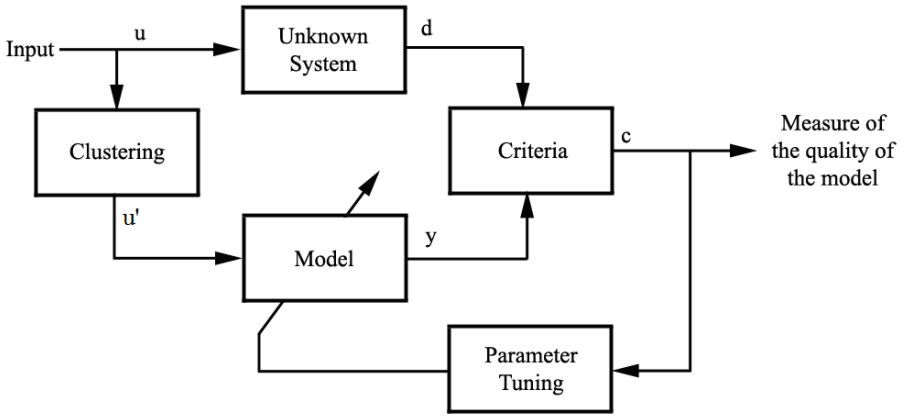


Figure 3.2: The new supervised learning scheme extended with clustering.

The purpose of the clustering step is to create a more general representation of the input data, in an attempt to reduce the number of training samples and the effect of noise.

The role of the inserted clustering step is to reduce the quantity of the input data (u) used during the training and parallel with it to preserve the information contained in the original data set as much as possible, thus making the time required for learning the mapping of the unknown system to be modeled shorter while preserving the accuracy of the systems performance as if it was trained with the original data set. The results of the clustering step are the centers of the appointed clusters (u').

The principle of the new technique is similar to the algorithm of that of the well-known k-means clustering method [52], but while in the original k-means algorithms the samples are compared to all of the existing clusters (k clusters are randomly assigned from the training data) and the sample is assigned to the best fitting (nearest) cluster; in my method, the given sample is compared to the existing clusters one by one and the procedure stops if a "near enough" cluster is found. I.e., the sample gets assigned to the first cluster where the distance between the cluster's center and the given sample is less than a predefined, arbitrary value (distance factor). If there is no such cluster then a new cluster is appointed with the given sample assigned to it.

Fig. 3.3 shows the flowchart of the algorithm. The clustering is used on only one class at one time, so the incidental similarity between patterns of different classes will not cause any problem during the training.

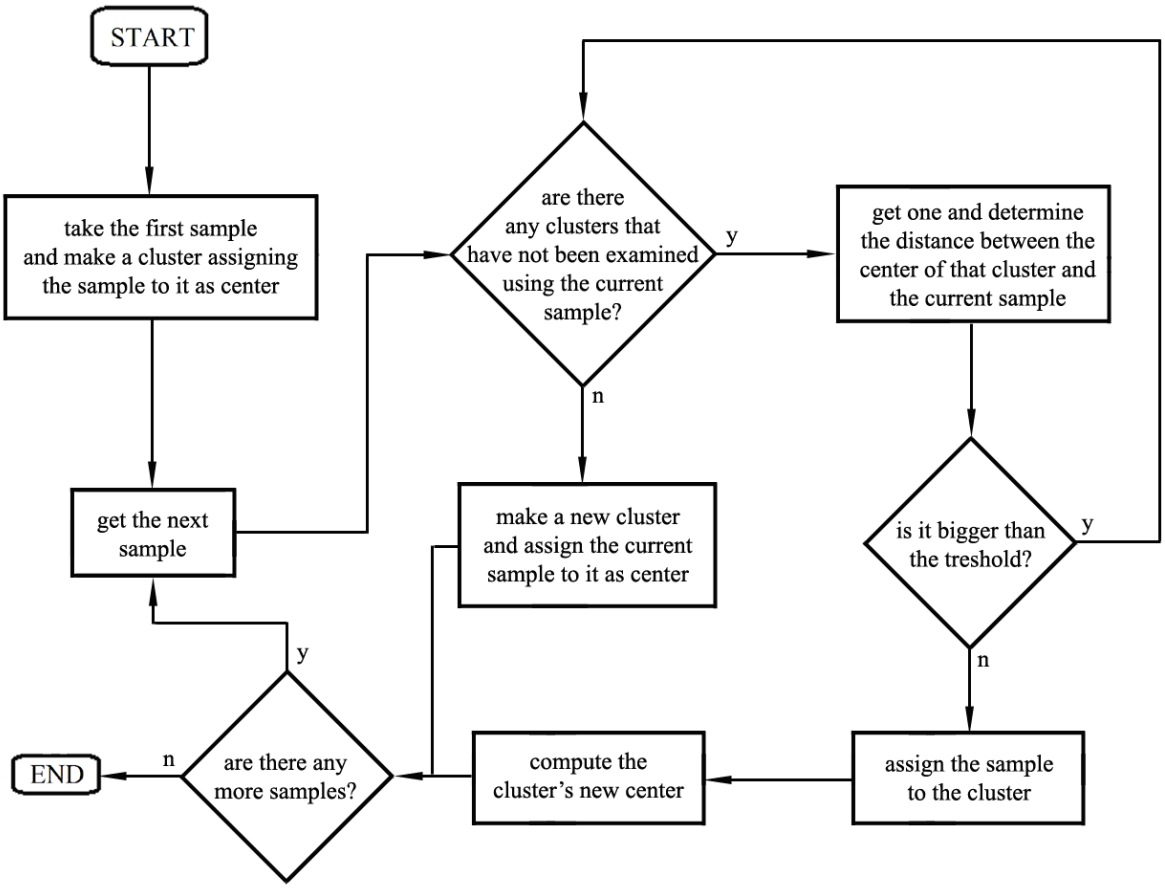


Figure 3.3: The algorithm of the time reduced clustering step.

The most important parameter of the new clustering method is the clustering distance (δ). Its value has a direct effect on the complexity of the training and on the accuracy of the classification. Choosing it too low results in too many clusters (nearing the complexity of the original training data set) while big clustering distances result in a fewer number of clusters, but less accurate classification.

3.3 Experimental Results

The effectiveness of the new training and clustering technique applied in various neural network models has been measured by the speed increase achieved in the training session and the classification accuracy (described by Eq.(A.6) in the Appendix) of the thus trained networks. Three series of experiments have been conducted, in which ANNs, RBFs and CFNNs have been used.

3.3.1 Experiments Regarding ANNs

The effectiveness of the new training and clustering technique applied in feed-forward neural network models has been measured by the speed increase achieved in the training session. The experiments have been conducted for ANNs who had to learn to classify 2 dimensional inputs. Four different networks have been trained and compared in each experiment: one using the original data set and three using clustered datasets (generated by different clustering distances). The original data set in all cases consists of 500 samples (though the classes do not necessary contain exactly the same amount of samples; however their ratio has been set to more or less equal). After the training, each network has been tested on a separate test data set (not used during the training) containing 1000 samples.

For easier reference, let us denote the three clustered data sets (generated by different clustering distances) used in all experiments by A_i, B_i, and C_i (clustering distance for A_i equals 0.05, for B_i: 0.1, and for C_i: 0.25; where i stands for the number of the experiment) and the cropped data sets by A'_i, B'_i, and C'_i, accordingly (e.g. A'_i is cropped so that it has as many samples taken from the original classes by chance as A_i does).

Both ANN experiments have been conducted on PC-3. The experimental options for the training session of the ANNs in all experiments have been set to the following:

- Learning rate: 0.8
- Coefficient of the momentum method: 0.1
- Number of hidden layer neurons: 10
- Error threshold: 0.01 (except for networks A_3 and A'_3, where 0.05 was used)

In the following, three typical examples, representing different problem complexities, are presented.

3.3.1.1 A Simple Problem: Easily Separable Classes

The first ANN example covers a relatively simple problem where three easily separable, non-overlapping classes are to be separated. In Fig. 3.4 *(a)* the original, unclustered data set can be seen. The first neural network (with parameters corresponding to the description above) has been trained with this (unclustered) data set. Fig. 3.4 *(b)* shows the responses of the trained network on the whole input domain [0..1]×[0..1], by using a resolution of 0.05. As it can be well seen, the trained network can solve the problem successfully and is able to classify the key regions correctly.

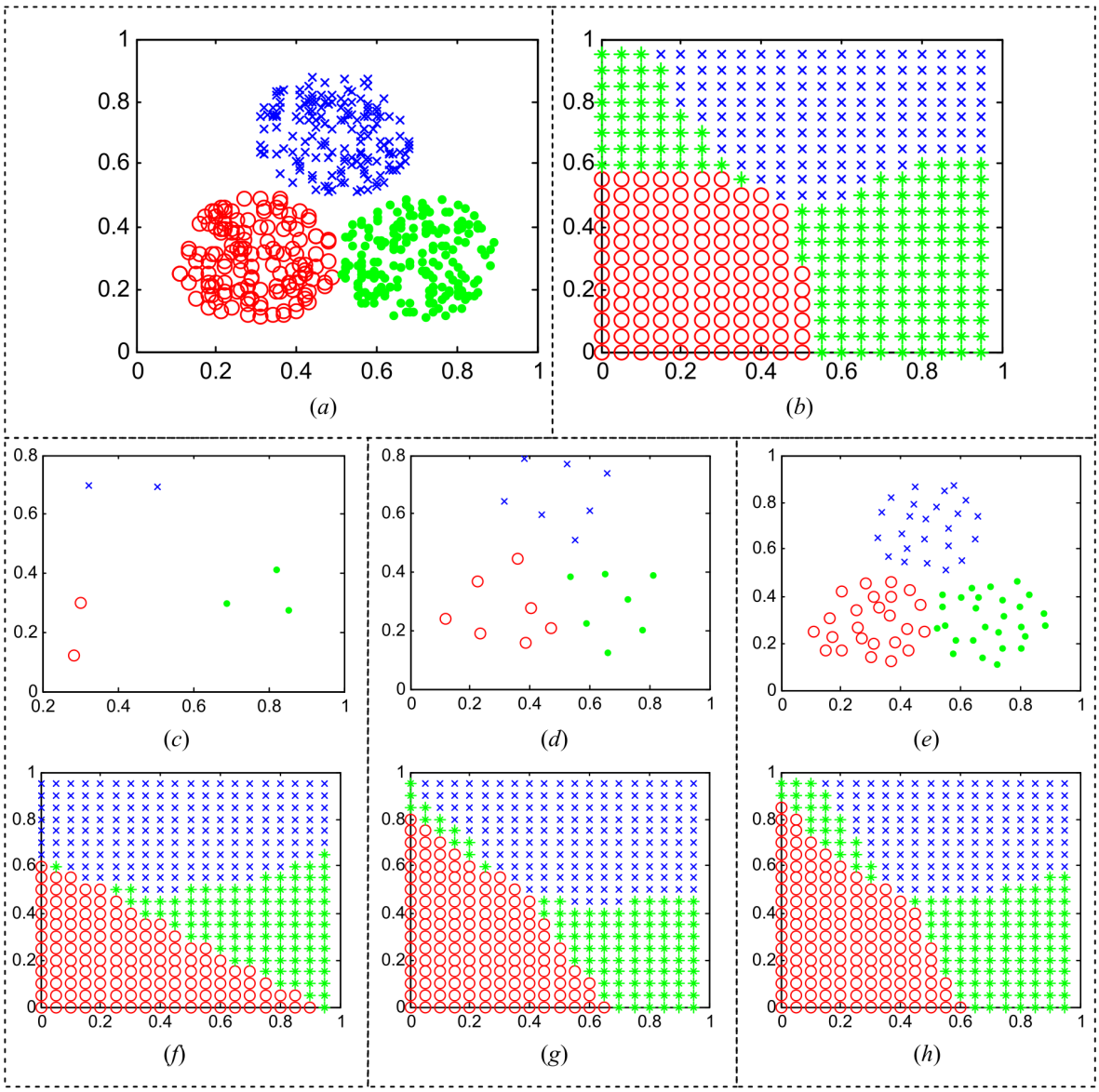


Figure 3.4: The original training data set *(a)* of the 1st ANN experiment, The performance of the network trained with the original training data set on the whole domain *(b)* and the centers of the resulted clusters along with the classification results of the networks trained with the clustered training data sets on the whole domain *(c), (d), (e)*. Figures *(f), (g), (h)* right under them present the responses of the networks trained by the corresponding clustered data, over the training domain.

Table 3.1: The Relative Speed Increase and Classification Accuracy of the ANN Networks using Data Preclustered with Different Clustering Distances (δ) in Experiment 3.3.1.1.

Network		**No. of Centers**	**Req. Time for Training**	**Relative Speed Increase**	**Classification Accuracy on the Unclustered Data**	**Classification Accuracy on the Testing Dataset**
Unclustered		*500*	*2 min. 38 s*	-	-	-
Clustered	$\delta = 0.05$	*75*	*44 s*	*359%*	*499/500 (99.8%)*	*997/1000 (99.7%)*
	$\delta = 0.1$	*21*	*22 s*	*718%*	*481/500 (96.2%)*	*956/1000 (95.6%)*
	$\delta = 0.2$	*7*	*6 s*	*2633%*	*450/500 (90%)*	*898/1000 (89.8%)*

Fig. 3.4 *(c),(d),(e)* shows the acquired cluster centers got by using the three, above defined clustering distances on the input data. As it is expected, the clustering done with the smallest clustering distance results in the most and the clustering done with the largest clustering distance results in the least number of clusters (and thus cluster centers). To provide a better opportunity for the comparison, Fig. 3.4 *(f),(g),(h)* presents the responses of the networks trained by the clustered data, over the training domain.

Table 3.1 shows the number of clusters, as result of the clustering, the time required to train the neural networks, and the speed increase relative to the time required to train the network with the original, unclustered data set. It also shows the accuracy of all the trained networks on the test data set. Compared to the original 500 data points, the clustering that uses the largest clustering distance results in only 7 clusters (and thus 7 cluster centers), while the clustering using the smallest clustering distance results in 75 clusters. The network trained with the original data set can achieve 100% accuracy, while among the networks trained with the clustered data sets, the worst accuracy equals 89.8% (in case of the largest clustering distance), and the best accuracy reaches 99.7% (in case of the smallest clustering distance). The training using the original data set needs 2 minutes and 38 seconds, while the training of network C_1 takes only 6 seconds (2633% speed increase or 96% decrease in training time compared to the original). B_1 takes 22 seconds (718% speed increase or 86% decrease in training time) and A_1 takes 44 seconds (359% speed increase or 72% decrease in training time). The table also shows the classification accuracy of the trained networks over the training and the test data sets. Out of the three results, A_1 proves to be the most accurate, achieving almost 100% accuracy on both data sets.

The original training data set *(a)* of the 1st ANN experiment, The performance of the network trained with the original training data set on the whole domain *(b)* and the centers of the resulted clusters along with the classification results of the networks trained with the clustered training data sets on the whole domain *(c), (d), (e)*.

3.3.1.2 A More Complex Problem: Slight Overlapping

The second ANN example presents a similar structure but more complex problem as is shown in the previous one. In this case, the problem includes 5 clusters. Fig. 3.5 *(a)* shows the original, unclustered data set. The training of the neural network (with parameters corresponding to the description above) proved to be unsuccessful in this case, as well; the problem is too complex for the used network with the given parameters.

Fig. 3.5 *(b),(c),(d)* presents the acquired cluster centers, similarly to the previous two examples and Fig. 3.5 *(e),(f),(g)* shows the responses of the trained networks on the whole domain. Table 3.2 lists the number of resulting clusters, the time required to train the neural network, and the accuracy of the trained networks measured over the test data set. Despite of the failed attempt of training the ANN with the original data set, the network has again successfully learned the classification when the three clustered data sets have been used. The C_3 network achieves an accuracy of 96.6% in 53 minutes and 11 seconds and B3 can also achieve a relatively high, 89.8% accuracy in less than 3 minutes. For comparison, Table 3.2 also shows the accuracy figures of the networks on the original training data set as well as on the test data set. It is interesting to remark that C_3 can achieve even better results on the test data set (97.8%), while B_3 proves to be only slightly less accurate (96.7%). Further, despite of A_3 providing only 71.6% accuracy on the training data set, it achieves a 82.3% accuracy over the test data set.

Table 3.2: The Relative Speed Increase and Classification Accuracy of the ANN Networks using Data Preclustered with Different Clustering Distances (δ) in Experiment 3.3.1.2.

Network		**Number of Samples**	**Required Time for Training**	**Classification Accuracy on the Unclustered data**	**Classification Accuracy on the Testing Dataset**
Unclustered		*500*	*Could not learn*	-	-
Clustered	$\delta = 0.05$	*134*	*53 min. 11 sec.*	*483/500 (96.6%)*	*978/1000 (97.8%)*
	$\delta = 0.1$	*44*	*2 min. 39 sec.*	*443/500 (89.8%)*	*967/1000 (96.7%)*
	$\delta = 0.2$	*16*	*27 seconds*	*359/500 (71.6%)*	*823/1000 (82.3%)*

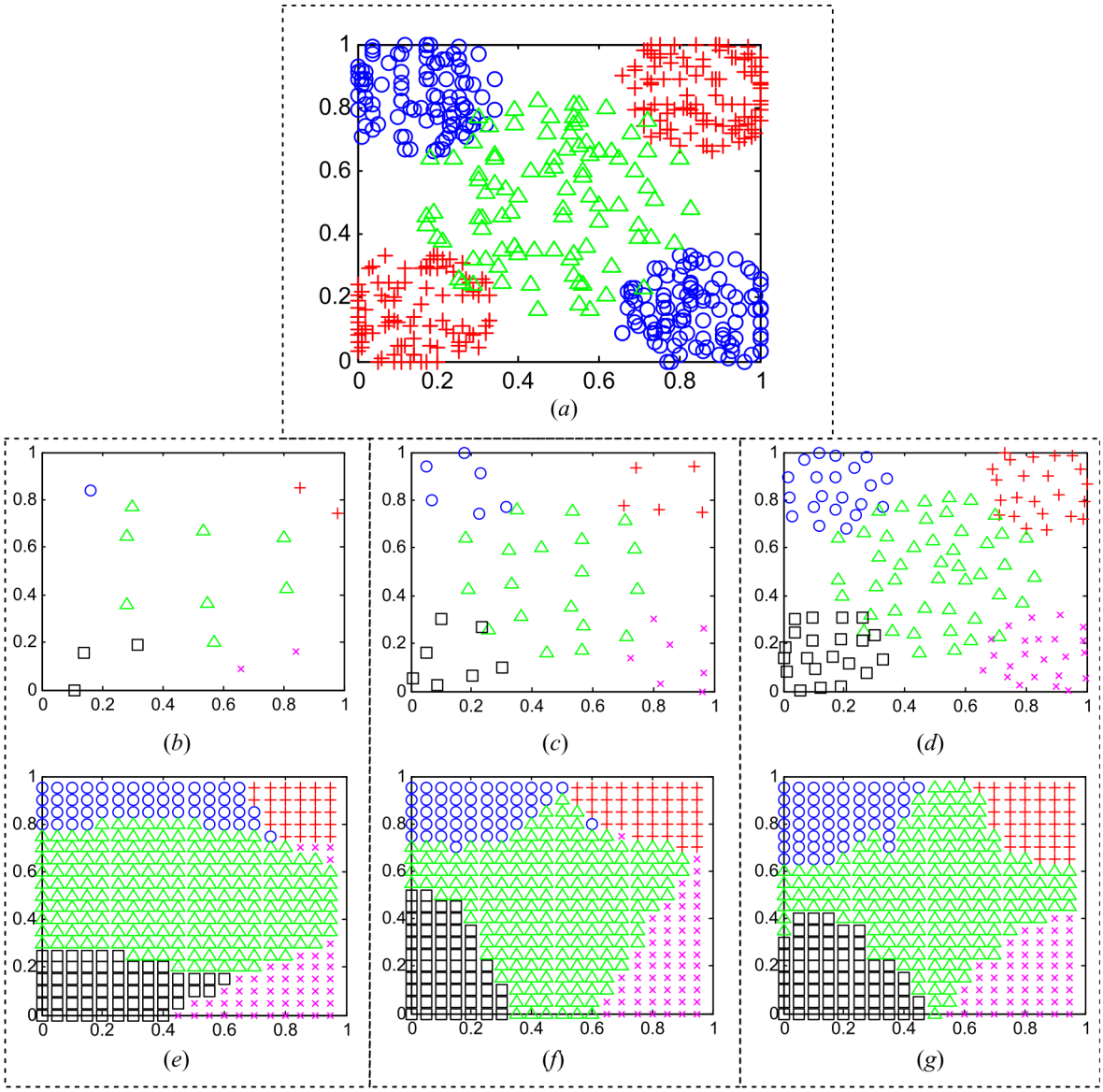


Figure 3.5: The original training data set *(a)* of the 2nd ANN experiment, and the centers of the resulted clusters along with the classification results of the networks trained with the clustered training data sets on the whole domain *(b), (c), (d)*. Figures *(e), (f), (g)* right under them present the responses of the networks trained by the corresponding clustered data, over the training domain.

3.3.2 Experiments Regarding CFNNs

In the experiments regarding CFNNs, the system has to learn to identify the six different hand postures seen in Chapter 2 (for better visualization, Fig. 2.2 $(b) - (g)$ shows these hand postures: *"Open hand", "Fist", "Three", "Point", "Thumb-up"* and *"Victory"*).

The effectiveness of the new training and clustering technique can be measured by the speed increase achieved in the training session. Two sets of CFNNs are trained: one using the unclustered sample set and one using the clustered sample set. (A set of CFNNs consists of the 3 CFNNs (A, B, and C), described in the previous section.) Two experiments (clustered and unclustered training) have been conducted; the networks use the same input samples in both cases. The distance factor δ is set to 0.5 during the clustering step.

The experimental options for the training session of the CFNNs in both experiments have been set to the following:

- Learning rate: 0.8
- Coefficient of the momentum method: 0.5
- S_L: small

While the CFNNs have been trained with S_L = *small* value, in the inference phase S_L = *large* has been used, which increased the accuracy of the Fuzzy Inference Machine.

The whole input data set consists of 600 samples of the previously mentioned 6 different hand postures, each hand posture having 100-100 samples. 20-20 samples are selected for each hand posture forming the unclustered training set, which thus consists of 120 samples. However, after the clustering step this is reduced to 42 samples, which is still capable of representing the original 120 samples during the training without sacrificing too much accuracy in the performance of the networks. The testing set consists of the 480 samples that are not used for training.

The first two experiments have been conducted on PC-1, while the third experiment has been conducted on PC-2.

3.3.2.1 General Time Reduction (Incremental Training)

In the first CFNN experiment, the CFNNs are trained in multiple phases incrementally. The training phases differ only in the error threshold parameter that indicates how much each CFNN needs to reduce the error of the training. Since the initial error in this example is about ~0.5 and the desired error e is 0.12, that interval has been divided into 3 sub-phases.

Table 3.3: Time Required for Error Threshold Intervals (hours:minutes) and the Relative Speed Increase Between the Unclustered and Clustered Cases (e.g. Network A with Unclustered and Clustered Data).

	Network type	Time Required for Error Threshold Intervals (hh:mm) and Relative Training Speed Increase (%)		
		$e \in [0.5 - 0.25]$	$e \in [0.25 - 0.15]$	$e \in [0.15 - 0.12]$
Unclustered	A	*0:28*	*1:56*	*2:24*
	B	*0:50*	*2:14*	*2:28*
	C	*0:53*	*0:52*	*2:40*
Clustered	A	*0:16 (∼175%)*	*1:39 (∼117%)*	*1:18 (∼185%)*
	B	*0:32 (∼156%)*	*1:03 (∼213%)*	*1:01 (∼243%)*
	C	*0:31 (∼171%)*	*0:46 (∼113%)*	*0:58 (∼276%)*

Table 3.3 shows the time required for each training phase for both networks that used unclustered and clustered input samples using different error thresholds (e). The reduction ratio of the required time is included in the result of the networks that uses clustered input data. The biggest reduction happens usually in the last phase of the training, resulting in ~234% speed increase on average, while the average speed increase for the whole training session considering all regarded networks is ~183%.

3.3.2.2 General Time Reduction (Non-Interrupted Training)

In the second CFNN experiment, the three networks are trained in a single phase (one for each feature group). In Table 3.4. a detailed comparison is shown based on typical runs of the training of CFNNs that are applying the two types of teaching (i.e. using unclustered and clustered data). In the first type of experiments the decrease in the required time equals 44.4% (~180% speed increase), while the accuracy of the networks is decreased by 2.9% (as using the unclustered data the system misclassifies 9 samples out of 480, while using the clustered data it misclassifies 23 samples out of 480). In the second session the decrease in the required time is 32.5% (~148% speed increase), while the accuracy of the networks is decreased by 2.3% (with the unclustered data the system misclassifies 9 samples out of 480, while with the clustered data it misclassifies 20 samples out of 480). The biggest reduction in training time results in a speed increase of ~207%, while the smallest results in ~122%. The results also suggest that using only one phrase makes the time required for training shorter than when using multiple phases to reach the same error threshold.

Table 3.4: The Required Training Times for each Feature Group G_i focusing on Error Threshold Range 0.5-0.12 and the Relative Speed Increases (the Unclustered Cases Being the Reference Times).

	Feature Group	**Required Time (hours:mins)**	**Relative Speed Increase**
Unclustered	A	*4:27*	*∼207%*
Clustered		*2:09*	
Unclustered	B	*3:08*	*∼132%*
Clustered		*2:22*	
Unclustered	C	*4:05*	*∼122%*
Clustered		*3:21*	

3.3.2.3 The Effect of Clustering Distance

In the third CFNN experiment the training of the CFNNs has been done on multiple data sets coming from clustering the same training data set (that has been used in the previous experiments) using different clustering distances. Table 3.5. shows the quantity of clusters resulted from using 3 different δ clustering distances (0.5, 0.4 and 0.35). The results evidently show that the clustering distance is inversely proportional to the number of clusters resulting from the clustering. While δ = 0.5 results in 42 clusters, δ = 0.4 results in 55, and δ = 0.35 results in 65 clusters. It is also noticeable that because of the clustering, the number of samples of some hand posture types can be reduced significantly (e.g. hand posture "*Three*" can be represented by 4 or 5 cluster centers instead of 20 samples).

Table 3.6 shows a comparative analysis about the time requirement of the training of the CFNNs with the data sets resulting from the clustering for different distances.

Table 3.5: The Number of Clusters Resulted from Multiple Clustering Steps using Different Clustering Distances.

δ	**Number of clusters for each hand type**						
	Open hand	**Fist**	**Three**	**Point**	**Thumb-up**	**Victory**	$\sum$
UC	*20*	*20*	*20*	*20*	*20*	*20*	*120*
$\delta = 0.5$	*10*	*13*	*4*	*7*	*4*	*5*	*42*
$\delta = 0.4$	*13*	*16*	*5*	*9*	*5*	*8*	*55*
$\delta = 0.35$	*13*	*17*	*5*	*12*	*10*	*8*	*65*

Table 3.6: Comparative Analysis on the Characteristics of the Differently Clustered Datasets, Considering the Decrease in Training Time and Classification Accuracy (*ACC*; Using the Unclustered Case as Reference).

	Measured attribute	**Measured value**	**Difference in ratio**
Unclustered	***Total time spent on training (h:mm)***	*6:30*	-
	Average ACC	*97%*	-
$\delta = 0.5$	***Total time spent on training (h:mm)***	*3:57*	*39% decrease*
	Average ACC	*65.2%*	*1.8% decrease*
$\delta = 0.4$	***Total time spent on training (h:mm)***	*4:22*	*32.8% decrease*
	Average ACC	*97%*	*0% decrease*
$\delta = 0.35$	***Total time spent on training (h:mm)***	*5:46*	*11.1% decrease*
	Average ACC	*97%*	*0% decrease*

Table 3.7: Detailed Information About the Classification Accuracy of the Differently Clustered Datasets (i.e. the Number of Correctly Identified Cases for each Hand Posture).

Hand	**Clustering Distance**			
Posture Type	***UC***	***0.5***	***0.4***	***0.35***
"Open hand"	*77/80*	*76/80*	*76/80*	*76/80*
"Fist"	*72/80*	*77/80*	*76/80*	*76/80*
"Three"	*78/80*	*74/80*	*79/80*	*80/80*
"Point"	*80/80*	*77/80*	*78/80*	*79/80*
"Thumb-up"	*80/80*	*78/80*	*80/80*	*78/80*
"Victory"	*79/80*	*75/80*	*77/80*	*77/80*
Average (in ratio)	*97%*	*95.2%*	*97%*	*97%*

It can be seen that while δ = 0.5 results in a 39% decrease in the required training time for the price of 1.8% decrease in the classification accuracy, δ = 0.4 and δ = 0.35 results in 32.8% and 11.1% decrease in the required training time, respectively. However, in the last two cases the classification accuracy is not decreased at all. This suggests that it is possible to achieve a speed increase with little or no loss considering classification accuracy.

Finally, Table 3.7 shows detailed information about the classification accuracy of the

differently clustered data sets (UC stands for unclustered). The systems taught by the clustered data are even more accurate for some hand posture types than the system trained with unclustered input data. Each field of the table shows how many of the hand posture samples were correctly identified out of how many samples. These experiments show that δ = 0.4 can be an ideal choice for the clustering distance in the clustering step.

3.3.3 Experiments Regarding RBFs

While in ANNs and CFNNs the main role of the clustering step was to reduce the training data with little or no loss regarding its ability to represent the original, unclustered data (thus creating a compressed data set), in the last set of experiments the proposed method is also used to determine the structural complexity of RBF neural networks (i.e. the number of hidden layer neurons).

In the experiments the RBF implementation (in Matlab) described in [61] has been used, which utilizes a single-step analytic learning method. The simplified description of the operation of the algorithm is the following. First, for every center (in each neuron) the algorithm calculates the distance from each input data points, and then calculates the Gaussian values using the distances, thus getting N by C matrix ϕ, where N is the number of data samples and C is the number of neurons (centers). Finally, the method produces the weight matrix by dividing matrix ϕ by the matrix of the target data (D). (For more information, see [61].)

In the following, three characteristic experiments are presented for RBFs. The training data consists of 500, while the testing data consists of another 1000 randomly chosen data points. In the experiments various amounts of RBF centers are applied, according to the amount of centers that the clustering step results in.

3.3.3.1 No Overlapping Classes

In the first RBF experiment, the data points of 5 different classes are laid out according to Fig. 3.6(a). There is a big set of data in the middle of the domain (marked with green). Another crescent moon shaped data patch (red) can be found near to class 1. Two further small classes black and magenta) are in two of the corners of the domain, while the rest of the domain is filled with the elements of the last, fifth class (blue). In this simple case, there is not any overlapping among the classes.

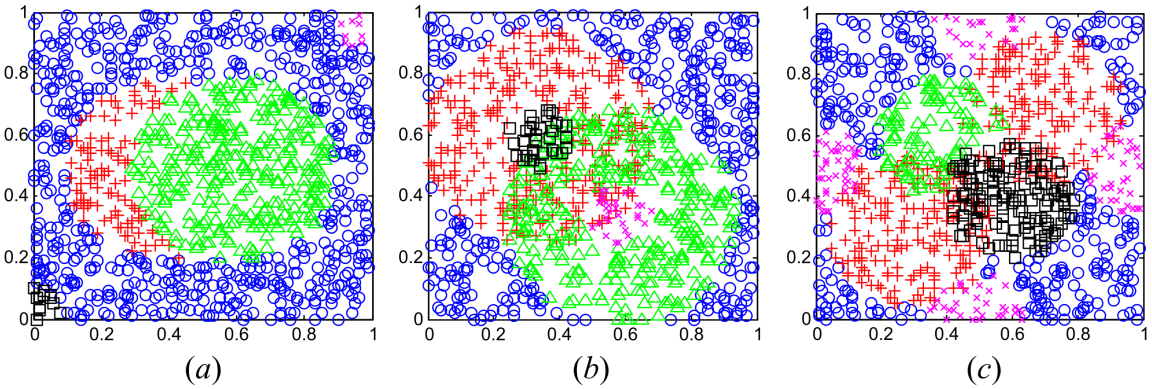


Figure 3.6: The domain of the problem in the three experiments: (a) with no overlapping, (b) with some overlapping and (c) significant overlapping of the class areas.

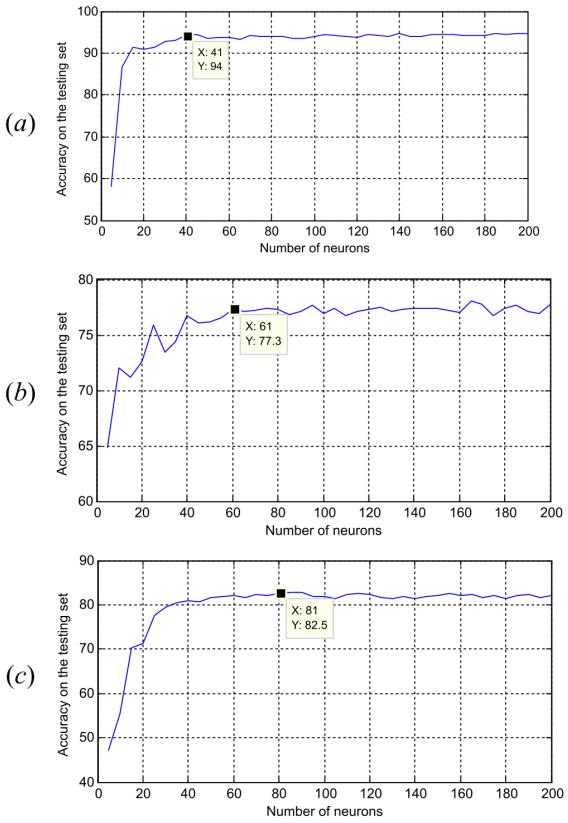


Figure 3.7: The relationship between the complexity and the classification accuracy of the network for the three experiments.

The relationship between the complexity and the *classification accuracy* of the network (measured on the testing data set) has been investigated. In Fig. 3.7(a) the obtained accuracy figures are depicted for RBFs built of different numbers of neurons. In these cases, the

initial parameters of the functions are chosen randomly. Corresponding to the result of the data analysis, the accuracy more or less reaches its peak around 40 neurons. The marked point shows the point indicated by the clustering results (δ=0.14), which seems to be a good choice concerning complexity and accuracy issues. It can also be seen that this particular implementation of RBF is not sensitive to over-learning, which usually occurs in case of too many neurons in case of other RBF architectures.

3.3.3.2 Some Overlapping in Classes

The second RBF example presents a much more difficult problem. The data points of 5 different classes are laid out according to Fig. 3.6(b). Two ring shaped bigger sets of classes (red and green) can be found in the middle of the domain having significant overlapping. (Overlapping can occur for many reasons. E.g., there can be other dimensions in the data that are either unknown or are too expensive to be dealt with.) The interiors of the rings form two non-overlapping classes (black and magenta) with smaller quantities of data. The rest of the domain is filled with the data points of the fifth class (blue).

The relationship between the complexity and the classification accuracy of the network (measured on the testing data set) has been investigated in this case as well (see Fig. 3.7(b)). The accuracy starts to be stabilized around 60 neurons. The marked point shows the point indicated by the clustering results (δ=0.12), which can be viewed as quasi-optimum in terms of the complexity and accuracy trade-off of the network. Although, looking at the results 40 neurons seems to be a good choice as well, Fig. 3.7(b) clearly shows that despite the acceptable accuracy of the case, the network is not stable yet from accuracy point of view.

3.3.3.3 Significant Overlapping in Classes

In the third RBF example, the data points of 5 different classes are laid out according to Fig. 3.6(c). Similarly to the second experiment, there is significant overlapping among the different classes.

Table 3.8 shows the classification accuracy of the trained RBF networks in case of different clustering distances for all 3 RBF experiments (the data from the first, second and third experiment is labeled A, B and C, respectively), as well as the number of neurons resulted from the clustering step. As it can be seen, the accuracy of the model on the training and on the (separate) testing data sets correlates well. It can also be followed how the number of neurons (resulted from the clustering step) is changing in contrast to the clustering distance (marked with red font color). The optimum clustering distance is mostly dependent on the layout of the original data points, so (at least in theory) it can be determined by analyzing the layout of the data points.

Table 3.8: The Number of Resulting RBF cluster centers Considering Different Clustering Distances, and the Classification Accuracy of the Trained RBF Networks Considering Different Clustering Distances (in percentages).

Clustering Distance δ	**Number of RBF Centers**			**Accuracy on the Training Set (%)**			**Accuracy on the Testing Set (%)**		
	A	B	C	A	B	C	A	B	C
0.4	*11*	*11*	*12*	*88.2*	*70*	*77,4*	*86.6*	*72.4*	*73.2*
0.3	*17*	*18*	*16*	*91.4*	*68.2*	*75*	*91.9*	*70.9*	*69.1*
0.2	*24*	*28*	*29*	*91.4*	*73*	*84,2*	*91.8*	*72.9*	*79*
0.15	*38*	*41*	*N/A*	*93.4*	*80*	*N/A*	*92.9*	*77.2*	*N/A*
0.14	***41***	*45*	*N/A*	***95.2***	*79.2*	*N/A*	***94***	*76.2*	*N/A*
0.13	*45*	*56*	*N/A*	*94.8*	*79.6*	*N/A*	*93.8*	*76.9*	*N/A*
0.12	*57*	***61***	*N/A*	*96.4*	***80.4***	*N/A*	*94*	***76.9***	*N/A*
0.11	*65*	*74*	*N/A*	*96.8*	*80.4*	*N/A*	*93.4*	*76.7*	*N/A*
0.1	*74*	*87*	***81***	*96.4*	*81.2*	***88.6***	*93.5*	*77.4*	***81.6***
0.05	*191*	*201*	*193*	*96.8*	*81*	*88.8*	*94.6*	*77.2*	*81.2*
0.02	*399*	*N/A*	*N/A*	*97*	*N/A*	*N/A*	*94*	*N/A*	*N/A*

By applying this optimum clustering distance, a good approximation of the optimum complexity of the RBFNN can be achieved. In this example, it is around 0.14, resulting in 41 clusters and more than 94% accuracy which nears the best obtainable less than 95% accuracy of the model.

Remark: for training set C, the experiment has been run on fewer in-between steps, which is why the table has some missing data values. The results, however, are similar to the results of the first experiment, though in this case the accuracy is not as high, because of the significant overlapping of two classes.

The relationship between the complexity and the classification accuracy of the network (measured on the testing data set) has been investigated in this case as well. Fig. 3.7(c) shows that the accuracy rises sharply at first and then slows down to finally be stabilized around 80 neurons. The marked point shows the point indicated by the clustering results (δ=0.1), which can be viewed as quasi-optimum in terms of the complexity and accuracy trade-off of the network. This example illustrates that the obtainable accuracy can slightly be better when starting of the clustered initial centers than when the initial parameters are

randomly chosen.

3.4 Evaluation and Applicability

According to the efficiency analyses, in case of CFNNs, the introduction of the preclustering step has reduced the training-time in average by more than 32% together with a reduction of the number of input samples by approximately 54%. (The difference between CFNNs and regular feed-forward ANNs is that the weights, biases, and outputs of CFNNs are fuzzy numbers (based on Fuzzy Neural Networks proposed in [28]), further their topology is realigned to a circular topology, and the connections between the input and hidden layers are trimmed.)

The time complexity of the clustering algorithm is $O(N \times P \times k)$ where k is the number of resulting clusters. In the worst case where δ is chosen to be too large, k approaches P and the complexity is $O(N \times P^2)$. Therefore, if the task is to enhance the training speed of a classifier by making a compression of the training data, then it is recommended to experiment with different k values to find a balance between information loss from the compression and the time reduction.

The advantage of the proposed approach over the k-means clustering method is that we do not need to know the suitable number of clusters (k) which is a necessary an input parameter of the k-means algorithm, but rather, the proposed algorithm automatically presents k as part of its output. On the other hand, some level of a priori knowledge is necessary about the data itself, so range parameter δ (the sensitivity of the clustering) can be set.

3.5 Summary of New Results

In this chapter, I have presented a new data preprocessing procedure as **a new supervised training method** for NNs (used for data classification). In order to achieve that, **I have proposed a new clustering method**. The clustering step produces a compression of the input data which in turn reduces the complexity of the neural network training (as the latter is heavily dependent on the amount of training data) and thus **reduces the training time**, along with a certain level of decrease in classification accuracy. The clustering step can be tuned with distance factor δ.

The proposed method has been investigated for three different NN structures:

- For *Feed-Forward Artificial Neural Networks* the relation between the relative training speed increase and the loss in accuracy has been investigated. As expected, there is a trade-off between the two quantities, although as shown in

Experiment 3.3.1.1, the speed increase can be as much as 359% of the original training speed at the cost of a classification accuracy rate of 0.3%. On the other hand, it has also been shown in Experiment 3.3.1.2 that there are cases where the problem is too complex for an FFANN to learn (where the classes cannot be easily separated using the given dimensions of the data), but with the preclustering step the network can still learn it with a high accuracy (which, depending on the chosen distance factor, can be as high as 97.8%). This is because the resulting reduced dataset, while retains roughly the same amount of information from the original dataset, it can potentially remove inconsistencies that made the separation of classes hard or even impossible.

- For *Circular Fuzzy Neural Networks* the proposed method is used for compressing the datasets of Fuzzy Hand Posture Models (proposed in Chapter 2), in order to reduce the time required for the training phase. It has been found that using a distance measure of 0.4 can reduce the required training time by 32.8% while not causing any decrease in classification accuracy (i.e. the implicit information carried by the reduced dataset is more or less the same as that of the original dataset).

- For *Radial Basis Function Networks* **the proposed method is used to determine the number of neurons in the hidden layer**, i.e. the structural complexity of the classifier. The experiments have shown that the complexity of a given problem is inversely proportional with the value of δ that leads to a quasi-optimal structure. E.g. a problem with little or no inconsistency in the data ("overlapping" classes) can be learned using fewer neurons that results from relatively high δ, while more complex problems require smaller distance factor values and result in a higher number of clusters (and thus, neurons). The method has been shown to achieve a classification accuracy of ~94% on the testing dataset for simple problems and ~76.9-81.6% for more complex problems.

3.5.1 Thesis Statement II.

In order to enhance the training speed of Artificial Neural Networks, I have introduced a new supervised Neural Network training method including an input sample clustering step. The proposed method generates clusters of similar input data samples and returns the centers of the clusters. The training is done by using the corresponding cluster centers instead of the input samples. The method has been shown to decrease training time significantly, at the cost of a slight decrease in classification precision. The new method has been validated on Artificial Neural Networks, Circular Fuzzy Neural Networks and Radial Basis Function Networks.

Related publications: [P. 6], [P. 15], [P. 16], [P. 17]

Number of independent citations as of May 2021, according to Google Scholar: 5

Chapter 4

An Intelligent Space Control System Framework

Today, with the spreading of machine intelligence, *smart environments* have become an important research area. Their aim in general is to increase humans' life comfort, offer support to disabled persons thus helping them in their full and independent life (ambient assisted living), etc. Smart environments can range from smaller areas like intelligent classrooms, through medium sized areas like smart homes, to larger areas like modernized hospitals, parks, or even a whole city.

Many research projects have taken aim to design such a system for various application areas. To mention a few, Microsoft's *Easy Living Technologies* [62] aims to develop an architecture that aggregates a diverse array of devices through a middleware (i.e. a program that offers common services to applications outside of the ones offered by the operating system), to create a more comfortable and coherent user experience. The *Interactive Workspace Project* [63] at Stanford University proposed an augmented work sharing environment for smart offices and laboratories. Another example is [64] that proposed a cloud-assisted agent-based Smart home environment that provides a framework for activity recognition in smart home applications.

ISpace control systems, existing and under development (see e.g [2] and [43]), just like other smart environment projects, aim at tasks such as monitoring of physiological functions of human users, positioning and tracking of humans, localization and control of mobile robots, finding paths for robots by using itineraries taken by people, etc. A detailed overview of the iSpace can be found e.g. in [1], [43], and [2].

The main difference between iSpace-based and other control systems is that in the former, it is presumed that the various sensors and non-human agents that the system is in contact with have a bare minimum "competence" level (i.e. their overall capabilities do not

necessarily exceed what is required to provide the functionality that is expected of them). E.g. a window controller agent only needs to know how to open or close windows, it does not need to understand commands issued by the human user, for which it can rely on another system; and similarly, an item carrier robot only needs to lift and carry items from point A to point B, if it also can rely on another system that can "see" in its stead and coordinate its path. Implementing such a functionality is one of the main ideas behind the Intelligent Space.

There are several requirements that the hardware and software architecture of the iSpace has to satisfy. As stated in [43], these are:

- **Modularity**: building the system from a set of connected modules ensures the possibility of run-time reconfiguration when a component is added or removed from the system.
- **Scalability**: the iSpace must be able to adapt itself to environments of various sizes and shapes; furthermore, it should be possible to integrate local systems into larger ones.
- **Ease of integration** it should be easy to integrate existing intelligent components into the system.
- **Low cost of the components**: as the iSpace is built of many smaller components, the economic factor is determined by the cost of the components.
- **Ease of configuration and maintenance**: it is expected that it should be easy to set up and maintain an existing system. This implicitly carries the requirement for *adaptability*, meaning that the control system should also have the capability to learn by itself, in order to automatically adapt to a changing environment, thus, reducing the amount of configuration and maintenance the user has.

In this chapter, **a new iSpace Control System (iSCS) framework** is presented that I have developed in order to implement a smart environment that can provide services to users based on the available resources, and satisfies the requirements listed above: it is modular, scalable, easy to integrate, can be built from low cost components, is easy to configure and maintain and capable of tuning itself to adapt to changes.

The basic schematic of the iSCS can be seen in Fig. 4.1. The system is designed in a modular way: each of the four parts in it contain multiple modules, which are customizable with the available devices in mind.

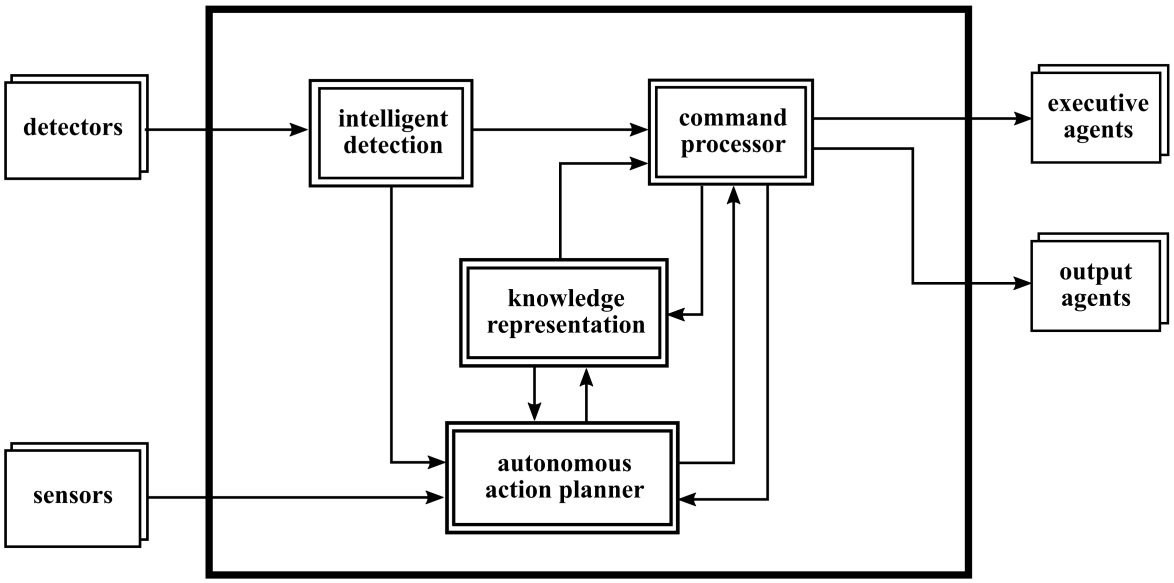


Figure 4.1: The simplified architecture of the iSpace Control System framework.

The modules of the *intelligent detection* part can process the data from various *detectors* (such as cameras and microphones), transforming it to the form of commands that the modules of the *command preprocessor* part can interpret using a suitable *knowledge representation*, and give instructions to *executive agents* (i.e. devices that can directly provide the user with a given service) and *output agents* (devices that communicate toward the users, such as screens and speakers). Furthermore, the system can learn to issue such commands though the modules of the *autonomous action planner*, using the data from sensors (providing e.g. temperature data) and the output of the intelligent detection module.

To implement a sufficiently advanced reasoning method (to understand commands) for the new iSCS framework, a suitable *knowledge representation* system is required. There are also various different knowledge representation approaches in the literature which can be applied in man-machine co-operation, as well. Among of the most prominent approaches are *conceptual graphs* (CG) (see [65]) that have primarily been developed for data base interfaces, to make easier for humans to understand the data and to make inquiries. This approach applies the concepts as basic primitives and the concepts themselves are types which incorporate every instance that shares that type. In [66] a CG-based knowledge model is proposed to represent *concepts*, *terminologies*, *methods*, and *processes* in Software Architecture. The knowledge is classified into a hierarchy of 4 levels: fundamental concepts, domain knowledge, process knowledge, and task knowledge. [67] proposes a *graph-based knowledge representation* for a Geographic Information System (GIS), to represent spatial and non-spatial data (nodes), also including spatial relationships (edges)

between spatial objects and use the model for generating a dataset composed of both types of data.

Another example is [68], where the authors propose *Feature Event Dependency Graphs* (FEDG). It focuses on representing the fact level knowledge in a compressed manner without losing any important information. FEDGs is efficient in retrieving user concerned knowledge patterns and is especially useful in discovering latent knowledge and in effective reasoning. In FEDGs, the information is represented by feature events (nodes) and the weighted context links and dependency links (i.e. directed edges) between them.

In order to describe knowledge that the iSCS framework holds about abstract concepts and real instances, **I have designed a new knowledge representation system** that uses a *graph-based structure*. The new approach belongs to the wider family of conceptual graphs mentioned above and can be considered as a more specialized version of CG. The proposed system also enables automatic observation-based learning for the control system it is implemented in. Hypotheses are used to store the observed a posteriori knowledge (gained by analyzing the details of commands issued by the user), which are used for the automatized operation.

This chapter is organized as follows: In Section 4.1 knowledge representation is described in details. Section 4.2 describes the general architecture of the proposed iSpace Control System framework. Section 4.3 presents a simulation to illustrate its operation, and Section 4.4 evaluates the proposed framework and gives a comparison to other systems. Finally, Section 4.5 summarizes the new results.

4.1 Knowledge Representation

In order to describe knowledge that the system holds about abstract concepts and real instances, I have designed a new knowledge representation system. In the proposed framework, *knowledge* is defined as *information about the relationship between abstract concepts, as well as between their real world instances*. This knowledge, however, is restricted to information that is *useful* for the functionality of the control system (e.g. information such as "*coffee is a drink*" and "*a window controller can open a window*" is directly useful for its operation, while facts like "*a brick can break a window*" is not. As a result, the size of the structure remains in manageable proportions, which also directly affects the speed with which the system can use it for inference (i.e. to find which device in the iSpace can execute a given instruction).

4.1.1 Knowledge Structure

To put the knowledge defined above into a form that an automated control system can process, a *graph-based structure* is used. The advantage of graph-based structures lies in their flexibility (they are easy to alter by adding, modifying or deleting *nodes* or *edges*) and readability (they are also easy to read and understand by humans). This knowledge is stored in a so-called *knowledge base*, which can be implemented as simple lists.

Each node represents an *abstract concept* or a *real-world instance*. Thus, the structure is divided into two parts: the *abstract* and *instance* domains. In the **abstract domain** the nodes denote the known *concepts* (e.g. the concept of windows) and the directed edges between them describe their relationship, thus describing the knowledge of the system about the world in general. In the **instance domain** the nodes represent existing objects (i.e. known instances of the abstract concepts).

The nodes themselves are homogeneous and nameless, their identification is done through *dictionaries*.

4.1.2 Dictionaries

In the new knowledge representation schema, *dictionaries* are used to map *labels* (one or more words (i.e. expressions)) to the appropriate *nodes*. Multiple labels can be assigned to the same node (synonyms) and multiple nodes can be assigned to the same label (ambiguity).

The most basic form of a dictionary is an ordered list of tuples containing the labels and the identifiers of their corresponding nodes (along with a unique ID number for each entry): `[ID|Label|NodeID]`. The search in such a structure can be done linearly with $O(N)$ complexity (where N is the number of entries) or $O(\log_2 N)$ if binary search is used (if sorted by the desired value, e.g. by NodeID), or even $O(1)$ if a hash or indexing table is used (although at the cost of a larger structure.).

The construction and maintenance of the dictionary and the associated knowledge base happens simultaneously: whenever a new node is created or deleted, a new entry is also added to or omitted from the dictionary. The two main domains can have the same, common dictionary or they can have separate ones. The advantage of the former is simplicity considering storage, while the latter is faster (e.g. no need to look through abstract nodes to find instance nodes).

Remark: Dictionaries of different languages can be assigned to the same knowledge base, in which case a primary dictionary (or dictionary pairs, if the two domain is kept separately) is needed to be distinguished and maintained constantly when changes are made to the knowledge base, with the rest being optional, at the discretion of the administrator.

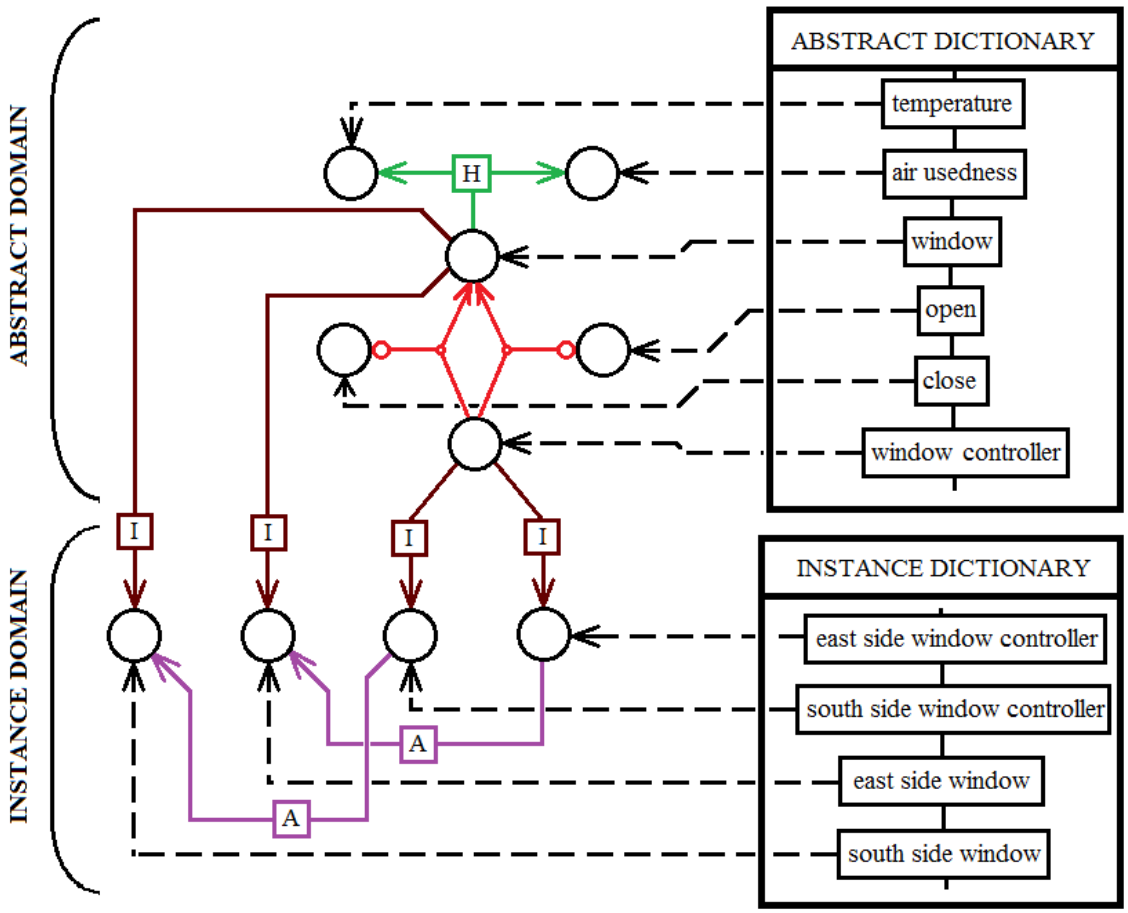


Figure 4.2: An illustrative example for the structure of the knowledge representation.

Fig. 4.2 depicts the distinction between the abstract and instance domains, as well as the dictionaries: the two domains have two separate dictionaries. In the figure an example is shown how human knowledge about the existing windows and their controllers in a room can be represented using the new schema. A window can be opened and closed by its controller. These actions depend heuristically on the temperature of the room and how used up is the air in the room (i.e. the measurable reasons why the user would want the window to be opened). For the sake of simplicity, the outside temperature and the current time is not shown in the figure (although they are also considerable reasons in this particular example). Let us consider that there are two windows and window controller devices in the room (denoted by the side of the room where they are located: east or south), and the controllers are associated with the appropriate windows they can do actions to, thus it is trivial which device can operate on which window. With this structure, an iSpace control framework can simply be made where the user can instruct the iSpace to open or close one or both windows.

4.1.3 Relationships between Concepts and Instances

In the new schema, different types of edges are used to depict different kinds of relationships between entities and actions. The edges can be sorted into 3 categories: *descriptive*, *quantifier* and *specification* edges. In the following, nodes are denoted with N_A, where A is a given concept that is represented by the node. Furthermore, a (i.e. lower case symbols) represents numerical values.

4.1.3.1 Descriptive edges

Descriptive edges describe the relationship between abstract concepts.

- **Inheritance edges** connect a general and a more specific concept. An inheritance edge connects node N_A to node N_B, if A belongs to type or category B. E.g., `coffee is a drink`. N_A inherits the properties of N_B: any other edges connected to B are regarded as if they were connected to N_A as well. They are denoted by an empty triangle-headed arrow towards the more general concept.

- **Ability edges** are three-way directed connections, where node N_A is connected to nodes N_B and N_C, if concept A *can do* B to concept C. B is typically the concept of an action (e.g. opening or closing something). For example, `drink machines can make drinks`, where A is node `drink machine`, B is node `make` and C is node `drink`. In visual representation, the action node B is marked with a small circle, and the direction is pointing toward C.

- **Heuristics edges** are directed edges between entities that provide heuristic information that can be considered for learning. Node N_A is connected to node N_B, if concept B can be heuristically bound to concept A. For example, actions done to `coffee` are dependent of `time` and `mood`, if the main determining factors for the given user are the current time and his/her mood when he/she instructs the system to make coffee. Heuristic edges are denoted by an arrow with label [H].

Fig. 4.3 depicts these relationships. For easier view, the labels assigned to each node are written into the nodes. The figure encodes the following the knowledge:

- `tea`, `coffee`, and `cappuccino` are `drinks`,
- `drink machines` can `make drinks`,
- and actions done to `drinks` are heuristically dependent on `time` and `mood`.

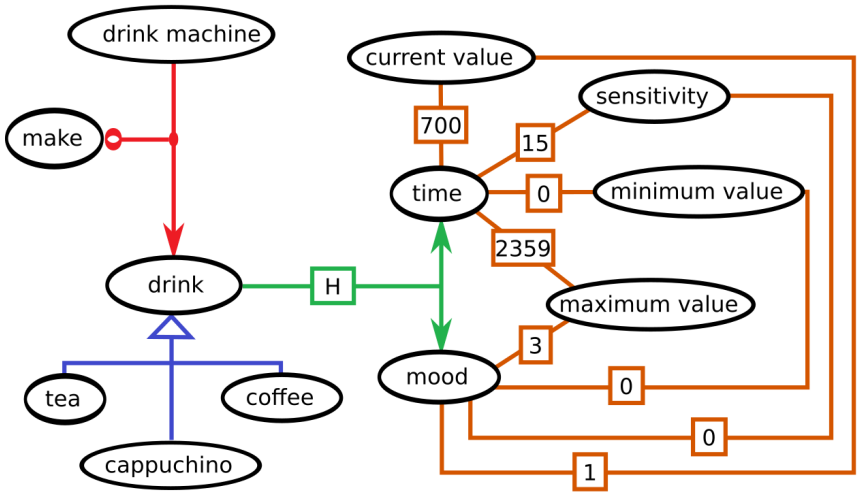


Figure 4.3: Example for the structure of the knowledge representation: inheritance, ability, heuristic and meta edges.

This makes it possible that if the control system received a task like "*Make coffee!*", then it can deduce from the structure that *coffee* is a *drink*, and *drink machines* can *make drinks*. Furthermore, if the system is such that it can learn from tasks, then it will know that the values of time and mood are needed to be considered regarding this particular task.

4.1.3.2 Quantifier edges

Edges can be used to assign numerical (crisp and fuzzy) values to concepts as well. Two types of edges are distinguished into this category:

- **Meta edges** assign crisp numerical values to concept pairs: node N_A is connected to node N_B, if concept A has a numerical value in B *quality*. E.g., if a meta edge between `current value` and `day` (of the week) is 6, that encodes the information that it is the 6$^{\text{th}}$ day of the week currently (which is Saturday, if the indexing of the week starts with Sunday with value 0). In the following, let us denote the value of the meta edge between nodes N_A and N_B with $V_{A,B}$.

  Meta edges make it possible to appoint *variables*, for storing the values of sensors or any other numerical values. A variable is defined by 4 attributes: *current value* (CV), *minimum value* (mV), *maximum value* (MV) and *sensitivity* (SEN). Therefore, a variable node is a node that is connected to nodes N_{CV}, N_{mV}, N_{MV} and N_{SEN} through meta edges. The attribute value u for variable U can be gained from $V_{U,CV}$, the range where u can take values from ($u \in [mv, MV]$) are $V_{U,mV}$ and $V_{U,MV}$, and

finally, $V_{U,SEN}$ is used to help handling *uncertainty*. Using the previous examples, e.g. $time$ and $mood$ are variables.

Fig. 4.3 depicts meta edges that connect `time` and `mood` (presuming that the control system that employs the knowledge representation can detect such thing) to the 4 aforementioned nodes (thus, defining them as variables). In the figure, it can be seen that the current time is 7:00 (700), the value of time can range from 0 to 23:59 (using the MMHH format to gain a single numerical value), the mood of the user can take up value between 0 and 3, with the current mood being 0 (more information about this in Section 4.3).

- **Fuzzy edges** assign *fuzzy membership functions* (FMF) to concept pairs. Node N_A is connected to node N_B with a fuzzy edge, if concept A has fuzzy membership function μ_B assigned to it. The input value x that is used to determine the FMF value $\mu(x)$ is gained from the meta edge between N_A and N_{CV} (i.e., $V_{A,CV}$):

$$M_{A,B} = \mu_B(V_{A,CV}) \tag{4.1}$$

where $M_{A,B}$ fuzzy edge holds the output value of the given FMF μ_B between nodes N_A and N_B, given $x = V_{A,CV}$. Fig. 4.4 depicts the FMF for B=`weekend`, where the input x comes from the current value of A = `day` ($x \in [0, 6]$), while the minimum and maximum values that x can take in the membership function are given by $V_{day,mV}$ and $V_{day,MV}$, respectively. $V_{day,SEN}$ determines the width of the boundary areas of the FMF. Fuzzy edges are generally useful for the learning functionality of the system.

4.1.3.3 Specification edges

The edges in the specification category are connections between either an abstract node and an instance node, or between two instance nodes.

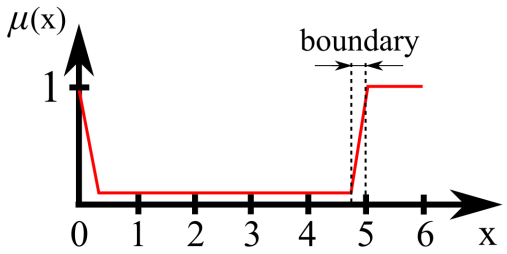


Figure 4.4: Fuzzy membership function for *weekend* (of the week). Input variable x is given by $M_{day,CV}$, while $M_{day,SEN}$ determines the width of the boundary areas of the FMF.

- **Instance edges** specify a real-world entity (be it an object or living being) to be an instance of a given abstract concept. They are denoted by arrows with label [I], and store the IDs of a given device (e.g. `window controller`) or other entity (e.g. coffee). In practice, the ID can be the network address of the given device.

- **Association edges** mark a connection between two instance domain nodes (that are the instances of abstract nodes connected to each other by an ability edge). Their main role is to provide additional information in cases where there are multiple instances of the same kind of abstract concepts, e.g. multiple windows in a room, with a window controller connected to each (this case can be seen in Fig. 4.2). Association edges are denoted by an arrow with label [A].

4.1.4 Observation-based Learning

Many control systems benefit from learning, be it about the habits of the user or a changing aspect of its environment. Observation-based learning is one of the most typical (and natural) approaches where the task is to analyze the features of given events, and create models that help recognizing those events afterwards.

In the proposed knowledge representation system, these models are called *hypotheses*. A **hypothesis** (in this framework) can be regarded as an assumption that each event occurs whenever a certain set of conditions (that are specific to each event) are met. For example, in case of an intelligent office, the event is the user giving orders to the system. He/she orders coffee to be made around 8:00 every *workday* (i.e. when $M_{day,weekend} < \tau$ (where $\tau \in [0, 1]$ is an arbitrary threshold) which means it is not currently weekend). If the control system takes note of the command and the values of the variables when the command was received (based on the given heuristics), and finds that the command was issued multiple times with very similar variable values (where $V_{Z,SEN}$ defines what "similar" means for a given variable Z: 0 allows no deviation, while $V_{Z,SEN} \geq 0$ values allow some level of diversion), then the next time the same conditions occur, the control system can confidently issue the same command. Thus, the user can walk in for a fresh cup of coffee waiting for him/her.

4.2 The General Architecture of the Proposed iSpace Control System Framework

Based on the presented new knowledge representation, I have developed a new iSpace Control System (iSCS) Framework.

Fig. 4.5 shows the general architecture of the framework, where the inputs for the system are provided by different kinds of *detectors* and *sensors*. The former are devices that are capable of detecting commands given by human users (e.g. microphones for audible or cameras for visual commands, like hand signs), while the latter ones are simple devices which can measure simple properties of the environment (like its temperature). *Executive agents* are electric devices that are able to satisfy the needs of the user (e.g. window controllers). They are assumed to have just enough built-in intelligence to be able to do their predefined task. The iSpace is connected to them, as well. When executive agents are registered by the administrator, setting their type as well in the knowledge base (e.g. a `window controller`). Finally, *output agents* are simple devices that the iSCS can use to communicate towards the user, e.g. screens and speakers.

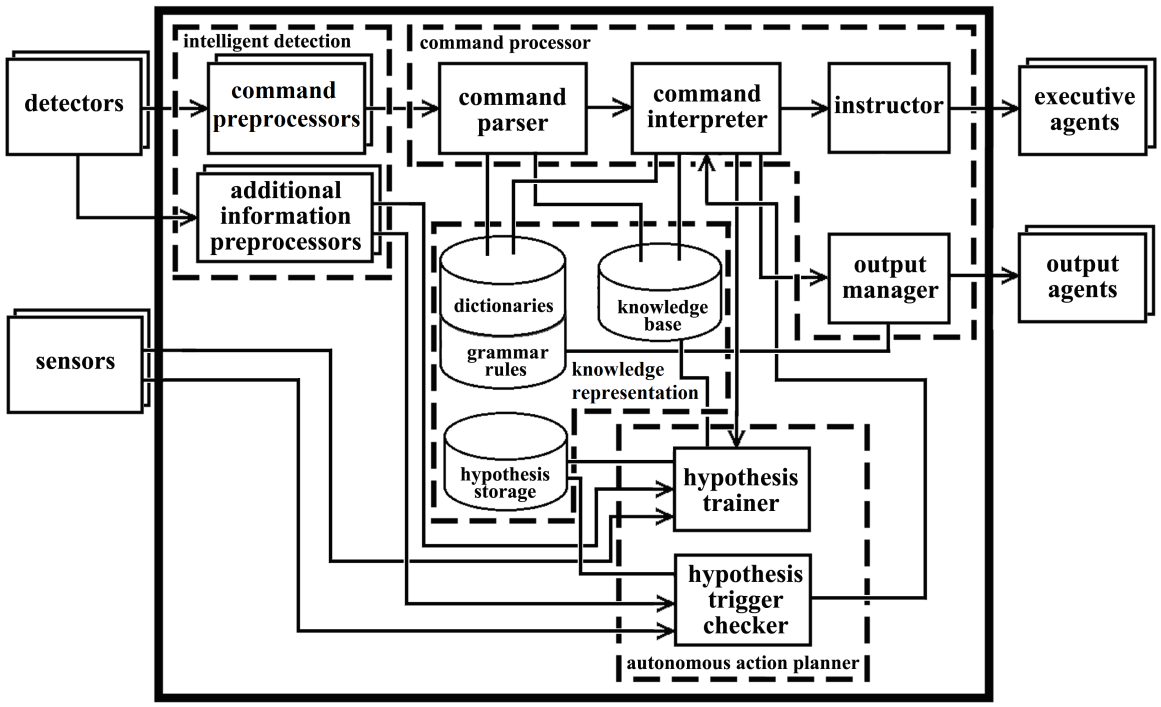


Figure 4.5: The detailed architecture of the iSpace control system framework.

The iSCS framework consists of 4 main parts (as seen in Fig. 4.5):

- **Intelligent Detection**, that detects commands given by human users and turns them into strings of words that can be processed by other modules.
- **Command Processor**, that processes the preprocessed commands and instructs the appropriate executing agents to carry them out.
- **Knowledge Representation**, that defines the way knowledge is described and stored in the system.
- **Autonomous Action Planner**, that is responsible for observation-based learning and the trigger-based autonomous actions of the system.

4.2.1 Intelligent Detection

In the Intelligent Detection part, *Command Preprocessor* modules (CPPM) extract usable commands from the data provided by the detectors. This can be done using different methods of man-machine interaction, be it through verbal communication, hand gesture-based sign languages, text messages, etc.

Furthermore, additional information can be used through the *Additional Information Preprocessor* modules (AIPM) to gain valuable data. Such information can be e.g., the mood of the user ([69] proposed such a method).

These modules are designed to be easily extendable or replaceable by other modules to make the operation of the iSCS more flexible and robust.

4.2.2 Command Interpretation and Command Processing

In order to achieve an advanced level of command processing, the system has to use grammar rules. These contain rules that determine how the words and sentences can be built with regards to the language. The modules of the *Command Processor* part analyze the given command (with regards to its language and grammar) and instruct the appropriate executive agents to carry it out.

4.2.2.1 Command Types

Two types of commands are be defined in the iSCS Framework: *instructions* and *prohibitions* (Fig. 4.6).

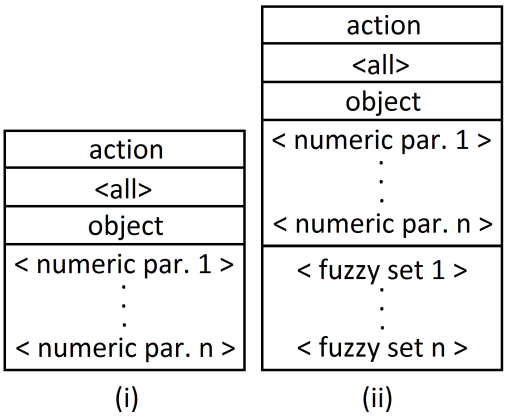


Figure 4.6: Structure of the (i) instruction and (ii) prohibition type commands.

Instructions are simple commands. They are given by the user to *induce change in the environment*. The first parameter of an instruction describes the action that is needed to be executed, possibly followed by an optional "`all`" word, which means that the action is needed to be executed to all available objects (e.g., "`open all windows`"). The second parameter describes the object of the action which can be followed by optional numerical values.

Prohibitions are commands that are given by the user to alter the behavior of the system by binding one or more commands that were already learned to additional conditions, thus, *inducing change among the hypotheses*. Their structure is basically the same as that of the instructions, except that they start with "`DO NOT`" and have (at least one) additional text parameters, which usually represent fuzzy variables. E.g., in case of the prohibition command "`DO NOT make coffee on weekends!`". `Weekend` is defined by the fuzzy membership function shown in Fig. 4.4. Remark: if the sensitivity of *day* (i.e. $M_{day,SEN}$) is more than 0, then small parts of the surrounding days (e.g. very early Monday and late Friday night) count as "`almost Weekend`", adding flexibility to the system (due to fuzzy logic).

4.2.2.2 Command Parsing

The *Command Parsing* module(CPM) first determines the type of the command, and then parses it. The latter step depends on the type of the command (due to the differences in the command structures described in the previous subsection). The input of the CPM is the preprocessed command and its output is the parsed command.

It is presumed that the command is in a predefined format (which, in the simplest case is based on the strictly defined word order of the English language), so the algorithm of the

parsing phase is quite trivial for simple cases. To mention an example, in case of instructions the first word is always the action and the second one is either the "`all`" word or (after the removal of the occurring articles, like "`the`") a noun that gives the object of the command.

On the other hand, more advanced parsing can be used as well, for which Fig. 4.7 shows an illustration. The system first parses the sentence to separate words, then analyzes each part. From that, it produces a graph, where the bigger circles are references to the appropriate concepts of the knowledge base, while the smaller ones connected to them are features of those words (e.g. an article belonging to the noun, is it plural or singular, etc.). Lastly, the squares denote the words-part of speech. This way the commands that the human users can give become more flexible, the users do not need to stick to a rigid order of words if it is unnatural in their native language.

4.2.2.3 Command Interpretation

The purpose of the *Command Interpretation* module (CIM) is to determine which executive agent is able to carry out a given command. Its input is the preparsed command and its output is the network address or reference of an executive agent. The CIM only processes instruction commands since prohibition type commands are not needed to be executed (they are processed by the Autonomous Action Planner part instead). The algorithm searches for three key nodes, in order: the *object node* N_{obj}, the *action node* N_{act} and the *executive node* N_{exec}. These three nodes are needed to be connected via an ability edge.

The *object node* is a node that either corresponds to the object of the command (through the dictionary) or can be reached from the corresponding node through inheritance and/or synonym edges with a constraint that the node is needed to have at least one instance (a node in the instance domain, which is bound to the *object node* with an instance edge).

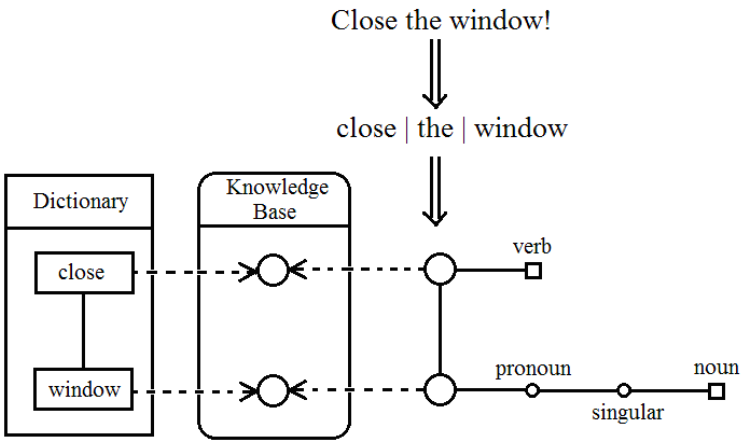


Figure 4.7: Example for the command parsing.

The *action node* is a node that either is corresponding to the action of the command, or can be reached from the corresponding node through inheritance, similarly to that of the object node, with the difference that the action node does not need to have any instance. The *executive node* is a node that can be found the same way as the object nodes, with the difference that its instance is needed to be associated with the instance of the *object node*.

Thus, the algorithm effectively does the following: First, it finds out what concept the object of the command is (by looking up the ID of the node assigned to the given label, then searching through the list of the nodes, combined with the list of the inheritance edges), then what action is needed to be done to the object (similarly); and finally, what concept can do that action to the object. If the algorithm cannot locate the three previously defined nodes, then it stops: the instruction cannot be carried out. If it finds three nodes that satisfy all of the constrains defined above, it returns the address or reference of the executive agent that is stored in the instance node of the *executive node*. Therefore, an instruction can be executed if the resulting N_Y node is not an empty (NULL) value:

$$N_Y = \begin{cases} N_{exec_i}, \;\; if \exists E^{ability}_{N_{exec},N_{act},N_{obj}} \; and \; \exists E^{instance}_{N_{obj},N_{obj_i}} \; and \; \exists E^{instance}_{N_{exec},N_{exec_i}} \; and \; \exists E^{association}_{N_{obj_i},N_{exec_i}} \\ NULL, \;\; otherwise \end{cases} \tag{4.2}$$

where N_{obj_i} and N_{exec_i} are the instance nodes of N_{obj} and N_{exec}, respectively, and $E^T_{N_1,N_2}$ is an edge with type T between two given nodes N_1 and N_2, and $\exists$ is the symbol for existence in mathematical logic.

Remark: Not all *objects* have instances that constantly exist, e.g. *coffee* does not exist in the desired form of a "cup of coffee" until the machine makes it. In the proposed system this problem is solved by creating so-called *void nodes* that serve as an instance to such concepts (and are associated with the instance of the coffee machine).

Let us consider the example from Subsection 4.1.3. The user issues the command: `Make coffee!` The algorithm first identifies node `drink` as N_{obj}, since it is the ancestor of the node N_{COFFEE} (see Fig. 4.3). Then it identifies node N_{MAKE} as N_{act}. Finally, it recognizes the node $N_{DRINK\ MACHINE}$ as N_{exec}. If it has an instance (even though the instances are not shown in that figure) that is associated to the instance of N_{DRINK}, then the system returns the address or reference of the drink machine agent stored in the found instance node.

4.2.2.4 Instruction and Output Manager

The task of the *Instructor* module (IM) is to instruct the executive agents (that the iSCS is connected to) indicated by the CIM (with N_Y) to execute the task in order to provide the desired service for the user.

In advanced man-machine co-operation the communication is usually bi-directional, i.e. the iSCS has to be able to construct questions or give information in a way that the user can easily understand. This is the main function of the Output Manager module (OMM). In case of verbal communication, this can be solved by the usage of voice synthesizers (see e.g. [P. 1]) or by simply writing the message out to a screen.

4.2.3 Knowledge Representation in the iSCS

The knowledge representation in general has already been described in the previous section. In the iSCS Framework the structure is stored in a so-called *knowledge base*, which consists of the lists of the nodes and the edges. Whenever the "world" of the iSpace is altered in any way, the changes should appear in the appropriate domains of the knowledge base as well. The effect of "physical" changes (addition or removal of sensors, detectors, agents, etc.) should appear in the instance domain, while "property" changes (temperature of the room, etc.) should appear in the abstract domain. The content of the knowledge base is maintained by the system administrator, or a human user with high enough permission levels. The modifications, however, are easy to make as the they only require the addition, modification or deletion or given nodes or edges. In practice, this can be simply done through a suitable graphical user interface. Simple grammar rules (defining e.g. the order of the words in a sentence, etc.) are used during command parsing and while constructing the messages that the system needs to send to the user via the output agents.

4.2.3.1 Hypothesis Storage

Due to the support for observation-based learning in the knowledge representation, the iSCS Framework is able to build hypotheses about the habits of its users, which means that the *commands* are treated as *events*. The hypotheses are stored in the *hypothesis storage* (using a simple list). A **hypothesis** consists of a compact form of the command (i.e. the IDs of the involved nodes), optional numerical values (for commands where the user wants to set a given value, e.g. the alarm clock), gained from the command the hypothesis is based on.

```
Hypothesis: turn off    light    -1:-1
   -Trigger: justification=3
       Conditions:
       - Node: time | values: 2200 | 15  permitted: true
```

Figure 4.8: A hypothesis made from the command "`Turn off the lights`".

Every hypothesis has at least one *trigger*. A trigger is a set of conditions that are valid at the moment of the detection of the command (i.e. their *current values* when the event was detected). Thus, each condition is set up from the *current values* and the *sensitivity* values of the variables that are heuristically connected to the given object of the command. Every trigger has a justification value, which keeps account of the times the command has been issued *by the user* under the same conditions. Lastly, each condition can be either permitted, or forbidden. This means that permitted conditions will be true if the current values during the checking are aligned with the values of the condition, while forbidden conditions are the opposite: they will be true if their values *do not* align with the current ones.

For example, Fig. 4.8 shows the structure of the hypothesis that was made using the command `"Turn off the lights"`. The command is written in the first row (since there are no additional numerical values, they are -1 by default). It is followed by the list of triggers, each having its own justification value and set of conditions. Let us assume that `light` is only connected to `time` with a [H] edge, thus it is used as condition node with the current value and sensitivity (which shows that the user issued the command at 22:00, and the time can deviate by 15 minutes (the latter is set by the system administrator)). Lastly, the condition is permitted, thus it will only be true when the current time is around 22:00.

4.2.4 Autonomous Action Planner

The part of the intelligence of the iSCS, so-called *Autonomous Action Planner* (AAP) is responsible for learning via hypotheses and for decision making: whether or not to take actions according to what the system has learned.

4.2.4.1 Hypothesis Training

The task of the *Hypothesis Trainer* module (HTM) is maintaining the hypotheses stored in the system according to the received commands. The algorithm (Fig. 4.9) of the hypothesis training works as follows: If the command is an *instruction*, then the system searches for a *hypothesis* that has the same action, object, and more or less the numerical parameters (within the range defined by the sensitivity parameter of the given variables). If no such hypothesis is found, then a new one is created using the parameters of the command, a new *trigger* and new *conditions* are added with the variable nodes that are connected to the subject node with heuristic edges. There will be as many conditions as many heuristic edges are connected to the subject node. The current value and sensitivity of each condition is derived from the current value and sensitivity of the variable (see example in Subsection 4.2.3.1). If there already is such a hypothesis, then the algorithm checks its triggers.

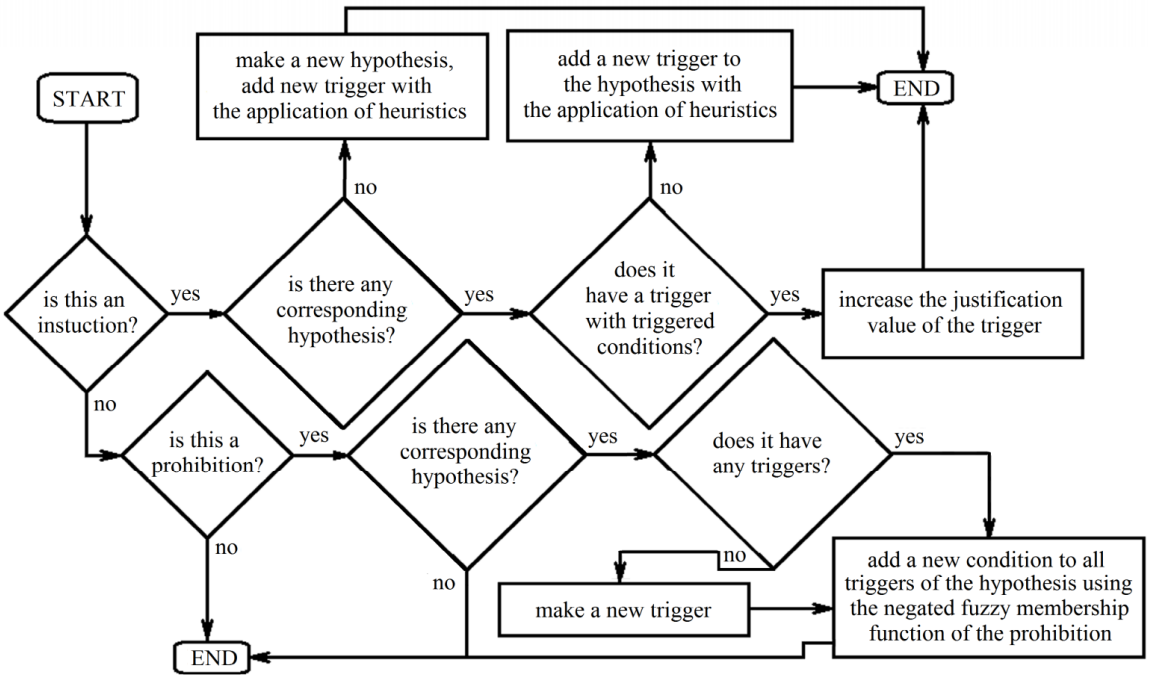


Figure 4.9: The algorithm for the Hypothesis Trainer Module.

If there is a trigger with conditions triggered by the current values of the variables, the algorithm increases the justification value of that trigger and the algorithm ends. If there are no triggers with fulfilled conditions, then a new trigger is added using heuristics just like it is explained above.

If the command is a *prohibition*, then the algorithm searches for a corresponding hypothesis. If the found hypothesis does not have any triggers, it adds a new trigger to it (along with the conditions, as explained above). If it has at least one, then the system adds a new condition to all its triggers using the fuzzy membership function of the prohibition and setting the new conditions to *permitted=false*.

4.2.4.2 Hypothesis Trigger Checking

Using the information from the observations, the system can anticipate the user issuing the same command the next time the conditions of the maintained hypotheses align with the actual *variable* values. The task of the *Hypothesis Trigger Checking* module (HTCM) is to frequently check the conditions of the hypotheses in the hypothesis storage and compare them to the current values of the variables in their trigger list.

A hypothesis is triggered if the conditions of at least one of its triggers is fulfilled:

- the current values ($V_{Z,CV}$ of each considered variable Z) is within the deviation defined by the sensitivity parameter of the variable ($V_{Z,SEN}$),

- all considered fuzzy membership functions return a value over an arbitrary threshold τ
- and the justification value of the hypothesis is higher than an arbitrary value Θ.

If a hypothesis is triggered, then the HTCM sends the command of the hypothesis to the CIM, which instructs the appropriate executive agents to carry it out.

4.3 Simulation

Figure 4.10: The virtual room in which the iSpace application framework has been implemented and tested.

In order to demonstrate the functionality of the system, it has been implemented in a virtual room, where a simulated man is living his everyday life. He gives commands to the iSCS integrated to the room to have his needs satisfied. The system is equipped with detectors that can detect the given command.

Fig. 4.10 shows the simulation from the view of one of the four cameras (set up in the four corners of the room). As it can be seen, the room has a coffee making machine, an alarm clock, as well as an automated window and curtains. The user is sitting at his home office workstation. The implementation itself has been done using Visual Studio 2015, C++ and OpenGL on PC-3. The program indicates all the important environmental variable values, and the virtual user communicates directly with the iSpace.

The system also has an emotion estimator as an additional intelligent detector to gauge the mood of the user. The mood in this case is set to one of 4 simple categories (0:sad, 1:happy, 2:frustrated, 3:neutral). The room is also equipped with executive agents: a coffee machine, an alarm clock as well as multiple windows and curtains, along with their respective controllers.

The man living in the virtual room has his own schedule for work, leisure and resting. He gives certain commands to the iSCS in certain times (for example, "`open the curtains`" after waking up at 7:00 (in a *bad* mood), "`make coffee`" at 7:33 and 12:00 (which turns his mood into *neutral*), "`turn on the heating`" if the temperature is lower than 18°C (64.4°F), etc.). The system makes and manages hypotheses based on these commands. The justification threshold θ is set to 2, thus the system is only able to give out the command of the triggered hypotheses if the command has been detected at least 2 times under the same circumstances. The value of τ is set to 0.5.

Fig. 4.11 shows two of the hypotheses that the system has learned in 4 (in-simulation) days, one of which is based on the instruction "`make coffee`" (Fig. 4.11(left)). The command has been issued around 7:17, 7:33 and 12:00. The mood of the user was frustrated (2) and neutral (3). One other hypothesis (Fig. 4.11 (top right)) has been created from the command "`Set the alarm to 7:00!`" (top right). Each time the user issued it, a new trigger was made or an existing one got its justification value incremented (starting from 0).

The system has started issuing automated commands for hypotheses that have triggers $\Theta \geq 2$, so it instructs the coffee machine to make coffee at the right time and detected user mood combinations.

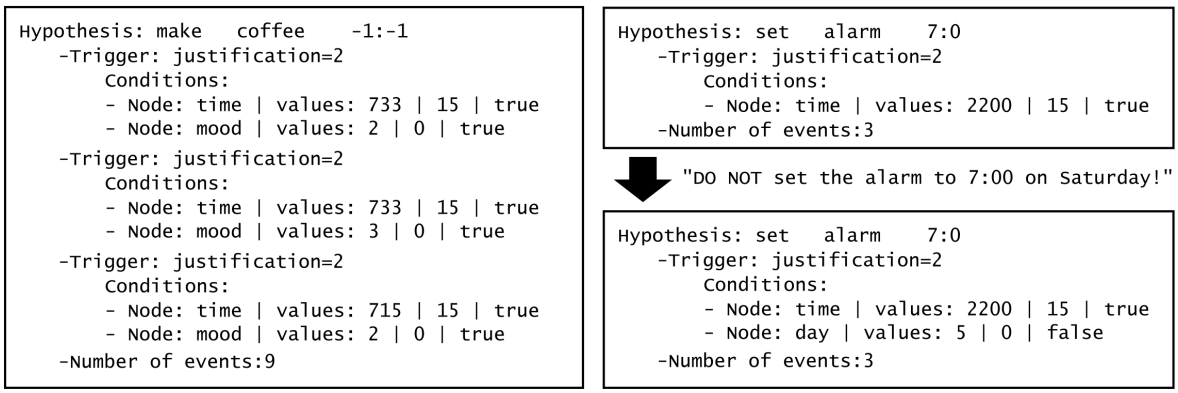


Figure 4.11: Hypotheses learned from the commands `Make coffee` (left), `Set the alarm to 7:00` (top right) and `DO NOT set the alarm to 7:00 on Saturdays` (bottom right). The latter was made from the previous one (with the same command), by adding a new condition to stop it from triggering on the 5th day of the week (i.e. Saturday).

However, as it can be seen in the figure, there are two conditions that trigger within a short time frame, which means that the user would get two cups of coffee in the morning, despite he usually only asks for one. The main reason for this is that the user did not ask for coffee in the same, exact time every morning, but slightly different times, and two of these times were further from each other than 15 minutes. This problem can be easily solved by either raising the value of $V_{time,SEN}$, or adding a so-called cooldown timer to the system that prevents it from issuing the same command multiple times in a set time frame.

Furthermore, the weekday and weekend schedules of the virtual man are different, though the system does not know about this at the first time, so it sets the alarm to 7:00 on Friday night. The first thing the user does after his unscheduled awakening on Saturday is giving the prohibition command "`DO NOT set the alarm to 7:00 on Saturdays!`". The system promptly adds this new condition to the existing hypothesis (Fig. 4.11 (bottom right)), and since Friday late night counts into weekends given that a FMF is used to describe the concept of *weekend*, the system will not make this mistake again.
Remark: issuing prohibitions does not have to happen *after* the hypothesis have triggered under undesirable circumstances, the user could have added the restricting condition to it beforehand as well.

4.4 Evaluation

In the simulation, has been shown that the proposed new Intelligent Space application framework is able to comprehend, interpret and execute detected and preprocessed commands given by a human user, if the commands are given in the anticipated form (in this case, simple English sentences). The system has been able to learn reoccurring commands that reoccur in the similar conditions, with the usage of heuristics and execute them when the conditions are fulfilled without requiring the user to give these commands again. During the simulations with the current implementation the system, it has been able to learn all the instructions and prohibition commands given by the virtual inhabitant of the virtual room.

Although the proposed iSCS framework belongs to the Intelligent Space applications, there can be found differences between it and a conventional intelligent room system such as presented in [70]. Comparing the architectures, in [70] a structure with 3 layers is proposed, where the perceptual systems (which determine what is happening in the room) are in the lowest layer.

In the second layer, a uniform agent based interface can be found associated to every device installed in the room and on the third layer there are application agents which

provide, with fairly limited levels of knowledge, functionality for specific types of uses in the Intelligent Space. On the contrary, the proposed system has a modular architecture and is managed apart from the sensors and executive agents. The advantage of using this flexibly designable knowledge base-based control procedure is that sensors, detectors, and executive agents can be added and removed anytime without reconfiguring the system; only a new entry or modification in the knowledge base is needed. Another important difference is that the proposed iSCS is designed to understand sentences instead of just keywords, which can be more comfortable to use for the human users. The system itself is still under development; in the current implementation it can only process the simplest English sentences. Improving this is one of the numerous ways to improve the capability and efficiency of the system.

A further feature that sets the presented system aside from other such systems is the application of *fuzzy membership functions*, which provides a simple way to represent and actively use more complex concepts, like "weekend", as well as help handling uncertainty.

The proposed *knowledge representation* system principally belongs to the wider family of conceptual graphs mentioned above and can be considered as a more specialized version of CG, however, with significant differences:

- The proposed system has been specifically developed for control systems.
- While in conceptual graphs the concepts (the nodes) themselves are types, the proposed system separates abstract concept nodes and their real-world instance nodes,
- In CG the nodes are implicitly labeled, while in the proposed system the labels are loosely attached to the nodes (using dictionaries to assign labels to node IDs), the proposed method does not define constrains like CG does, etc.
- In the proposed system, specific edges carry most information, as they encode the relationship between concepts and instances, while CGs define relationship nodes for that end.

4.5 Summary of New Results

I have developed a new smart environment application called iSpace Control System (iSCS) framework. The system acquires data from the environment it is situated in, and commands from human users that is parsed and interpreted in order to provide the user with various services, which are carried out by agents in the iSpace. Thus, said agents do not need

to possess such capabilities, they only need to be able to carry out their specific function. The system consists of **intelligent detection modules** that process data from sensors, and **command processor modules** that interpret the received command, determine which known agent can carry it out, then notify the agent to do so. By analyzing the details of the received commands, the **autonomous action planner modules** of the system can learn what to do (i.e. which commands to issue) automatically when certain conditions are met (derived from the observations).

The iSCS satisfies the main iSpace requirements: it is designed to have a *modular structure*, and as a result, any part of the system can be extended with or exchanged to new ones, so changes and improvements made to the system can be carried out easily. The system is *scalable*, meaning it can be used in environments with any size and volume, by extending with a sufficient amount of additional modules, sensors and agents. It is *easy to integrate* into any environment with *low cost components* (e.g. placing cheap, widely available single board computers with cheap sensors, like cameras and microphones). Finally, the system is *easy to configure and maintain*, due to the flexibility of the knowledge representation and the observation-based hypothesis learning.

I have designed a new knowledge representation system that uses a *graph-based structure* to describe information about concepts and instances, as well as about the relationship between them. The novelty of the presented structure lies in its specific edge types and that it stores and handles theoretical (abstract) and practical (real-word) knowledge separately. It is also able to build upon uncertain and ambiguous knowledge by using *fuzzy membership functions* and *variables*. As a result, the proposed knowledge model makes it easy to store, retrieve, modify, and extend theoretical and practical knowledge, making it ideal for the iSCS to interpret commands and to associate them with physical means and actions, to adapt to changes and learn new (both certain and uncertain) knowledge which aspects are especially important when the system is involved in e.g. man-machine communication. Another valuable property of the presented knowledge representation system is that **the core knowledge representation itself is independent of natural languages**, given that the information is carried by the edges that define the relationships between various nodes, so reasoning can be done with them even if the language component (i.e. the dictionaries) was taken away. This makes it simple to add more languages to the system.

I have implemented and tested the proposed ISCS framework and the knowledge representation system in a virtual 3D environment, using Visual Studio 2005, C++ and OpenGL. In the simulation, it has been shown that the new iSCS framework (using the new knowledge representation system) is able to *comprehend*, *interpret* and *execute* detected

commands given by a user. Furthermore, the iSCS framework has been shown to be able to *learn commands* that are issued multiple times in similar conditions with the usage of *heuristics*, and *execute them automatically* when the conditions are fulfilled without requiring the user to give these commands again.

4.5.1 Thesis Statement III.

I have developed a new iSpace Control System (iSCS) framework consisting of intelligent detection modules, command processor modules, and autonomous action planner modules, together constituting a smart environment system that can provide services to users based on the available resources and agents. I have also designed and implemented a new graph-based knowledge representation system to describe a priori knowledge for control systems.

The knowledge representation uses a graph-based structure that provides a flexible and easy to use architecture. The new knowledge representation system also enables automatic observation-based learning, in which hypotheses are used to model the observed a posteriori knowledge (gained by analyzing the details of commands issued by users), which are used for the automatized operation.

The iSCS satisfies the following requirements: modularity, scalability, ease of integration, can be built from low cost components and easy to configure and maintain, mainly due to the wide availability of cheap Single Board Computers and sensors that can be used to implement the iSCS framework. The functionality of the system has been validated in a simulated environment.

Related publications: [P. 2], [P. 7], [P. 18], [P 19], [P. 20, [P. 21]

Number of independent citations as of May 2021, according to Google Scholar: 1

Chapter 5

Adaptive Fuzzy Rule-based Classification with Fuzzy Filter Networks

Color filtering is an important field of image processing. It is typically used for enhancing the quality of the image output, but it can also be used for identifying or even locating objects with known color tones in camera images. Such a function is very beneficial for smart environments such as the Intelligent Space [2]: being able to provide location information for robots in the surveilled area, detecting events that require the call for help (e.g. an elderly person falls and cannot get up), etc.

There have been numerous machine learning methods that have been used for color filtering in the last 3 decades. Among the many different approaches, the most typically applied ones are *Artificial Neural Networks* (ANNs, [47]) have been advantageously used for many problems such as fruit ripeness recognition [71], traffic sign recognition [72], ocean color measurements [73], dermatological disease detection [74], etc. Although they are flexible classifiers with a very strong generalization ability (recognizing unseen data), their main disadvantages involve slow training and low interpretability.

k-Nearest Neighbor (kNNs, [75]) classifiers have also been used for color classification regarding beef quality identification [76], grape leaf disease identification[77], tomato ripeness classification[78], hair color detection in face recognition systems[79], etc. Their advantages are simplicity (since the training is practically non-existent, in turn they calculate a proximity function for each stored training data point during the evaluation phase), and generally high accuracy. On the other hand, their computational complexity is high, (as they need to do all calculation during the evaluation).

Another often used approach for color-based classification is Support Vector Machines [80]. They have been successfully applied in face detection [81], vision-based pest detection [82], vehicle color identification [83], tomato ripeness [84], etc. Although SVMs provide

robust classification, they do not perform very well when the data set has more noise i.e. target classes are overlapping.

Aside from the mentioned ones, there are two common disadvantage among the listed classifiers: low levels of interpretability (i.e. the data administrator cannot simply "look into" the knowledge model the classifiers have built to gain additional insight to the problem), and that their acquired knowledge is hard to purposefully configure after training (i.e. making the classifier 'forget' certain information, or learn new information without changing the already obtained knowledge).

Rule-based systems are yet another approach, where the classification of the input data is based on predefined `IF-THEN` rules. Such rules can be either crisp or fuzzy, and are usually described by sets of feature-value assignments, which makes it easy for a human data administrator to understand the implicit knowledge of the system (by looking at the individual rules), while also being able to modify it by adding, altering or removing rules. Rule-based classifiers have been effectively applied in many problems, such as online sentiment analysis [85][86], self-driving cars [87], malaria diagnosis [88], data stream classification [89], human activity recognition [90] etc. Fuzzy rule-based systems have been applied in color filtering problems as well, e.g. [91] and [92].

In this chapter, a new fuzzy rule-based classifier is introduced that can be used for color filtering. The so-called **Fuzzy Filter Network (FFN)** classifier is derived from the classic Radial Basis Function (RBF) networks: using the same structure, but instead of gaining the output value from the weighed and activated outputs of the neurons, *the FFN classifier uses each neuron to calculate the distance between the inputs and a given rule*. The output of the system is thus the class associated with the rule that is the closest to the inputs, i.e. the one that is the most similar to known models.

The training of the classifier entails the determination of the parameters of the rules that belong to each neuron in the hidden layer, which can be done through clustering. To achieve this, I have developed a **new clustering method**, which aims to make a covering of the problem space with hyperspherical multidimensional clusters. The two main parameters of the clusters are the center and range parameters, which are directly used in the neurons. In order to gain these parameter values, the new so-called *Analytical Radial Representative* (ARR) clustering method designates all input data points as an individual cluster center, sets their radiuses to half the distance to the closest cluster center (of a different class), then increases them to cover as much of the problem space as possible, without causing significant overlapping between the areas covered by the individual clusters.

Finally, in order to enhance the speed of the classifier, I have designed **a parallel version of the FFN** classifier, alongside with the parallel version of the ARR clustering method. The

performance of both sequential and parallel FFNs are illustrated through experiments.

This chapter is organized as follows: In Section 5.1 the proposed filter network is described, as well as the clustering algorithms used for training the network. In Section 5.2 experimental results are described to illustrate the efficiency of the classifier, while in Section 5.3 an evaluation is given and the applicability is analyzed. Finally, Section 5.4 summarizes the new results.

5.1 The Fuzzy Filter Network

5.1.1 The Architecture of the Classifier

The idea behind the new filter network [P. 8] is treating classification problems as (fuzzy [29]) inference problems. The reason why Radial Basis Function networks are ideal for this is that they are already realizing some sort of fuzzy inference in their neurons: their Gaussian activation function (like a fuzzy membership function) computes the similarity distance between the given (center and width) parameters of the neurons to the input data. The center and width parameters of each neuron can be derived from the parameters of the training data samples. Additionally, the categories of the samples (as they are given in supervised learning) are assigned to the corresponding neurons, either by an extra variable or simply by using the weight parameter of the neuron. This way, each neuron stores a filtering rule by its [center, width, and category] parameters.

Remark: For *distance* measure in this thesis work, Euclidean distance is used:

$$d(A,B) = \|A - B\| = \sqrt{\sum_{i=0}^{N-1}(A_i - B_i)^2} \tag{5.1}$$

where A_i and B_i are elements i of 1D vectors A and B, respectively; and N is the length (dimension) of both A and B.

Overall, only the output layer of the RBF is needed to be altered significantly, where the RBF normally calculates the weighted sum of the outputs of the neurons: instead, in the proposed architecture the highest value of the outputs of the neurons is determined. Thus, while in the original system, its output is simply the weighted sum of the output of the neurons in the hidden layer; in the proposed FFN the output is the *category* or *class* of the neuron with the highest output, or in other words, the class of rule that is the most similar to the input data:

$$y = \begin{cases} w_{\underset{\forall j}{\operatorname{argmax}}(g(x,C_j,\sigma_j))} & \text{if } g(x,C_j,\sigma_j) \geq \theta \\ 0 & \text{otherwise} \end{cases} \tag{5.2}$$

where 1D vector w stores the class labels assigned to each neuron, θ is an arbitrary threshold value; C_j and σ_j are the center and width parameters of neuron j, respectively. The argmax() function returns the j index belonging to the largest calculated Gaussian function value. The threshold parameter $\theta \in [0, 1]$ is used to set the sensitivity of the classifier: lower θ values allow for more vague similarity, while higher values make the classification stricter.

Fig. 5.1 illustrates the architecture of the Fuzzy Filter Network, where X is the vector of the input parameters (presuming a problem with N attributes); C_j and σ_j stand for the center and width parameters of neuron j ($j \in [0, J-1]$), respectively (J being the number of neurons); and $g(x, C_j, \sigma_j)$ denotes the Gaussian activation function of neuron j:

$$g(X, C_j, \sigma_j) = e^{-\frac{\left\|X - C_j\right\|^2}{2 \cdot \sigma_j^2}} \tag{5.3}$$

Remarks: The value calculated by $g(x, C_j, \sigma_j)$ can serve both as a proximity measure, and a fuzzy membership function value that reveals at what rate the given input x is part of the given fuzzy sets that constitute the given rule j. Thus, it can indicate how certain the system regarding any given output y.

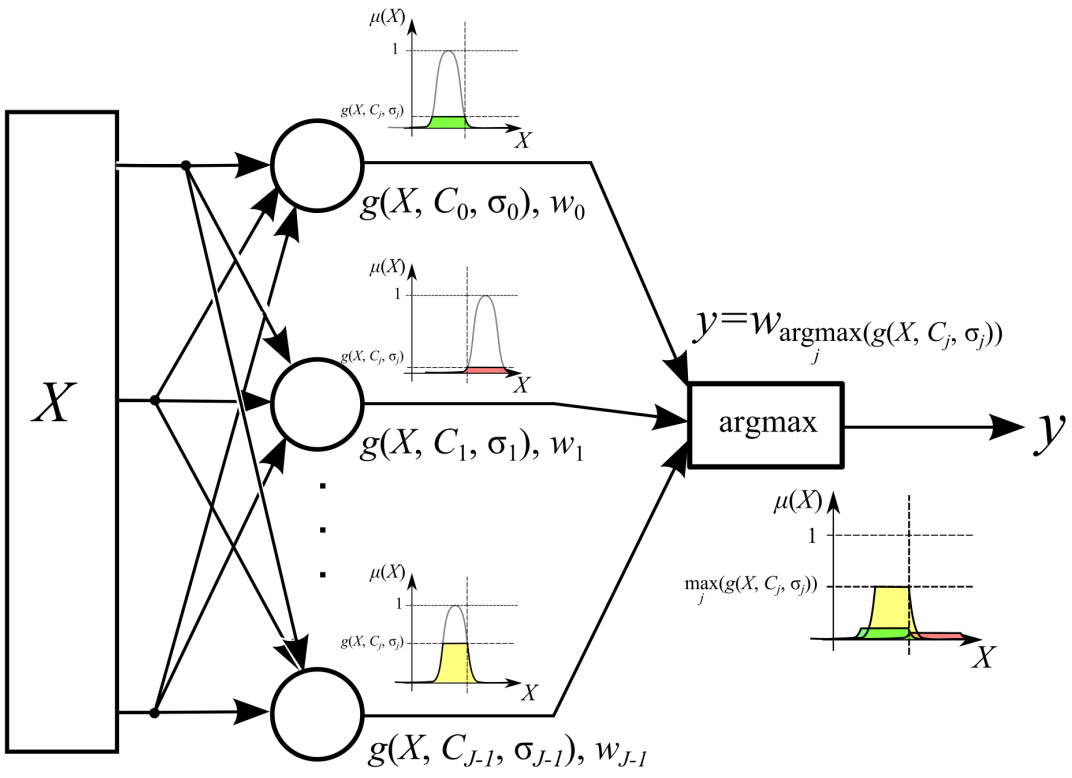


Figure 5.1: The architecture of the Fuzzy Filter Network.

Furthermore, it is easy to see that $g(x, C_j, \sigma_j)$ implements a fuzzy inference (with C_{jk} being the k^{th} coordinate value of C_j) considering fuzzy rule j (that consists of a set of N membership functions for N dimensional problems):

$$g(x, C_j, \sigma_j) = e^{-\frac{||x-C_j||^2}{2\cdot\sigma_j^2}} = e^{-\frac{\sqrt{|X_0-C_{j,0}|^2+\ldots+|x_{N-1}-C_{j,N-1}|^2}^2}{2\cdot\sigma_j^2}} = e^{-\frac{|x_0-C_{j,0}|^2+\ldots+|x_{N-1}-C_{j,N-1}|^2}{2\cdot\sigma_j^2}} = e^{\frac{(-1)|x_0-C_{j,0}|^2}{2\cdot\sigma_j^2}\cdot\ldots\cdot\frac{(-1)|x_{N-1}-C_{j,N-1}|^2}{2\cdot\sigma_j^2}} = e^{-\frac{|x_0-C_{j,0}|^2}{2\cdot\sigma_j^2}} \cdot \ldots \cdot e^{-\frac{|x_{N-1}-C_{j,N-1}|^2}{2\cdot\sigma_j^2}} = \prod_{k=0}^{N-1} e^{\frac{-|x_k-C_{j,k}|^2}{2\cdot\sigma_j^2}} \tag{5.4}$$

which is the *fuzzy product t-norm* [93] of the Gaussian membership functions of the individual parameters ($k \in [0, N-1]$).

The determination of the rule parameters (i.e. the training of the network) can be done by using all the training data points directly (setting the center parameters as the parameters of the data samples ($C_j = x_j$), and either calculating the radiuses from the dataset (based on the distances between data points) or using a single, uniform measure for the radius).
However, it is important to note that the quantity of training data samples can be very high, which in turn slows down the operation of the network significantly. In cases where the network is used in a loop (e.g. in online image processing) the operational speed of the network is very important. One effective way to reduce the set of rules to a more manageable amount is clustering. For this, two new clustering methods are presented in the following subsection.

5.1.2 The Training of the Classifier - Radial Representative Clustering

During the training of the classifier, the goal is to determine the center (C) and width (σ) parameters of each rule belonging to each neuron in the hidden layer.

To achieve this, I have developed a novel clustering method, based on the idea of taking the set of input data points x, considering them all clusters of their own, then removing the ones that are not necessary to represent a reliable covering of the problem space with hyperspheres (i.e. multidimensional areas with the same radius in all dimensions, e.g. circles in 2D, spheres in 3D, etc.). Let us call this the *Radial Representative* clustering principle.

As a result of the training phase, the set of the training samples is divided into two disjunctive subsets: *active* and *inactive clusters*. Active clusters are used as rules in the system, while inactive clusters are either deleted, or kept for future training purposes. In the following, the "activeness" status of a cluster j is marked with Ψ_j. It can take 3 values: indeterminate ($\Psi_j = -1$), inactive ($\Psi_j = 0$) or active ($\Psi_j = 1$).

Remark: Depending on the problem, the dataset that contains the observed knowledge of the problem can be either balanced, or imbalanced [94]. In the former case, the distribution of classes among the samples is relatively even. However, in the latter case, the number of samples from one class outweighs the number of samples from all other classes. This can be taken advantage of: the more numerous class can be considered as a *default* or *background class*, that is the most likely candidate for unknown places of problem (i.e. areas that are not represented in the training dataset). Color filtering problems are such problems, where the task is to recognize given color tones, while the rest of the tones can be regarded as a single, default class that is present where the desired color tones are not. Because of this, only the rules for the non-default classes are needed to be stored, and if the fuzzy output membership function value of the system is less than an arbitrary value, then it can be concluded that the result is most likely the default class.

In the following, let us introduce the concept of **adversity** among samples: Given a dataset X, sample $x \in X$ is **adversarial** towards a given sample $z \in X$ ($z \neq x$), if their classes are different ($\kappa_z \neq \kappa_x$). Since clusters are built around single samples, this definition is applied to them as well: given a set of rules R, cluster $j \in R$ is **adversarial** towards a given cluster $i \in R$ ($i \neq j$), if $\kappa_i \neq \kappa_j$.

The system is set up so at the "border" of each cluster j (i.e. points that are r_j distance from C_j) the value of $g(x, C_j, \sigma_j)$ should be 0.5 (so all points in the domain withing the area of the cluster return a value between 0.5 and 1). Thus, the width (σ_j) of the corresponding rule j can be calculated from radius r_j of cluster j (given Eq. (5.3)):

$$\begin{aligned} 0.5 &= \mathrm{e}^{-\frac{r_j^2}{2\cdot\sigma^2}} \\ \ln 0.5 &= -\frac{r_j^2}{2\cdot\sigma^2} \\ \sigma &= \sqrt{\frac{(-1)\cdot r_j^2}{2\cdot \ln 0.5}} \end{aligned} \tag{5.5}$$

Remark: Before clustering, redundant and inconsistent data is removed from the training dataset. Redundancy is defined as the presence of two or more instance of the same training sample (with the same attribute and class values). Sample x is redundant, if $\exists z \in X$:

$$\forall k \in [0, N-1] : x, z \in X \text{ and } x_k = z_k \text{ and } x \neq z \text{ and } \kappa_x = \kappa_z \tag{5.6}$$

where x_k and z_k are the k^{th} attributes of samples x and z, respectively, and similarly, κ_x and κ_z are the class labels of said samples.

On the other hand, inconsistency is defined as the presence of two or more training samples with the same attribute values but different class values. Sample x is inconsistent

with regards to sample z, if

$$\forall k \in [0, N-1] : x, z \in X \text{ and } x_k = z_k \text{ and } x \neq z \text{ and } \kappa_x \neq \kappa_z \tag{5.7}$$

The removal of the former is done by simply comparing each sample to all others, and removing all but the first occurrence. The latter is a bit more complicated, as it influences the classification accuracy of the classifier trained with the data. Its implementation depends on the problem: for problems with balanced data, one sample in each group of inconsistent samples is randomly chosen and kept while the others are disregarded; while in a case of imbalanced data, it can be simply taken as *default* class (i.e. being more careful not to falsely identify a negative sample as positive).

Initially, two different approaches have been considered when I have designed the Radial Representative clustering method: a *top-down approach* that begins with appointing the biggest possible cluster radiuses and in the following steps, then iteratively reducing the cluster radiuses until an optimal coverage is achieved (in which all input samples are correctly classified using the given clusters); and a *bottom-up approach*, in which the cluster radiuses are started from the smallest necessary sizes, and can only be increased in order to achieve an optimal coverage of the problem space.

The main difference between the two approaches is in their time complexity: since the top-down approach uses an iterative operation to decrease the cluster sizes, then checks all previously appointed clusters in each step to make sure that the change have not reduced the overall classification ability (i.e. making sure that samples that have been correctly classified with the clusters before the change, will be correctly classified after the change as well). This results in a much slower operation, which is why only the bottom-up approach has been implemented and used in the Fuzzy Filter Network.

5.1.2.1 Analytical Radial Representative Clustering

The *Analytical Radial Representative* clustering method approaches the solution in a *bottom-up* manner, in order to gain the center and radius parameters of each cluster. It starts with all training samples being *active* clusters, setting their radiuses to the smallest necessary measure and only *increases* them if there is an opportunity to do so. The detailed algorithm can be followed in Fig. 5.2:

- First *(yellow parts)*, it initializes distance array D (where D_i will store the smallest distance from sample i in the problem domain), and index vector Λ for the indices of the closest adversarial sample j for each sample i (i.e. if j is the closest to i, then $\Lambda_i = j$).

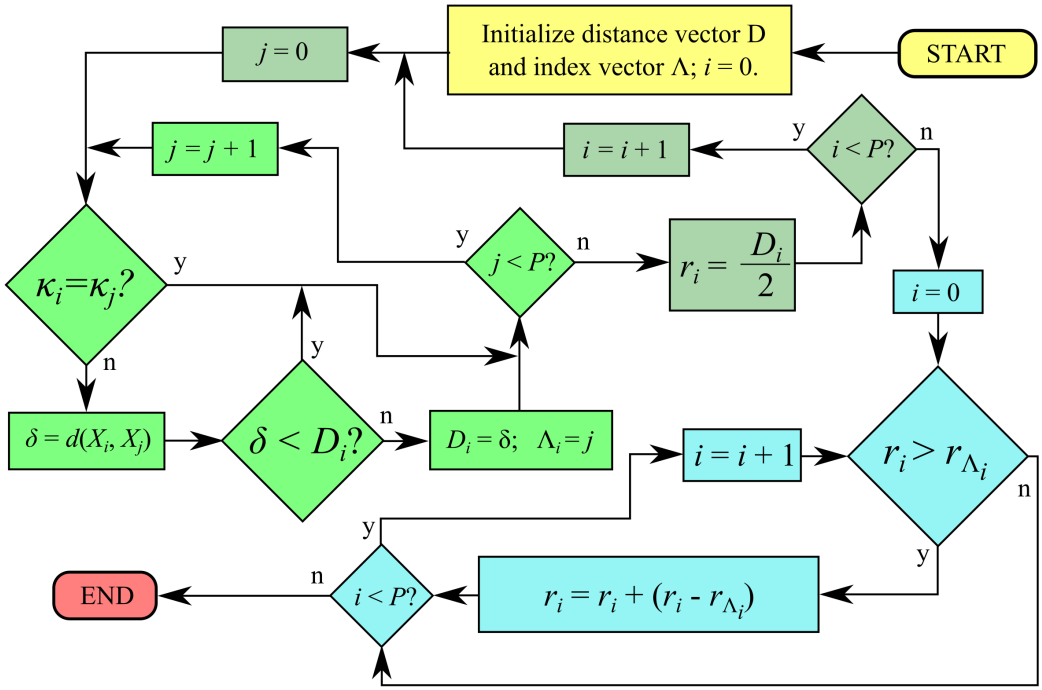


Figure 5.2: The algorithm for the Analytical Radial Representative Clustering approach.

- After that, for each sample, the closest adversarial sample is determined and noted in D, along with the index of the closest sample in Λ_i *(light green cycle)*.
- The radiuses of all clusters are set to half of the smallest found distances (stored in D; *dark green parts*).
- Finally (*cyan cycle*), for each cluster i: where the r_j radius of the closest adversarial cluster j is less than r_i, r_i is increased in order to account for the deficit (so the two borders "meet"). This is done for all P clusters.

Fig. 5.3 illustrates the analytical approach: first the smallest distances are calculated between adversarial clusters *(a)*, which are marked with arrows that point towards the closest sample. Notice that some arrows are pointing both ways, these mark sample pairs that are mutually the closest to each other, so their final radius is set to cover half of their distance *(b)*. However, samples with no arrow pointing at them have the potential to cover a larger area without intersecting with the area of the cluster marked as closest to them (marked with dashed circle). In the last step *(c)* these radiuses are increased, so the final radiuses for each cluster i:

$$r_i = \left\{ d(C_i, C_j) - r_j \mid j = \underset{\forall j \neq i}{\operatorname{argmin}}(d(C_i, C_j)) \right\} \tag{5.8}$$

where cluster j is the closest to cluster i.

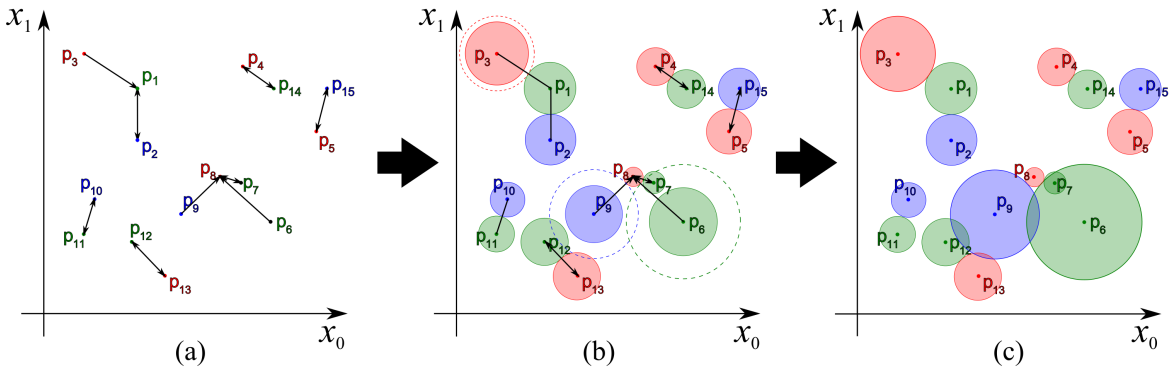


Figure 5.3: Illustration for the *Analytical* Radial Representative clustering method for a 3-class problem: (a) distance calculations, (b) range setting and (c) range increase. The color of each data point p_i shows its class.

Remarks. For *balanced data*, Eq. 5.8 provides the final radiuses. However, for **imbalanced data**, this would lead to skewed results, because the training data often do not cover the whole of the problem domain, so it is highly possible that unseen negative (default) class points would get into the area of the appointed clusters. Thus, for imbalanced data additional calculation is added to limit the radisues by an arbitrary range value ρ:

$$r_i = \begin{cases} r_i & \text{if } r_i \leq \rho \\ \rho & \text{otherwise} \end{cases} \tag{5.9}$$

5.1.2.2 Reduction

After the clustering, there is an additional reduction phase to make sure that the resulting set of clusters only contains clusters that contribute to the accuracy of the network, i.e. clusters that are covered by other (non-adversarial) clusters should be eliminated, since they do not contribute to the classification and so only increase the computational complexity by having to evaluate them as well. Thus, the algorithm compares non-adversarial clusters, and if a cluster i is covered by a larger cluster j, then i is set as inactive. Only active clusters are put into the result of the clustering step.

Fig.5.4 shows the algorithm of the reduction step. It simply compares each active cluster i to all active clusters j that has the same class. If cluster j covers i (i.e. r_i is smaller than r_j even if we add their distance ($d(C_i, C_j) + r_i < r_j$)) then cluster i is set to inactive, and the stops comparing i to any other clusters, goes on to the next cluster instead.

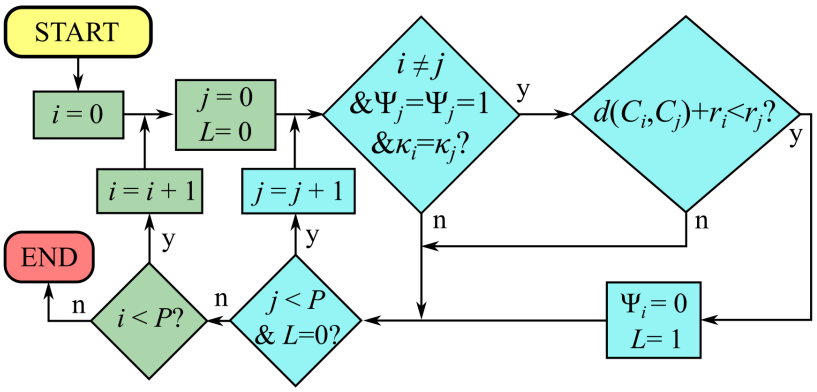


Figure 5.4: The algorithm for the reductions step.

5.1.3 Parallel Fuzzy Filter Network

While the advantages of the FFN classifier are that they are simple to use and modify (as one only needs to add, remove or modify the parameters of the neurons), the main disadvantage is that they require a significant amount of computation, as all clusters are needed to be examined for each input sample. In case of image processing problems, this is a serious issue as typically there are at least hundreds of thousands of pixels in the input image. In order to accelerate the training and operation, I have created a new, parallel implementation of the FFN classifier (called *parallel Fuzzy Filter Netwwork* or pFFN) using parallel computing [P. 9]: the evaluation process is applied to input data (all pixels in the image) at the same time, or at least as many pixels at once that are allowed by the parallel computing capabilities of the given computing platform.

5.1.3.1 Parallel Training

Fig. 5.5 shows the summary of the parallel training algorithm. Just like that of the sequential version, the goal of the parallel training process of the classifier is to build a list of clusters with a manageable size that represents as much of the (non-background class) training data as possible.

- The initial training list can potentially contain samples that are either redundant (Eq. (5.6)) or inconsistent (Eq. (5.7)) with each other, thus the first step is filtering these samples out. This can be done sequentially (comparing the samples one by one with $O(N \cdot P^2)$ time complexity); or in parallel. Depending on the size of the problem and the computational capabilities of the system, it can be fully parallel (if $N \cdot P^2$ processes can be launched at once, then it can be done with a technique called parallel reduction [95], $O(N \cdot \log_2 P)$) or semi-parallel (P processes are launched,

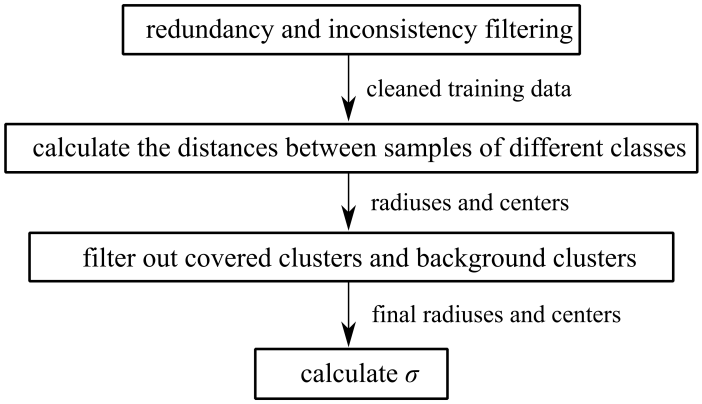


Figure 5.5: The simplified algorithm for the parallel clustering method used to train the parallel FFN.

each compares a given sample to the others sequentially, resulting in an $O(N \cdot P)$ time complexity algorithm).

- The training data (that is clean of any redundancies and inconsistencies) is then processed by the parallel clustering step. The applied new method is a simplified version of the analytical approach presented in Subsection 5.1.2.1: for each sample j, the distance from the nearest adversarial sample i is calculated:

$$r_i = \min_{\forall j}\{d(C_i, C_j) \mid \kappa(j) \neq \kappa(i)\} \tag{5.10}$$

  where $d(C_i, C_j)$ is the same distance metric between the centers of clusters i and j described in previous subsections (Eq. 5.1), while $\kappa(k)$ is the class label of any cluster k.

- Using the obtained data, the list of the clusters is filtered once more: the default class samples are filtered out (in case of imbalanced data), as well as clusters that are covered by larger clusters of the same class.

- Finally, parameter σ_i for each remaining (active) cluster i is calculated from the radial data (using (Eq. (5.5) and (5.9)). This can be done in a single parallel step for all clusters.

5.1.3.2 Parallel Evaluation

Ideally, the evaluation should be done by assigning a process to each input sample – rule combination, but that would require an amount of processes that cannot be provided in most problems nowadays, considering the limited computational power (the limit of the number of parallel processes) of contemporary graphics cards. Thus, the classifier can realistically process the input data in two ways in the evaluation phase:

- **One by one**. This approach is more fitting for streaming data, where the samples are made available one by one. An easy way to handle this in the proposed pFFN is assigning one process for each cluster to calculate the Gaussian activation function with their respective rules, then parallel reduction can be used to get the largest value, along with the index of the rule (so its class can be determined). Since it is fully parallel, the time complexity for ω rules is $O(N \cdot \log_2 \omega)$ (given the time complexity of the parallel reduction).

- **In batches** (i.e. multiple inputs at once). This approach is more typical for image processing problems like color pattern recognition, in which case one process is assigned to a given input sample (e.g. a single pixel and its color triplet). Given the limits on contemporary graphic cards, one process is assigned to each input (e.g. a single pixel), in which each rule is evaluated in a sequentially using (Eq. (5.2)) and (Eq. (5.3)), thus, all pixels can be classified at the same time. Since it is semi-parallel, the time complexity is $O(N \cdot \omega)$.

Similarly to the sequential case, the resulting class label for each pixel is accepted only if the corresponding maximum value is larger than an arbitrary threshold parameter Θ, given by Eq. 5.2.

5.2 Experimental Results

In order to show the capabilities of the network and the new FFN classifier, four experiments are described in this section, of which the first two (regarding the comparison of the IRR and ARR clustering methods) have been conducted on PC-4, and the last two (color filtering) one has been implemented and evaluated on PC-16 using MS Visual Studio 2015, CUDA 9.0 and OpenCV 3.3.0.

5.2.1 Experiment 1: Sequential Color Filtering

The first experiment presents a real application, where the task is to locate the areas that contain given color tones in images. The classification is based on the HSV (Hue, Saturation and Value) color coordinates of the pixels (examined one-by-one). The main goal of this experiment is to examine how fast the trained system can process images sequentially and in parallel, as well as analyze their statistical performance measures [96].
Remark: The descriptions and formulas for the applied statistical performance measures can be found in Section A. of the Appendix. Given that the problem revolves around an imbalanced dataset (the amount of background tones significantly outweigh the sought

ones), thus *balanced accuracy* (Eq.(A.7)) is regarded instead of the classic classification accuracy (Eq.(A.6)).

Another indicator considers the recall and precision ratios [97]: while recall indicates how likely the classifier is to detect the given class, precision shows how "trustworthy" the classifier is in its positive findings.

The problem itself is a 2-class classification problem (as one pixel either falls into such an area or not). Two sequential FFN classifiers have been trained: one using HSV, while the other using the RGB color space. Similarly, two pFFN has been trained as well, however, their results turned out to be the same (with insignificant differences), so here only one set of results are presented. All the networks have been trained with the same range parameter ($\rho = 15$), and the tests have been done for the whole scale of the threshold parameter ($\theta \in [0, 1]$).

The training data set is made by manually marking the important areas (that contain color tones needed to be learned) in one or more training images (e.g. Fig. 5.6 (top left), with resolution 640×480). The system extracts these data points as the sought class and sets the rest of the color triplets in the image as *background colors*. The resulting dataset is then filtered of redundancies and inconsistencies. This yields a much lower number of color triplets (e.g. for the figure, it returned 25138 samples (1898 positive (with a tone of the human skin) and 23240 negative (not a tone of the human skin)). The subsequent training (the modified analytical algorithm) results in 1574 clusters (of the sought human skin areas; the background class clusters are not kept) for the sequential FFNs and 1622 for the pFFN. Interestingly although the two implementations realize the same algorithm overall, but small differences of the code (mainly regarding parallelization techniques) makes the two approaches slightly different.

In Fig. 5.6 (top right) the results of the training can be seen on one of the training images, showing that the new classifier managed to learn the problem. Only the sequential cases are shown in the figure, because there are no significant differences between those and the results of the pFFN.

In the example image sequence shown in the figure, the average evaluation time for each is ~51.7 seconds. As it can be seen, most of the areas are correctly covered. Although, with new color triplets appearing that are not present in the training image, a certain level of distortion emerges in the subsequent images Fig. 5.6 (bottom left and right). On the other hand, the pixels classified as false positive are notably scattered, which still reliably enables the determination of the skin regions by choosing the areas if the results are filtered afterwards for pixel density.

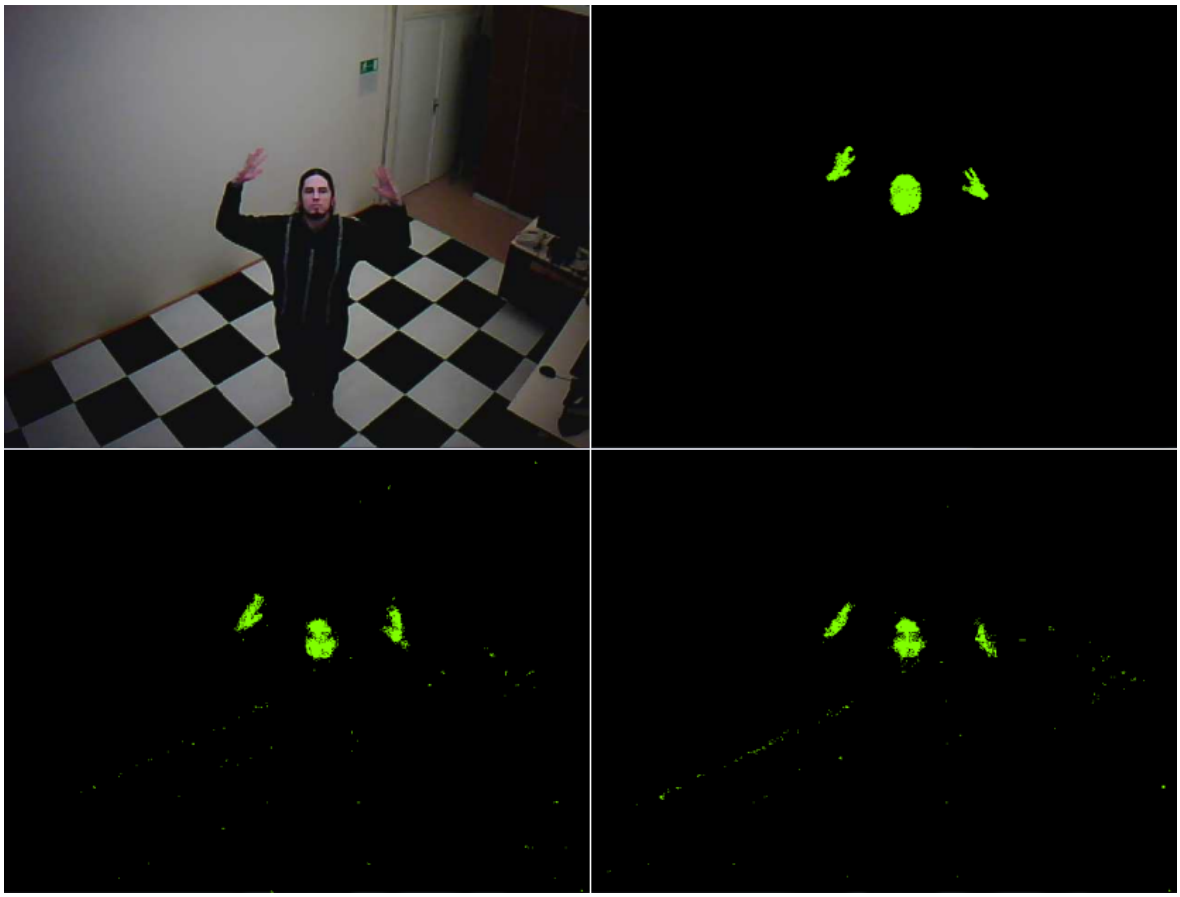


Figure 5.6: The training image of the first experiment (top left), the classification result for the training image (top right) and two subsequent images of the third experiment: the system found skin areas in the highlighted areas (bottom left and right).

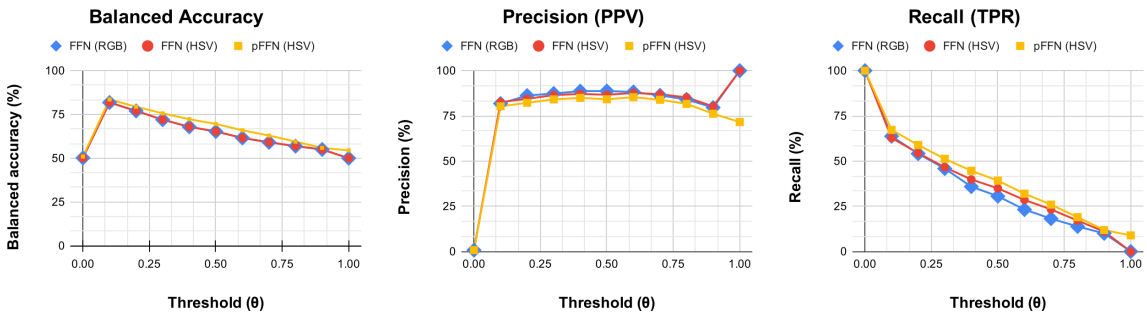


Figure 5.7: Statistical performance measures for the two sequential FFNs and the pFFN: (a) Balanced Accuracy, (b) precision and (c) recall.

In Fig. 5.7, statistical performance measures can be seen that have been gained by using a training image (e.g. Fig. 5.6 (top left)), then evaluating the classifiers for all threshold θ

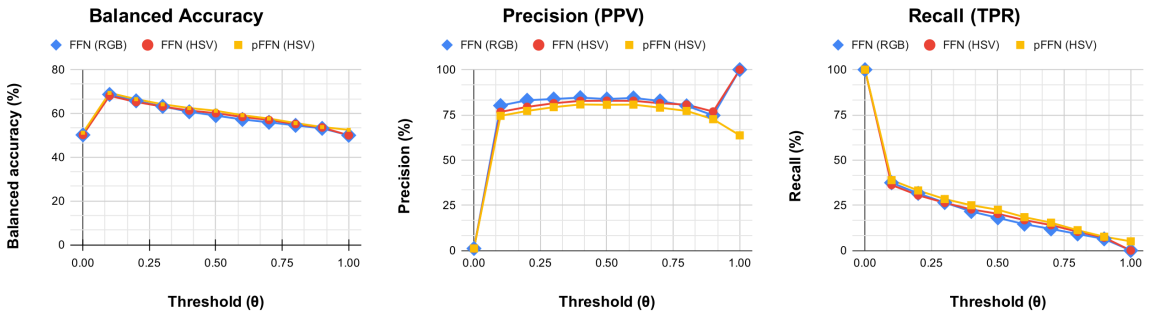


Figure 5.8: Statistical performance measures for the two sequential FFNs and the pFFN: (a) balanced accuracy, (b) precision and (c) recall.

($\theta \in [0, 1]$) values. As it can be seen, the three classifiers have roughly the same balanced accuracy (BA), with the pFFN achieving a slightly higher rate. At $\theta = 0$ the classifiers mark all pixels in the image as positive, which puts their BA around 50% (since they correctly mark all positive areas). This means, however, that even though their recall is 100% at that point (i.e. they mark all positive pixels), their precision is near zero (i.e. the vast majority of the positive findings are wrong). As θ increases, however, their precision stabilizes. They reach their classification peak at $\theta = 0.1$, then their recall steadily decreases (as the filter lets through less and less pixels, as their fuzzy membership values (i.e. their proximity to the closest rule) are regarded with a more and more strict threshold). With this, their BA also slowly decreases, but their precision stays steady until they reach $\theta = 1$, where the recall of the sequential classifiers hit zero (and subsequently, their precision peaks at 100%). The pFFN, on the other hand, manages to provide a small amount of recall, with a decent precision as well.

Fig. 5.8 shows the accuracy of the same networks, but using the image after the previous one in the sequence. As it can be seen, even though the performance values decreased (as new color variants of the sought tones appeared in the image due to the noise of the camera), their general curve shapes stayed the same.

5.2.2 Experiment 2: Parallel Color Filtering

In the second experiment, the performance of the proposed classifier implementations is investigated for a 4-class image processing problem, where aside from human skin areas, the classifiers need to mark to color tones of specific objects too. Similarly to the previous one, the task is to mark areas of images based on color information (e.g. areas with human skin tones). Similarly to the previous one, this experiment has been conducted on multiple image

sequences, one of which is the one used in the previous experiment.

Again, three FFN classifiers (a pair of sequential FFNs using RGB and HSV color spaces, and one pFFN using the HSV color space (again, the RGB tests proved to be the same, with insignificant differences)) have been trained and evaluated multiple times ($\rho = 15$), for the whole range of $\theta \in [0, 1]$.

Using the image in Fig. 5.6 (top left) as a reference example, the two sequential FFNs took ~57 seconds to train on average using the input images with resolution 640 $\times$ 480, resulting in 4119 rules for the RGB, while 4118 rules for the HSV version. The pFFN returned 3666 rules. The parallel training takes ~12.3 seconds on average, of which ~88% is spent with the redundancy and inconsistency removal, which is realized in a semi-parallel way (due to the limitations of the computation ability of the available graphics card in PC-16).

Fig. 5.9 showcases an example of the conducted experiments: the training image (top left) alongside the training mask (top middle) and the testing image (top right). In the bottom row, we can see the output of the sequential FFN for HSV (bottom left) and RGB (bottom middle) color spaces, as well as that of the pFHM. All images show the $\theta = 0.1$ case, when the recall and balanced accuracy are at their height for all three classifiers. The processing of images on average takes ~126 seconds for the sequential FFNs, while for the pFFN it only takes ~0.65 seconds (~193 times faster).

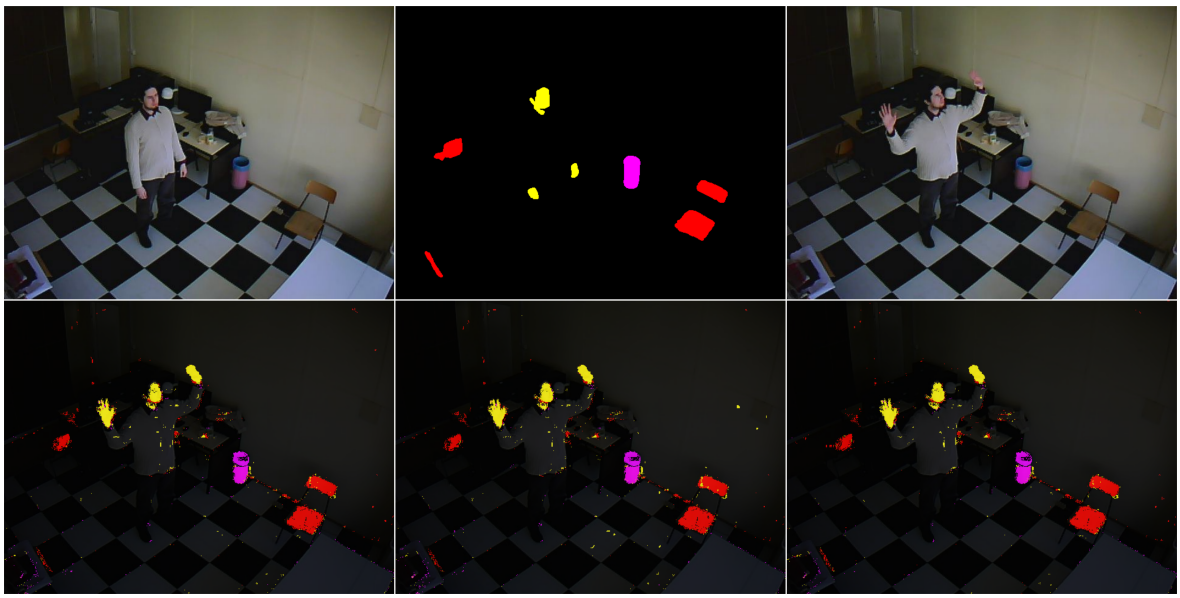


Figure 5.9: A training image (top left), the training mask (top middle) and the testing image (top right), the output of the sequential FFN for HSV (bottom left) and RGB (bottom middle) color spaces, as well as that of the pFHM (bottom right).

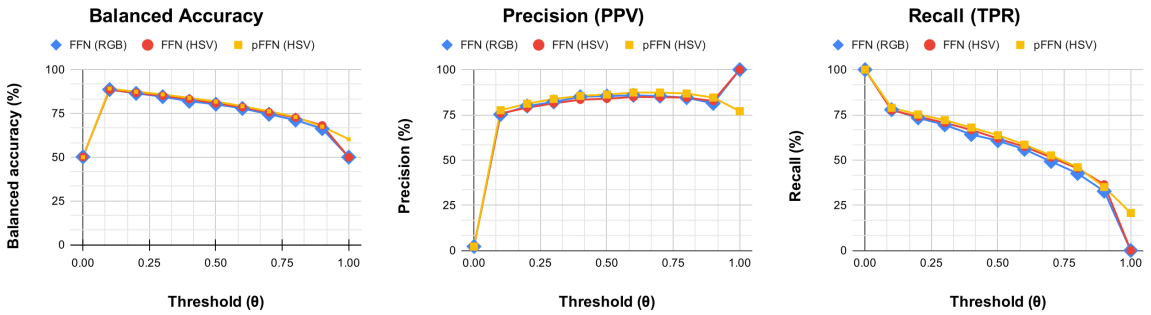


Figure 5.10: Statistical performance measures for the two sequential FFNs and the pFFN: (a) balanced accuracy, (b) precision and (c) recall.

Fig. 5.10 sums up the performances of the classifiers for all θ values, which is similar to what the previous experiment presented. However, the classifiers managed to achieve a higher balanced accuracy (peaking at ~88-89% at $\theta = 1$) and overall higher precision. Their recall rate is above 75% around $\theta = 0.1$.

The speed of the trained classifier was also tested on a single (non-batched) input, where the parallel operation can be fully exploited (each process computing a comparison to a single cluster). The time required to evaluate a single input (a color tone) takes ~0.019 ms (an $O(\log\omega)$ time complexity).

Remark: although it might seem reasonable to just use this latter (non-batched) method to process the image pixel by pixel considering it only takes ~21.88 μs, but that can add up even for a small image to more than what the batch processing provides (e.g. it takes ~5.836 s/image for an image with size 640×480, as opposed to the ~0.305 s/image speed of the batch (whole image) processing).

5.3 Evaluation and Applicability

The time complexity of the Analytical Radial Representative clustering algorithm is linear in the number of attributes (N, which influences the distance calculations) and quadratic in the number of input samples (P)): $O(N \cdot P^2)$.

The operational time complexity of the sequential FFN (which involves sequentially computing the distance of the input data points from each of the ω rules and taking the rule that is the closest to the given inputs) is $O(N \cdot P \cdot \omega)$. It is significant for a large number of inputs, like in image processing (as Experiment I. shows), even for relatively small images (640×480).

The *parallel FFN*, on the other hand, has the operational time complexity of $O(N \cdot \omega)$,

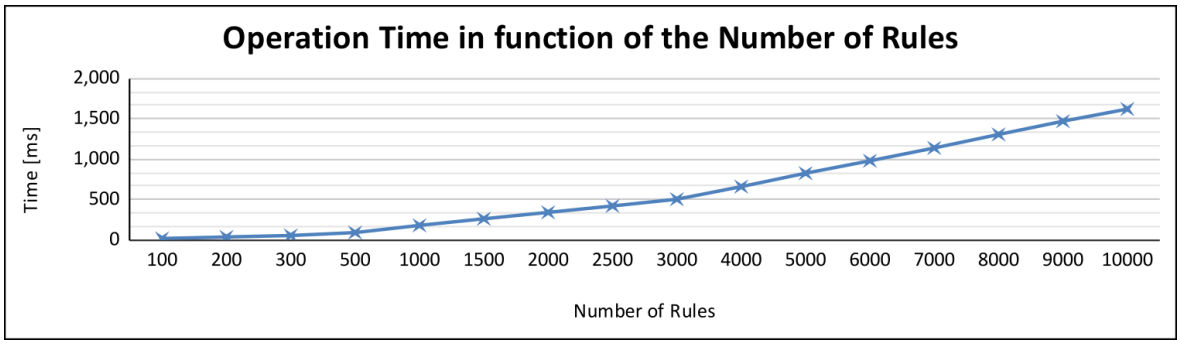


Figure 5.11: The operation time of the pFFN classifier in function of the number of rules (ω).

which is a significant reduction given that the number of neurons (ω) is usually much smaller than the number of input samples (P). In practice, this means that instead of more than 10 seconds, the time required is reduced to ~0.3 seconds. Fig. 5.11 visualizes the dependence of the operation time (of the semi-parallel implementation) on the number of rules (i.e. ω). It is based on the averages of 48 evaluation sessions using the images from the sequence shown in the previous section. As it can be seen, it shows a near linear increase in the microscopic level (though with slowly increasing degree of slope), but on the macroscopic level, this implies an exponential increase. This shows the importance of finding a rule-base for a given problem with optimal cardinality: one with the lowest number of rules that can sufficiently cover the problem space (balancing between classification performance and speed).
Remark: It is still worth noting that even for 10000 rules, the operation time stays below 2 seconds (on PC-16).

5.4 Summary of New Results

In this chapter, I have presented a new *rule-based classifier* is proposed for supervised classification problems, the so-called **Fuzzy Filter Network (FFN)**. The new classifier is derived from the classic *Radial Basis Function (RBF) networks*: using the same structure, but instead of gaining the output value from the weighed and activated outputs of the neurons, the FFN classifier uses each neuron to calculate the distance between the inputs and a given rule. The output of the system is thus the class associated with the rule that is the closest to the inputs, i.e. the one that is the most similar to known models.

The training of the classifier entails the determination of the parameters of the rules that belong to each neuron in the hidden layer, which can be done through clustering. In order to implement the training, I have developed a **new clustering method**, which aims to make a covering of the problem space with hyperspherical multidimensional clusters. The two main parameters of the clusters are the center and range parameters, which are

directly used in the neurons. In order to gain these parameter values, the new so-called **Analytical Radial Representative** (ARR) clustering method designates all input data points as an individual cluster center, sets their radiuses to half the distance to the closest cluster center (of a different class), then increases them to cover as much of the problem space as possible, without significant overlapping between the areas covered by the individual clusters.

Finally, in order to enhance the speed of the classifier, I have designed **a parallel version of the FFN** (pFFN), alongside with the parallel version of the ARR clustering method. The performance of both sequential and parallel FFNs are illustrated through experiments.

5.4.1 Thesis Group IV.

5.4.1.1 Thesis Statement IV.1.

I have proposed a new fuzzy rule-based classification method called Fuzzy Filter Networks that is based on a topological modification of Radial Basis Function neural networks. I have also introduced a new clustering algorithm called Analytical Radial Representative clustering in order to train the new fuzzy rule-based classifier. The algorithm follows the bottom-up approach.
Its training is done through clustering, where the clusters that cover the problem domain based on the training data are calculated. Said clusters are used as rules during the operation of the classifier, during which each hidden layer neuron calculates a Gaussian fuzzy membership function value based on each rule. The output of the classification is the class label associated with the rule that is the closest to the input values. The classification performance of the filter network using the proposed clustering algorithm as training has been evaluated through a color filtering experiment.

Related publications: [P. 3], [P. 22], [P. 8]

5.4.1.2 Thesis Statement IV.2.

I have designed a parallel realization of the new Fuzzy Filter Network classifier, which reduces the complexity of both the training and the operation of the classifier. The efficacy of the method is shown through a color filtering experiment.

Related publications: [P. 9]
Number of independent citations as of May 2021, according to Google Scholar: 2

Chapter 6

Real-time Classification with Fuzzy Hypermatrices

Classifiers are regularly used in most fields of science and engineering. The majority of classifiers in literature apply a complex algorithm to calculate the output values from the input values (such as ANNs/MLPs), or apply a proximity-based approach to find a known rule or model that resembles the most to the input data vector (such as Fuzzy Filter Networks [P. 3]). The disadvantage of these methods is that the more complex their evaluation algorithm is, the slower it usually gets in function of the amount of input data.

There is, however, an approach that aims to minimize the amount of computations for each input data sample. *Lookup Tables* (LUTs) are structures that implement a direct association between the input data vector and the desired output, in the form an array that uses the input data attributes as array coordinates, and stores the desired values (either class IDs or precalculated function values) in the corresponding array element.

The LUT method is most often used in network routing applications (e.g. [98]), but it also has been successfully applied for color filtering in image processing applications, typically using a matrix-like table to store the 2D histogram of certain attributes (see, e.g. [32]). The two attributes can be chosen arbitrarily from a color system; however, they are generally chosen not to correspond to illumination or intensity, in order to avoid complications caused by various lighting effects, like shadows. The table is filled with the number of occurrences belonging to each color tones in the training images then normalized by the largest value, thus creating a 2-parameter likelihood function that determines the probability of each color tone in one step. For color filtering, the problem space is generally a 3D array and the color coordinate values (RGB, HSV, etc.) provide the address to the given elements.

One of the most significant advantages for LUTs is that they can be implemented as hardware [99] [100] using a multiplexer (for simple cases) or a field-programmable gate array

(FPGA [101], for higher dimension problems); directly integrating them into the measuring process and thus the operation even faster.

In this chapter, the results of a research are presented that aimed to create a classifier that has a lower computational complexity than most other classifiers, in order to achieve a very fast operation for real-time systems. The new classification method, called **Fuzzy Hypermatrices** (FHMs, [P. 4]), utilizes the LUT method for classification, but while the previously described color filtering technique ([32]) uses probabilities (which are affected by the frequency of occurrences, i.e., the more times a color tone appears the more likely it belongs to the given class), the new method applies fuzzy logic to determine the degree of certainty in each cell of the array (thus, the number of occurrences of a tone does not matter at all). By this, a certain degree of generalization ability can be added to the classifier.

Furthermore, **I have developed a parallel version of the FHM classifier** that has been shown to be able to process images with a significantly lower time complexity than that of the sequential version, and provide a significantly better classification performance than parallel Fuzzy Filter Networks [P. 9] (a parallelized improvement of an classifier from my previous work).

The rest of the chapter is as follows: Section 6.1 describes the general architecture and training of the classifier, while Section 6.2 proposes a parallelized version. Section 6.3 shows experimental results, Section 6.4 investigates the applicability of the proposed method and lastly, Section 6.5 summarizes the new results.

6.1 Fuzzy Hypermatrices

The new method can be defined as a multi-dimensional lookup table approach that is enhanced with *fuzzy logic*. This is done by regarding not only the input data points, but the area around them in the problem space as well, and thus, the classifier can generalize from the knowledge it gains through training (which means that it can recognize inputs that are similar to the ones it has already learned).

6.1.1 The General Architecture

The general architecture of the classifier uses one multidimensional array (so-called *hypermatrix*) to represent the implicit knowledge of the system derived from the training data (i.e. the known positions of the individual classes in the problem space (Ψ), e.g. color tones in the 3D color space) and as many additional hypermatrices (with the same sizes as the former) as many classes are needed to be distinguished, in order to store the

precalculated fuzzy membership function values of the classes (hence the labeling *Fuzzy Hypermatrices* (FHMs)).

Let us denote the hypermatrix containing the known locations of the classes (in the problem domain Ψ) with C, and the arrays that hold the fuzzy membership values with M_κ (where κ is the label of the appropriate class). C is used during the training phase(s), while the evaluation step requires only M_κ.

Many filtering problems have to deal with *imbalanced* data [94], in which the samples of one class (usually designated as the *negative*, or as in this thesis work, the *default* class) significantly outweighs the number of one or more positive classes. The advantage of this is that the default class does not have to be stored, since in theory it can be presumed that it fills every place in the problem domain that the other classes do not. This slightly reduces both the storage space that is required for the structure and the evaluation time. Latter is important if the classifier is applied in a loop (e.g. pixel-wise classification of an image) during real-time processing.

In general, the output of the system considering N input parameters and a default class is as follows [P. 23][P. 24]:

$$y(x_0, ...x_{N-1}) = \begin{cases} \lambda & \text{if } M_{x_0,...x_{N-1},\lambda} \geq \theta \\ default & \text{otherwise} \end{cases} \tag{6.1}$$

where $x_0, ...x_{N-1}$ correspond to the input parameter values and θ stands for an arbitrary threshold value that is used to set the sensitivity of the system (typically 50%). λ is the identifier of the class with the highest stored fuzzy membership function value, which can be calculated as

$$\lambda = \underset{\forall \kappa}{\operatorname{argmax}} \left(M_{x_0,...,x_{N-1},\kappa} \right) \tag{6.2}$$

Thus, the usage of the classifier for multiclass cases is simply checking the values of the fuzzy hypermatrices of each class considering the input parameter values ($x = x_0, ..., x_{N-1}$): the class of the highest such value (that is higher than the arbitrary threshold θ) is the output. For 2-class problems it is even simpler, since there is only one FHM to check.

Fig. 6.1 shows an illustrative example for a 4 class (with class 0 being the default class) case with 2 attributes. In the figure, *(a)* shows the problem domain where each class has its markers pinpointing places where the samples occur in the training data set. As a general rule of thumb, if a place would be occupied simultaneously by two different classes according to the data set (i.e. inconsistency occurs) then the marker is considered as default instead. Fig. 6.1 *(b)*, *(c)*, and *(d)* show the fuzzy hypermatrices for classes 1, 2, and 3, respectively. Here, linear (triangular) fuzzy sets are applied, with their center being at the position of the class marker and their area spreading till the closest adversarial marker.

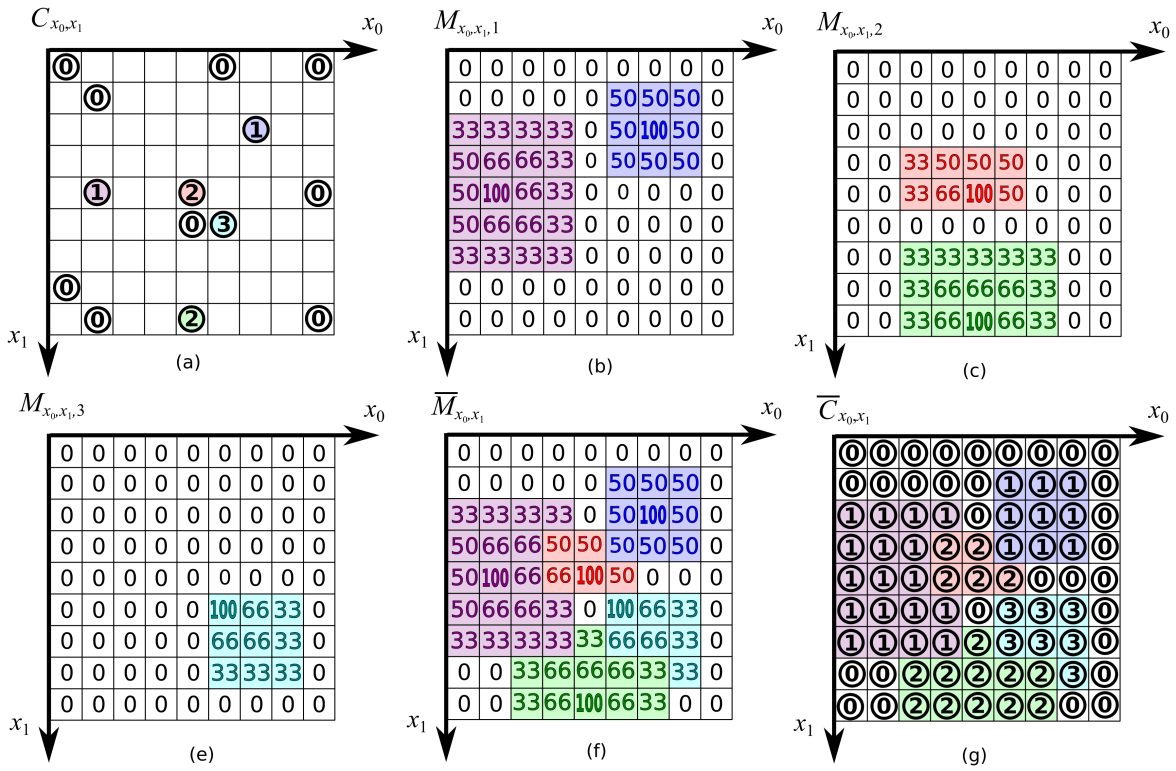


Figure 6.1: An example for a 2D FHM classifier with 4 classes (considering class 0 as default class, thus it does not have an M_0 associated with it): (a) 2D array of the classes; (b,c,d) fuzzy arrays of the fuzzy membership functions; (f) depicts how the largest fuzzy values of each of the classes cover the problem space, and (g) shows the resulting covering in terms of the class labels.

Manhattan distance (or taxicab geometry) [102] is used to limit these areas (by an arbitrarily chosen range parameter ρ) further regulating the generalization ability of the system (against overgeneralization) in case of sparse matrices. If a class contains two overlapping fuzzy sets, the values at the overlapping areas are not aggregated; the larger one is used instead. Character type arrays can be stored more efficiently than integer or floating point arrays, hence this research uses values from 0 to 100 to describe the fuzzy sets.

Fig. 6.1 (f) and (g) shows how the structure of the FHM looks like from the perspective of the evaluation, i.e. the highest fuzzy membership function values ($\bar{M}_{x_0,x_1}$) returned at each element (f), alongside their classes ($\bar{C}_{x_0,x_1}$) (g).

6.1.2 Training

During the training of the FHM, imbalanced data is presumed, so the goal is to make a model of the non-negative areas of the problem space Ψ (i.e. where the non-negative class labels from the training data an be found).

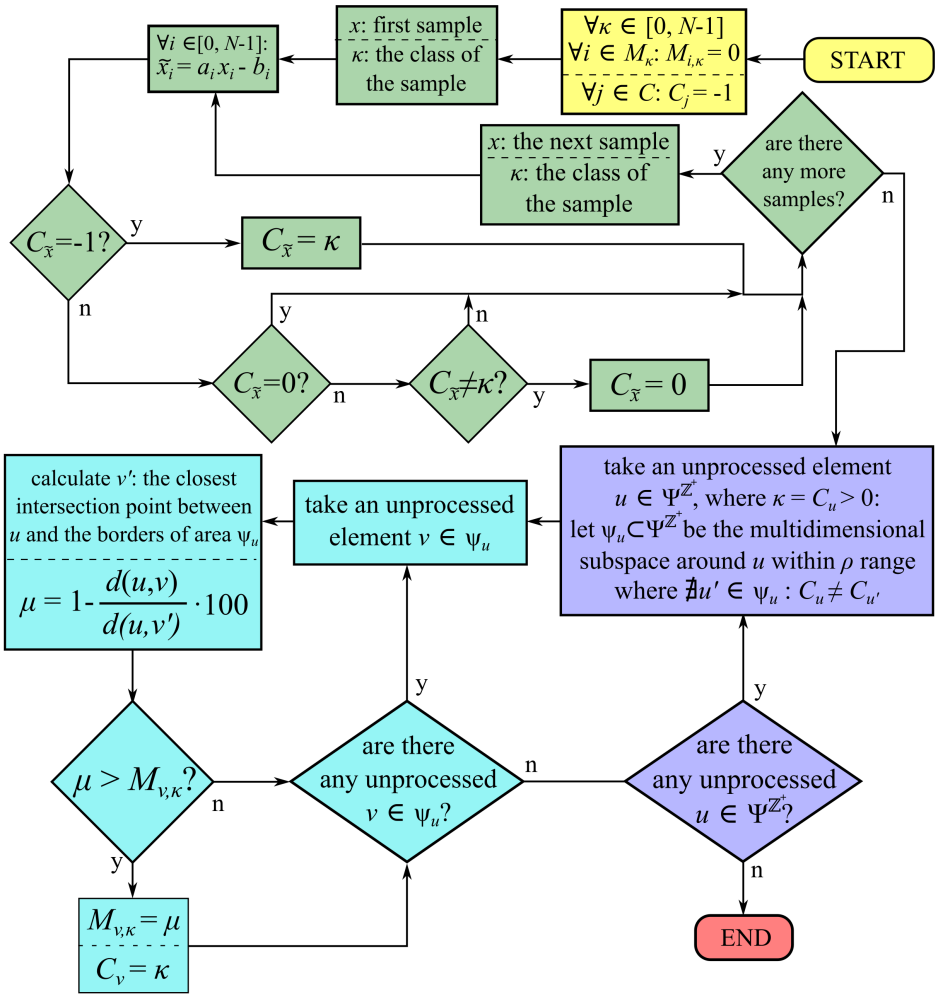


Figure 6.2: The training algorithm behind Fuzzy Hypermatrices.

The training consists of two major parts: filling one class array C with the class labels of the input data using the input data values to get the location of the appropriate element; then using the contents of C, fill each fuzzy arrays M_κ around the non-negative labels.

Remark: In the next algorithm, the following denotations are used for the elements of the given arrays: $C_x = C_{x_0,..,x_{N-1}}$ and $M_{x,\kappa} = M_{x_0,..,x_{N-1},\kappa}$, i.e. for the sake of simplicity, the input vector is written instead of listing its attributes.

Fig. 6.2 shows the algorithm of the training. It begins with initializing the arrays: setting all contents of M and C to -1's and 0's, respectively. Then, the first sample x is taken alongside its class κ, and mapped to a given positive integer domain:

$$\tilde{x}_i = a_i x_i - b_i \tag{6.3}$$

where a_i and b_i are arbitrary coefficients for each attribute and thus, $\tilde{x}_i \in \mathbb{N} \cup \{0\}$, $\forall i \in [0, N-1]$. Their values are chosen regarding the data at hand. Remark: for integer valued

data with already limited domain (e.g. color triplets) this step can be skipped.

The gained integer valued vector is used to examine element $C_{\tilde{x}}$:

- If its value is -1, that means these values have not been processed yet (a new color tone), the element is set to its class κ.
- If its value is 0, then the same values have already been seen with the *default* class (i.e. a color tone from the background), so nothing will be done to it (using a pessimistic approach, i.e. in case of inconsistencies the system presumes that the tone must be of the default class, instead of a sought tone, due to noise).
- If its value is greater than 0 but does not match κ, then using the same pessimistic approach as above, set it to 0.

This is done to all samples. If there are none, the system changes aspect and instead of the list of the input values, it begins looking over the elements of class array C (i.e. the elements of the problem space Ψ at positive integer ($\mathbb{N} \cup \{0\}$) points). For each element $u \in \Psi^{\mathbb{N}\cup\{0\}}$ that is marked positive ($C_u > 0$), the largest subspace $\psi_u \subset \Psi^{\mathbb{N}\cup\{0\}}$ is considered that is around u within a range of ρ that does not contain any u' elements that are adversarial towards u (i.e. their class label is different form that of u).

Thus, a reduced multidimensional area is determined around u, in which the fuzzy membership function values of each element v ($v \neq u$) is calculated using their relative distance from u and from the border of the area. Remark: this allows for asymmetrical areas, as it can be seen in e.g. Fig. 6.1 (e), where one class 3 can only "spread" towards one direction due to the limiting factor of the other classes. The calculation of v' is done by finding the intersection point between the line defined by uv and the bounding box of the area (which can be done with line clipping algorithms [103][104]).

Using the resulting vector v', for each $v \in \psi_u$ in this area the fuzzy membership function (using triangular fuzzy sets) is calculated:

$$\mu = 1 - \frac{d(u,v)}{d(u,v')} \tag{6.4}$$

If μ is larger than the previously held $M_{v,\kappa}$ value (κ being the class associated with u), then $M_{v,\kappa}$ is overwritten by μ and C_v is set to κ. This is done for all $v \in \psi_u$.

Remarks: The shape of the applied fuzzy membership function is arbitrary (e.g. *triangular, Gaussian, Epanechnikov* ([105], see Fig. 6.3), etc). Inconsistency filtering is implicitly part of the training algorithm, as it can be immediately detected if a color is already contained by array M or not.

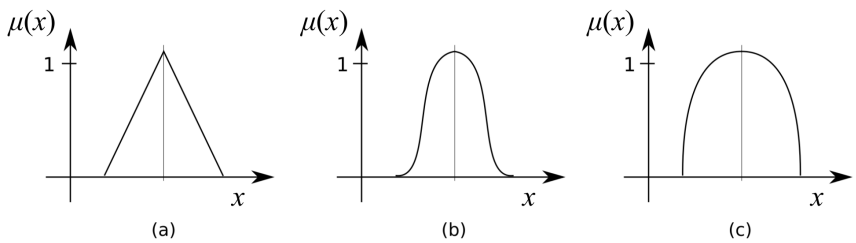


Figure 6.3: (a) Triangular, (b) Gaussian and (c) Epanechnikov fuzzy membership functions.

6.1.3 Evaluation

As it has been mentioned, the evaluation of the FHM classifier is simply checking the values of the fuzzy arrays for each input samples in a sequential manner, and accepts the class for which this value is the largest. Fig. 6.4 shows the evaluation algorithm in detail. As mentioned, it processes the input set X sample by sample. First, the an unprocessed input is taken and mapped with Eq.(6.3) to gain $\tilde{x}$, which is then used to find the largest value of $M_{\tilde{x},\kappa}$:

$$\mu = \max_{\forall\kappa} M_{\tilde{x},\kappa} \tag{6.5}$$

If μ is at least θ, then the class of the largest value of the fuzzy matrices is marked as the output for sample x:

$$y = \operatorname*{argmax}_{\forall\kappa} M_{\tilde{x},\kappa} \tag{6.6}$$

Otherwise, 0 is returned for sample x. This is done till all samples are processed.

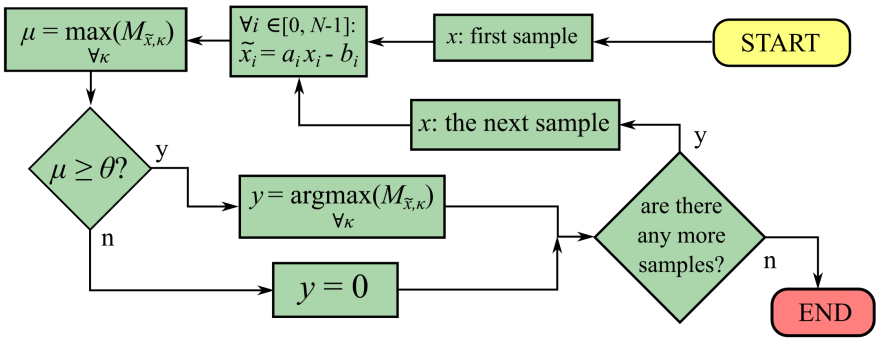


Figure 6.4: The evaluation algorithm behind Fuzzy Hypermatrices.

6.2 Parallel FHM Color Filtering

Although the sequential FHM is very fast, it can still be accelerated by evaluating all input samples concurrently. In this section, a new algorithm is given for both the training and evaluation that uses parallel computing.

Similarly to that of the sequential case, the structure of *parallel Fuzzy Hypermatrices* (pFHMs) consist of one class array $\widehat{C}$, but implemented in the form of a single 1D vector, to make it easier to handle for parallel functions. The elements of $\widehat{C}$ are addressed as follows: let z be an element of the positive integer valued problem space ($z \in \Psi^{\mathbb{N}\cup\{0\}}$). The position of z in $\widehat{C}$ is given by the following function:

$$A(z) = \sum_{i=0}^{N-1} \left(z_i \cdot \prod_{j=0}^{N-1-i} D_j \right) \tag{6.7}$$

where D_i is the size of the domain of dimension i of the problem space. For example, using a suitable color space, the element of color tone $z = [1\ 2\ 3]$ ($z_0 = 1$, $z_1 = 2$, $z_2 = 3$) can be found at $A(z) = 1 \cdot 256^2 + 2 \cdot 256^1 + 3 \cdot 256^0 = 66051$.

Furthermore, the pFHM structure has a single fuzzy array $\widehat{M}$ (with the same size as $\widehat{C}$), in which the fuzzy information of all classes are aggregated.

6.2.1 Parallel Training

During the training, a temporary class and fuzzy arrays $\breve{C}$ and $\breve{M}$ are created and filled. These arrays have Q_κ times the size of the final arrays: they are divided into Q_κ *sectors*, and each sector is representation of the problem space regarding a given class.

Fig. 6.5 shows an example: in the figure, each sector contains the color space (with size $D = 256^3$). The idea behind this is that each class is calculated simultaneously, then the resulting structure is filled with the largest values from each class.

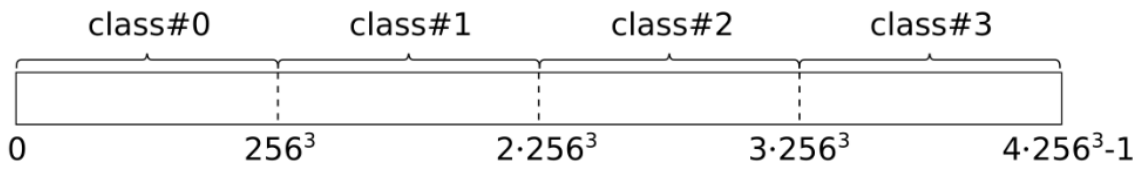


Figure 6.5: The architecture of the arrays in the pFHM model (for color filtering). In practice, the most efficient way to implement it is as a 1D array, where each 256^3 sized block belongs to a given class.

Although there is an additional dimension (the class label) κ, the address function can easily be used to gain the given element in $\breve{C}$ and $\breve{M}$:

$$A(z,\ \kappa) = \kappa \cdot \prod_{j=0}^{N-1} D_j + \sum_{i=1}^{N-1} z_i \cdot \prod_{j=0}^{N-1-i} D_j \tag{6.8}$$

Using a color filtering example again, is $\kappa = 2$ and $z = [1\ 2\ 3]$: $A(z, \kappa) = \kappa \cdot 256^3 + z_0 \cdot 256^2 + z_1 \cdot 256 + z_2 = 33620483$.

After setting up the class array $\breve{C}$, it is filled with the input data, with one process launched for each data point x the training dataset X:

$$\breve{C}_{A(x,\ \kappa)} = 1,\ \forall x \in X \tag{6.9}$$

where κ is the class of x. Remark: non-integer inputs x are mapped to $\tilde{x}$ beforehand using Eq.6.3, so in the following $\tilde{x}$ is used.

The next step is inconsistency filtering, in which all elements z in the positive integer problem space $\Psi^{\mathbb{N}\cup\{0\}}$ are regarded across all sectors (i.e. looking at what classes have been associated with it):

$$\forall z \in \Psi^{\mathbb{N}\cup\{0\}}, \forall \kappa \in [1, Q_\kappa - 1],\ \exists \kappa' \in [0, N-1],\ \kappa \neq \kappa',\ C_{A(z,\kappa)} \neq 1,\ C_{A(z,\kappa')} = 1 : \\ C_{A(z,\kappa)} = 0 \text{ and } C_{A(z,0)} = 1 \tag{6.10}$$

i.e. if there is another class ($\kappa' \in [0, N-1]$, $\kappa \neq \kappa$') that is marked as 1 ($C_{A(z,\kappa')} = 1$), then there is inconsistency: element z needs to be set to the default class ($C_{A(z,0)} = 1$ and $C_{A(z,\kappa)} = 0$ ($\forall \kappa \in [1, Q_\kappa - 1]$)).

After this, the array $\breve{M}$ for the fuzzy membership function values is created (with the same size as class array $\breve{C}$), which is filled in the areas around each element z where $C_{A(z,\kappa)} = 1$, $\forall \kappa \in [1, Q_\kappa - 1]$ (i.e. only non-default classes are regarded for pFHMs as well).

Let $\psi_z \in \Psi^{\mathbb{N}\cup\{0\}}$ be the positive integer subspace around z, in which the fuzzy values are needed to be calculated. For each element z' in subspace ψ_z, the fuzzy membership function value is calculated using a suitable function (e.g. the following is the Epanechnikov function [105]):

$$\breve{M}_{A(z',\kappa)} = 1 - \frac{d(z,z')^2}{\rho^2},\ \forall z' \in \psi_z,\ \breve{C}_{A(z,\kappa)} = 1,\ \kappa > 0 \tag{6.11}$$

Remark: Although it is possible to take adversarial elements into account just like in the case of sequential FHMs, but in order to reduce the necessary training time, the presented pFHM implementation uniformly takes the area around the given elements (within ρ range).

Finally, the sectors of arrays $\breve{C}$ and $\breve{M}$ are compressed into arrays $\widehat{C}$ and $\widehat{M}$, by selecting the element with the highest fuzzy value in each class sector:

$$\widehat{C}_{A(z,0)} = \left\{ \breve{C}_{A(z,\kappa)} : \kappa = \underset{\forall j}{\operatorname{argmax}} \left(\breve{M}_{A(z,j)} \right) \right\} \tag{6.12}$$

for all coordinates $a, b, c \in [0, 255]$. The final fuzzy array $\widehat{M}$ is calculated similarly:

$$\widehat{M}_{A(z,0)} = \left\{ \breve{M}_{A(z,\kappa)} : \kappa = \underset{\forall j}{\operatorname{argmax}} \left(\breve{M}_{A(z,j)} \right) \right\} \tag{6.13}$$

6.2.2 Parallel Evaluation

In the evaluation step, a process is launched for input sample $\tilde{x}$ is given by:

$$y = \begin{cases} \widehat{C}_{A(\tilde{x},0)} & \text{if } \widehat{M}_{A(\tilde{x},0)} \geq \theta \\ 0 & \text{otherwise} \end{cases} \tag{6.14}$$

This is done for all input samples at once (as long as the number of inputs is less than the number of processes the given graphic cards can handle).

6.3 Experimental Results

In the following, in order to demonstrate the performance of the FHM classifier, three series of image processing related examples are presented (conducted on PC-4) and the results of one set of experiments is shown to illustrate the effectiveness of pFHMs (using PC-16).

6.3.1 Sequential Experiments

In the first series of experiments, the speed and accuracy of the sequential classifier is investigated. The goal of this color filtering problem is to recognize human skin areas in camera pictures. This task is important for applications like human face and facial emotion recognition, hand posture and gesture identification for natural human-computer interfaces, etc.

Thus, the objective is to find and mark the pixels that are in certain regions of the 3D (HSV) color space, while ignoring the ones that are outside of these regions. Let us consider the former as positive, and the latter as negative (i.e. the default class). The training data has been generated from images (see, e.g. Fig. 6.6) where the locations of the human skin areas (the samples of the positive class) are manually marked. Each pixel in the images (with resolution 640×480) is a sample, of which its HSV color coordinates are considered together

with its class. Moreover, since in this case the negative class can be considered as the default class, it is sufficient to calculate and store the positive FHM only.

6.3.1.1 Comparison with Classic Lookup Tables

In the first set of experiments the training speed, classification speed and performance of the FHM is compared to that of the standard lookup table (LUT) method. The training data is from a sequence of images taken in the former Intelligent Space Laboratory of Óbuda University. In the training images, the human skin areas have been manually marked, so the task is to identify these areas in the image. Since noise from the camera can be expected, the goal is to mark the pixels with the sought color tones in a high enough density so they can be found using simple image processing methods after classification, e.g. morphological tools.

Two different configurations are used for the LUT classifier: one using all 3 components of the HSV color space, and a classic LUT that only uses the hue and saturation components (in order to improve the classification rate by neglecting the component that involves luminance).

Figure 6.6: A training image for the first experiment (top left), classification accuracy of the trained Fuzzy Hypermatrix ($\rho = 5$, top right) classifier, as well as the trained 2D (bottom left) and 3D Lookup Table (bottom right) on the next image of the sequence.

The applied LUT training algorithm (with a minor modification in order to take inconsistency into account) is the following. For each pixel marked as positive in the training image, increment the value associated with its color tone, otherwise decrement it. After finishing all pixels in the image, replace all negative values in the matrix by 0, then normalize all values by the largest value. The inputs of the training algorithm are the same as in the previous experiment.

Table 6.1 shows the average speed results. The training of the 2D LUT classifier takes 4.918 seconds on average per image, while the 3D LUT requires slightly more, 5.044 seconds, more or less the same as the $\text{FHM}_{\rho=5}$. The classification speed of the 2D LUT is 0.092 s on average per image, while the 3D LUT is 0.138 seconds, same as the $\text{FHM}_{\rho=5}$.

Fig. 6.6 highlights the results for the regarded classifiers using one of the images as input testing input (top left). The classifiers have been trained with the image before it in the sequence. As it can be seen, the $\text{FHM}_{\rho=5}$ managed to mark the skin regions fairly densely, while the other two performed worse, as they could only recognize the color tones they have been trained with, while the FHM was able to generalize its obtained knowledge to tones it have not seen during the training. This proves the base presumption that adding fuzzy logic can implement a certain level of generalization ability in a classifier that had none by default. Another interesting thing to note that considering the LUTs, disregarding the luminance color parameter can indeed lead to slightly better results, as it makes the classifier slightly less sensitive to any changes in shading.

6.3.1.2 Investigating the Effect of the Range Parameter

In the second set of experiments, the multi-class classification capabilities of the FHM classifiers are demonstrated, for FHMs trained with different range (ρ) values. The task of the systems is similar to that of the previous experiment, but with multiple objects that have to be found aside from the skin regions.

Table 6.1: The average speed of the classifiers in the first experiment.

Classifier	**Average training time required for one image (s)**	**Average testing time required for one image (s)**
2D LUT	*4.918*	*0.092*
3D LUT	*5.044*	*0.138*
$\text{FHM}_{\rho=5}$	*5.454*	*0.138*

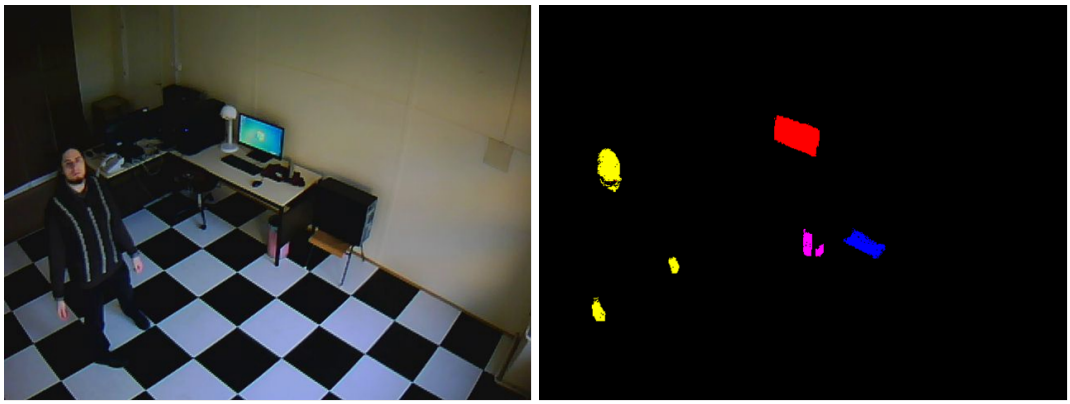


Figure 6.7: Training image for the second experiment (left) and result of the training for this particular training image (right).

Table 6.2: The average balanced accuracy, recall and precision of the classifiers in the second experiment, as well as the average training and evaluation times per image.

Classifier	**Recall**	**Precision**	**Balanced Accuracy**	**Training Time**	**Evaluation Time**
FHM$_{\rho=5}$	*47.10%*	*84.85%*	*73.52%*	*5.454 s*	*0.1535 s*
FHM$_{\rho=15}$	*42.68%*	*82.95%*	*71.3%*	*16.75 s*	*0.1543 s*
FHM$_{\rho=30}$	*41.06%*	*82.29%*	*70.5%*	*88.438 s*	*0.154 s*

Five different classes are used: the default class (colors of the unmarked pixels in the training image), the human skin regions (marked with yellow), as well as 3 different objects: the seat of the chair (blue), the trash can (magenta) and the monitor screen (with a constant image displayed on it, marked with red). Fig. 6.7 shows a training image and the output of the classification system after the training session, which is similar for all three FHMs.

As expected, the training speed of the classifiers are very much dependent on the range parameter. Table 6.2 shows the average speed results. In case of $\rho = 5$ the training time is 5.454 seconds on average per image, 16.75 seconds for $\rho = 15$, and 88.438 seconds for $\rho = 30$. Thus, as expected, the training time scales with the range parameter, although in a non-linear fashion. However, the average evaluation times are ranging from 0.152 to 0.155 seconds on average per image for all FHMs, so even though the training time is significantly longer for FHM$_{\rho=30}$, the evaluation time is still the same for all three classifiers.

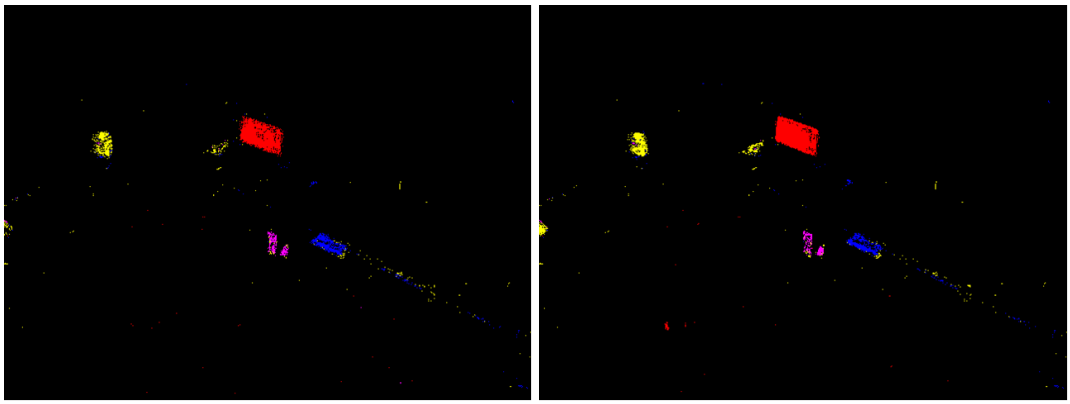


Figure 6.8: Classification accuracy of FHM$_\rho = 5$ (left) and FHM$_\rho = 15$ (right) for a test image.

Furthermore, the table shows ($\theta = 50$) statistical performance measures regarding the classifiers. Since the problem consists of imbalanced data (99% of which are negative values), *balanced accuracy* measure is used instead of classification accuracy, that gives a better estimation of the performance of the classifiers. Furthermore, the average recall and precision rates have been calculated as well. Interestingly, FHM$_{\rho=5}$ achieved the best results from all metrics: although its recall is just below 50% (meaning that it can detect about half (47.1%) of the positive (i.e. the sought) pixels), its precision is very high (84.85%, which means that the ones it marks as positive have a very high chance to be true positives). Opposite from what would be expected, the results get worse with higher ρ range values (though not drastically).

Fig. 6.8 shows a comparison between the performance of FHM$_{\rho=5}$ and FHM$_{\rho=15}$. The latter classifier proves to be slower, but manages to cover slightly more areas correctly. Considering accuracy, FHM$_{\rho=15}$ and FHM$_{\rho=30}$ prove to be more or less similar, in spite of the larger range (and thus, the longer training time) of FHM$_{\rho=30}$.

6.3.2 Parallel FHM Experiments

In this subsection, a comparison between the sequential and parallel version of the FHM and FFN classifiers are given based on experiments conducted on PC-16 (that has CUDA compute ability 5.2). The implementations have been done in Microsoft Visual Studio 2015, using CUDA 9.0 and OpenCV 3.3.0.

6.3.2.1 Training Speed

First, the training and evaluation speeds of the sequential and parallel FHM classifiers is analyzed. The images have been taken from the series used in the previous experiment, of which some are highlighted: e.g. Fig. 6.9 depicts one of the training images (left), with the

manually marked classes in it (right). The subsequent images from the same series are used for testing. Similarly to the previous experiments, the goal is to mark the pixels with the desired color tones. In the resulting image the objects are to be located based on the density of the classified pixels.

Figure 6.9: A training image (left) and the corresponding mask that marks the different classes (right).

Table 6.3: The Steps of the Training Phase and the Time Requirements of the Sequential and Parallel Implementations (P=307200, P'=37218, ω=4124).

Operation	**Required time (ms)**		**Time Complexity**	
	Sequential	**Parallel**	**Sequential**	**Parallel**
Extracting data from the image to set up the list of centers	*118*	*1.07379*	$O(P)$	$O(1)$
Redundancy and inconsistency filtering	*31747*	*13086.49*	$\sim O(P^2)$	$O(P)$
Calculating radial information	*6900*	*762.233*	$O(P'^2)$	$O(P')$
Filtering out background and covered clusters	*17043*	*195.2792*	$O(P')$	$O(P')$
Calculating σ parameters	*0.1*	*0.01952*	$O(P')$	$O(1)$
Total	*55808.1*	*14045.09*	$O(P^2)$	$O(P)$

Table 6.3 compares the speed and time complexity of sequential FFNs to that of parallel FFNs (pFFNs). As it can be seen, the most time-consuming step for both filters is the inconsistency filtering, but while the former makes ~$\frac{P^2}{2}$ comparisons (P=640×480=307200) to incrementally build the shorter list, the latter uses a semi-parallel approach for the training (dividing the processing of the input data array into a sequence of multiple subarrays, and using parallel computing to process the elements of each subarray).

The filtered list (P') contains considerably less training samples (P'=37218), though it still takes the sequential algorithm 6.9 seconds on average to calculate the ranges of each clusters and 17 seconds to filter out the covered and background clusters. The parallel algorithm does these in 0.762 and 0.195 seconds (9 and 87 times faster), respectively. The required total time for the training is 55.8 seconds for the sequential implementation on average, while 14 seconds for the parallel one, achieving an almost fourfold (392%) overall speed increase. Although in theory a larger amount of acceleration is expected, but in practice there are limiting factors, such as the speed of moving data back and forth between the host and the device (CPU and GPU), the launching and maintaining of concurrent processes, etc.

The sequential implementation of the FHM filter takes ~1.2 seconds to train on average, while the parallel version requires ~3.089 seconds, of which the calculation of the fuzzy membership function values take 3.058 seconds. This is because $L \times (2\rho + 1)^3$ concurrent processes are used (where L is the number of marked values that serve as the centers of the 3D fuzzy sets that have a uniform discrete radius of ρ). To illustrate the magnitude of the amount of concurrent processes, let us consider one of the training images in which L=37084 and ρ=10. In this case the number of required concurrent processes is 343434924, which causes significant computational overhead during the training.

6.3.3 Evaluation Speed and Performance

The classifiers (trained using the inputs and mask in Fig.6.9) were tested on images with different sizes. On the images with the smallest resolution (640×480) the sequential FHM and the pFHMs achieve an operational speed of ~105.8 ms and ~2.04 ms, respectively (which means the parallel version achieves a speed increase more than 50 times over the sequential one); while the FFNs are much slower, taking 117.247 seconds and ~658 ms, respectively (where the parallel version achieved a >99% decrease in training time over the sequential one). The required time drastically increases for the (sequential) FFN on the images with size 960×720, while the other filters only get 25-119% increase. The even larger images have proved too much for the FFN implementations (in case of the parallel FFN, due to its semi-parallel implementation, the kernels simply take too much time and thus time out), but

the FHMs are resilient enough to process even 1080p images. Furthermore, the pFHM can process such images in a time as low as ~7 ms (achieving a frame rate of more than 142 FPS (frames per second)).

Table 6.4: The Evaluation Speed [ms] of the Filters on Images with Different Resolutions

Filter	**640×480**	**960×720**	**1280×960**	**1440×1080**
FHM	*105.8*	*230.4*	*420*	*518.8*
pFHM	*2.04*	*3.84*	*5.919162*	*7.085222*
FFN	*117247*	*458832*	*n/a*	*n/a*
pFFN	*658.17*	*1472.18*	*n/a*	*n/a*

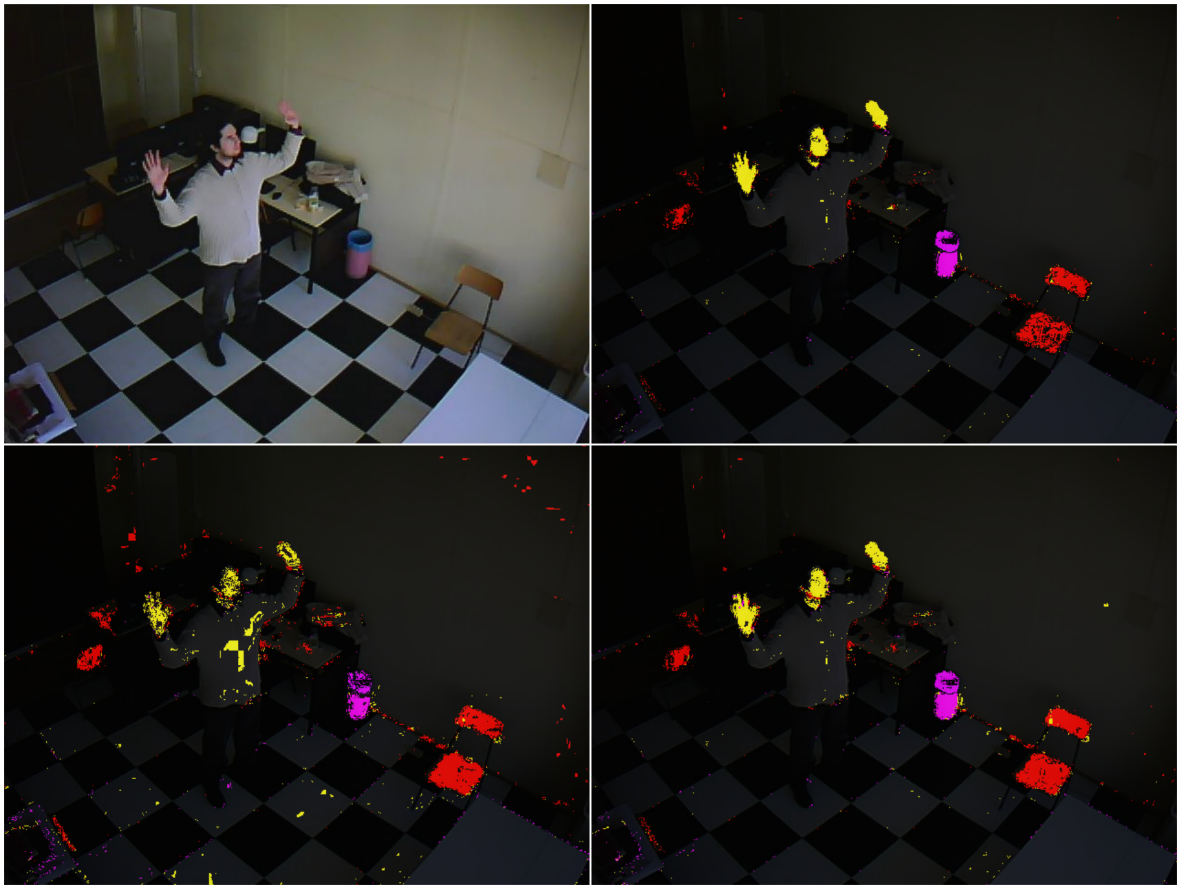


Figure 6.10: A testing image (top left) for a 3-class problem (color filtering) and the results given by the sequential FHM classifier (top right), the pFFN$^{HSV}_{\rho=15}$ filter (bottom left) and the pFHM filter (bottom right). The different color classes are denoted as follows: chairs are marked with red, garbage bin marked with magenta and human skin regions marked with yellow.

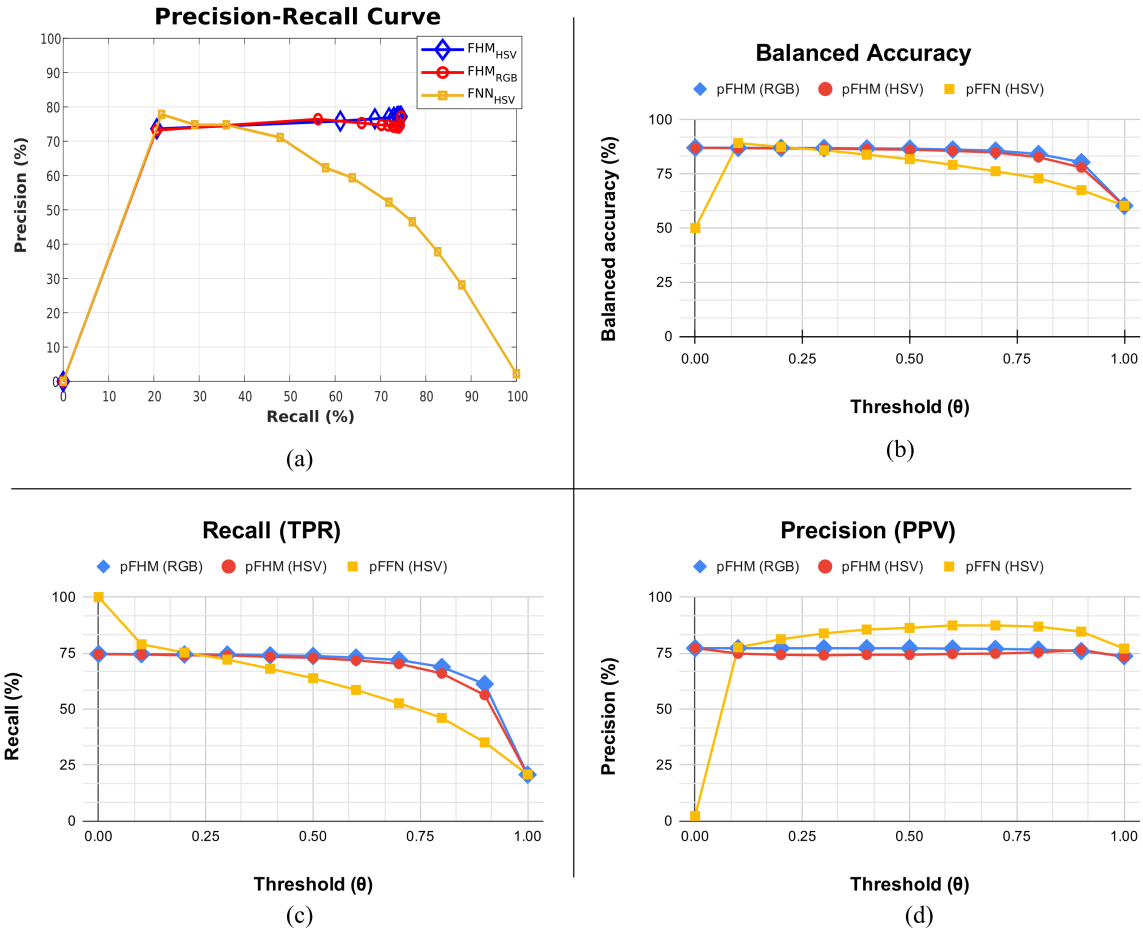


Figure 6.11: The performance of the $\text{pFFN}^{HSV}_{\rho=15}$ (orange), $\text{pFHM}^{RGB}_{\rho=10}$ (blue) $\text{pFHM}^{HSV}_{\rho=10}$ (red) classifiers: (a) Precision-Recall curve, (b) balanced accuracy, (c) recall (in function of θ) and (d) precision (in function of θ).

Finally, the classification performance of the pFFN and the pFHMs have been examined. For the latter, RGB and HSV color spaces have been regarded: i.e. which one yields the better classification performance. The pFFN has been trained with full clustering distance in the HSV color space ($\text{pFFN}^{HSV}_{\rho=15}$), while both the $\text{pFHM}^{RGB}_{\rho=10}$ and $\text{pFHM}^{HSV}_{\rho=10}$ classifiers have been trained with $\rho = 10$. In the results, correctly identified pixels (i.e. the returned class matches with the ground truth (which has been manually made)), count as *true positives*, while incorrect positive classification counts as *false positive*. *True negative* is counted as a background colored pixel classified as such, and *false negative* is the case when a sought color tone is deemed as a background tone.

Fig. 6.10 shows a testing image and the output of each classifier. Statistical metrics have been used to measure the performance, by evaluating each classifier multiple times and averaged for different values of θ threshold. The results are shown in Fig. 6.11:

- In (a) the precision-recall curve [106] can be seen, which shows that the $\text{pFFN}^{HSV}_{\rho=15}$

(orange) generally has an inverse proportional correlation between its recall and precision (i.e. it either gives many guesses about the positives of which only a low percentage is true, or gives only a few of which most are true). The two pFHMs on the other hand generally provide a high level of precision (~73-76%), for a wider range of the recall value. This implies a more stable classifier.

- In (b) the balanced accuracy can be followed in function of θ. The pFHMs generally show a good performance (~80-87%), while the pFFN$^{HSV}_{\rho=15}$ shows a slow decline as the threshold increases.

- In (c) and (d) the recall and precision rates are shown in function of θ. As it can be seen, the pFFN$^{HSV}_{\rho=15}$ marks all the actually positive pixels in the image, but in the process it marks *everything* in the image, achieving a near zero precision in the process. For higher threshold, the precision is increasing as the classifier is getting more and more cautious with marking pixels as positive (i.e. the recall rate is decreasing). The precision and recall rates of the pFHMs generally stay stable until the threshold exceeds $\theta = 90\%$.
  Remark: The PR curve has been generated in Matlab 2018b, while the other three figures in Google Spreadsheets (the latter cannot use multiple series in the horizontal axis, which is necessary for the PR curve).

Overall, the two pFHMs achieved a more stable operation, although they have not achieved a recall rate over ~75% (i.e. missed about 1/4 of the areas), but as it can be seen in Fig 6.10 (which shows the $\theta = 50\%$ case), they still mark the areas in high enough density so the sought objects can still be easily found with post-processing (e.g. noise filtering: because of their high precision, the misfires only appear as sparse noise). Remark: Interestingly, of the two FHMs, the one using the RGB color space (pFHM$^{RGB}_{\rho=10}$) has consistently performed slightly better than its HSV counterpart (while in case of RNNs, the opposite is the case). Investigating the reason for this the subject of further research.

6.4 Applicability

Although the lookup table-based method is proven to be much faster than rule-based approaches (e.g. FFN), this enhanced operational speed comes at the cost of increased memory usage as the whole problem space is required to be stored in the memory (as shown in the previous subsection). While the problem space is of manageable size for color filtering, for bigger problem domain sizes and higher dimensionality the memory

requirements can quickly surpass the capabilities of the available hardware the system is implemented on.

The advantage of using FFN-based filters is that they are easy to manually modify by adding, deleting, or changing existing rule parameters. Furthermore, it is much more compact in structure size, but it also requires a significant amount of computation, making it much slower. On the other hand, the LUT-based Fuzzy Hypermatrices can be implemented on the hardware level to process the output of the measuring instruments (like a camera, tristimulus colorimeter or a colorimetric spectrophotometer) automatically, due to its simple structure and functionality.

6.4.1 Spatial Complexity

Since the hypermatrices are needed to be coordinated by integer values, the classifier is most suitable for problems with discrete attribute values (or categories). One excellent example for that is color tone identification.

For a given problem, the applicability of the method can be determined easily. Let us denote the amount of memory necessary to store the structure with S_M. It can be calculated according to Eq. (6.15), assuming character type hypermatrices (thus, each cell takes 1 byte to store) [P. 25]:

$$S_M = (Q_\kappa + 1) \cdot \prod_{i=1}^{N} D_i \tag{6.15}$$

where D_i corresponds to the number of integer elements of the domain of attribute i, N stands for the number of attributes, and Q_κ denotes the quantity of classes that are not equivalent to the default class. As it can be seen, the amount of memory necessary to store the structure is most sensitive for the amount of attributes and the length of their respective domains.

Overall, the sequential FHM classifier has a spatial complexity of $O(N \cdot D_i^N)$, which makes it unfeasible for higher dimension problems (with 5 or more attributes), or problems with large problem domains.

On the other hand, color filtering has already been mentioned multiple times, because the proposed classifier is very well suited to this application area. The most commonly used color spaces (e.g. RGB, HSV) have only 3 attributes and their domain lengths are typically 256 for each dimension.

Applying Eq. (6.15), it gives that ~33.5 MB is enough to store a 2-class problem (where one of the classes is considered as default), and ~Q_κ·16.7 MB for Q_κ classes, making it easily manageable for modern computers.

The parallel version also suffers from the same downside in general (having the spatial complexity of $O(D_i^N)$), but choosing to aggregate the fuzzy arrays into a single one makes it more usable for parallel color filtering problems.

Remark: One advantage of the FHMs considering their spatial complexity, however, is that they are independent from the number of inputs. Thus, the size of the structure remains the same regardless of the amount of the input data it has been trained with. It is also worth noting that theoretically the classifier can be used in higher dimensions, if the domain of each dimension is very low (e.g. each attribute can only have a binary value). On the other hand, with the dimensions the complexity of the implementation is increasing.

6.4.2 Time Complexity

The time complexity of the training of sequential FHMs is $O(P \cdot \rho^N)$.

The time complexity of the evaluation on the other hand is drastically lower: since the algorithm only needs to regard a given number of array elements for each input sample, its time complexity is $O(P \cdot \kappa)$, for P inputs. If the fuzzy arrays are aggregated into a single one during the training, then this drops to $O(P)$. For the parallel version this is reduced even further, since all inputs are processed simultaneously (it is not $O(1)$ (i.e. not constant, so not completely independent on the amount of input data), however, as P still influences the amount of processing elements (kernels) that are needed to be launched, so there is always a low amount of dependence on P).

6.5 Summary of New Results

In this chapter, **I have proposed a new classification method, so-called Fuzzy Hypermatrices (FHMs)** that extends the classic lookup table classifier structure in order to achieve a fast and robust classifier. Ther base idea behind the classifier is the introduction of fuzzy logic into classic lookup table structures in order to gain generalization ability.

The method is shown to perform at high speed and high precision on low dimension problems with imbalanced data, such as color filtering, through multiple experiments where the performance of the classifier is analyzed for multiple (range) parameter values, comparing it to that of classic lookup table classifiers and Fuzzy Filter Networks [P. 9]. The proposed classifier consistently achieved a better classification performance than the other ones.

Furthermore, through the experiments, I have proven that the presumption of **introducing fuzzy logic can indeed add a certain level of generalization ability in a classifier that had none by default (like lookup tables) is true**.

I have also developed a parallel version of the FHM classifier (pFHM), which achieved a much (~50 times) faster online operation compared to the sequential version.

Through the conducted experiments, it is shown that the **pFHM classifier provide real-time color filtering with an balanced accuracy of ~80-87% and precision of ~73-76%** on the testing data, which is enough to make it possible to find the sought objects based on the density of the marked pixels.

The classifier overall provides a robust classification performance (maintaining roughly the same level of precision and recall for most threshold values), while providing a very fast operation. The limitation is that it is restricted to problems with a low number of dimensions (<5), since the spatial complexity is an exponential function of the dimension number.

Although computational complexity of the operation of the pFHM classifier is not completely independent on the number of inputs, it is still low enough to be advantageously used in real-time systems.

6.5.1 Thesis Group V.

6.5.1.1 Thesis Statement V.1.

I have introduced Fuzzy Hypermatrices, a new classification method that extends lookup table classifiers in order to achieve a fast and robust classifier. I have introduced generalization ability to the method by implementing Fuzzy logic implicitly into the structure. The method is shown to perform at high speed and high precision on low dimension problems, such as color filtering.

Related publications: [P. 4], [P. 23], [P. 24], [P. 25]

6.5.1.2 Thesis Statement V.2

I have developed a parallel version of the FHM classifier that is able to process images with a significantly lower time complexity than that of the sequential version, and provide a more stable classification performance than pFFNs.

Related publications: [P. 4], [P. 23], [P. 24], [P. 25] [P. 26]
Number of independent citations as of May 2021, according to Google Scholar: 1

Chapter 7

Real-time Classification with Sequential Fuzzy Indexing Tables

In one of my earlier works, I have introduced Fuzzy Hypermatrices (FHMs) [P. 4], which are Lookup Table-based classifiers for real-time pattern recognition of low dimension data. However, since most practical classification problems have a higher number of attributes (more than 4), the development of a new method was necessary: one that follows the LUT-principle (realizing a simple, direct association between input and output values), but is also more flexible in structure than FHMs and requires less space to store, while maintaining a high level of classification accuracy.

Another aspect of the problem is data attribute number dynamism. Most classifiers have been designed for classification problems with a *constant, fixed number* of input parameters. For cases with variable length data classification, the data is usually converted into a form with a constant number of attributes and processed with an appropriate traditional fixed-length input method, like artificial neural networks (ANNs, [46]).

Among the classifiers designed to be able to work with variable length data, *Recurrent Neural Networks* (RNNs, [107]) are the most widely used methods for classification, regression, and prediction of sequential data, such as text, speech, or time series. They are used in numerous fields of instrumentation and measurement, in fault detection (e.g. in [108] the RNN is applied to model sensor nodes, their dynamics, and interconnections between them in a wireless sensor network); defect propagation (e.g. [109] utilizes an RNN to process statistical parameters from vibration signals of a defect-seeded rolling-element bearing, for the prognosis of machine health status); power system control (e.g. in [110] RNN is used for high-speed phasor detection and adaptive identification of control and protection signals), etc. The main difference between ANNs and RNNs is that in the latter each neural unit has a recurrent connection from the previous time step, so it not only uses

data from the inputs but from the state of the neuron from the previous step as well.

Among the various training methods that have been proposed for RNNs, the so-called backpropagation through time (BPTT, [111]) method is one of the most typically applied, which realizes the temporal aspect of the neuron like a regular feed-forward network where the consecutive layers represent consecutive time steps (one layer for each time step). Then the network can be trained with the classic backpropagation algorithm. However, their training is made difficult by the so-called problem of vanishing/exploding gradients [112], which is generally a problem for neural networks with gradient based methods, but it is even more prominent in case of RNNs.

Long Short-Term Memory (LSTM, [113]) RNNs have been developed as a solution for this problem. They utilize memory cells with a unit recurrent connection between each time step. So-called gate layers are defined for each memory cell in order to control the inputs and outputs or serve as a reset option to flush the memory. These gates are responsible for regulating the flow of information and thus, preventing the gradients from vanishing, while the gradient exploding can be avoided by enforcing a hard constraint over the norm of the gradient.

In this chapter, I present the results of a research that aimed to develop a new kind of LUT-based classifier that can achieve two goals: a very fast operation with using a more compact structure than that of Fuzzy Hypermatrices, and reliable variable-length sample classification. For this end, I have developed a new pattern recognition method that is called *Sequential Fuzzy Indexing Tables* (SFIT). SFITs are multidimensional extensions of FHMs: instead of storing the whole problem space in a single array, they use a sequence of 1D and 2D arrays, composed into a layered architecture. This change makes it possible to implement *constant length* (c-SFIT) and *variable length* (v-SFIT) versions of the classifier as well.

The structure of this chapter is as follows. Section 7.1 describes the proposed classification method in details: the general architecture and the training and evaluation procedures for both approaches. Section 7.2 illustrates the operation through experiments and shows experimental results comparing the performance of the classifier to that of other classifiers. Finally, Section 7.3 investigates the applicability by analyzing the time (Subsection 7.3.1) and spatial complexity (Subsection 7.3.2) of the SFIT classifier, as well as presents a practical application (Subsection 7.3.3). Finally, Section 7.4 summarizes the new results.

7.1 Sequential Fuzzy Lookup Tables

The base idea of the SFIT classifier is looking at the problem space from the aspect of a road map with a single starting point. The road system (in the problem space) is created using the values of the observed data (X) of the problem. Each road (a given sample $x \in X$) leads through a series of junctions, where each outgoing street is labeled by a given value: the next path to be taken is decided using the attribute values of x. At the end of each road is a class value κ that belongs to x. An illustration can be seen in Fig. 7.1 showing the concept.

In practice, this means that the classification of sample x takes N steps for an N dimension problem, using an appropriate attribute value x_i to choose the next step, and by that, restricting the problem space by one dimension with each step.

The structure of the SFIT classifier is designed to model the important parts of such problem space (where the known class values are located, given the training samples) and to implement a method that makes restrictions on the problem space by combining the attribute values, in order to find the class that is assigned to them. It has a layered architecture, where each layer is associated with an attribute. The first layer serves as a starting point, assigning a unique ID to each unique value of its attribute (based on the training data). Let us call these unique IDs *index markers*, or simply *markers*. In the subsequent layers, the markers represent the combination of unique values of each attribute. Each layer adds one attribute value to the combination, restricting the problem space dimension by dimension.

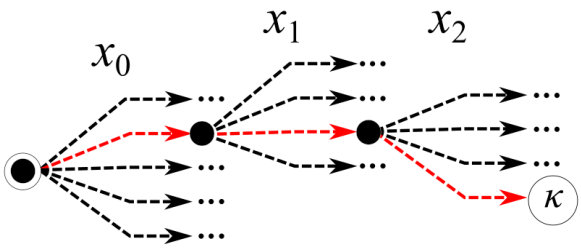


Figure 7.1: A road map illustration where the goal is to find the class κ that belongs to a given sample x. The search is done using the attribute values (x_i) to select the next path at each junction. The path sequence taken by x is marked by red, while the others are not continued (they are marked with three dots.)

7.1.1 Constant Length SFITs

7.1.1.1 The Architecture of Constant Length SFITs

The structure of the c-SFIT can be seen in Fig. 7.2. Just like the mentioned hypothetical road map, it consists of N layers (representing the set of junctions in each step). The arrays

follow each other sequentially. In each layer L_i, there can be four arrays: an index array Λ^{L_i}, a fuzzy array M^{L_i}, and statistical arrays G^{L_i} and υ^{L_i}. The first layer only has the former two arrays, because the data of the latter two can be calculated using data from layer previous to them (L_{i-1}).

Index array Λ^{L_i} that indicates which path the classification step needs to take in the next step, in the form of *index markers* (or *markers* for short). Let us denote the number of markers in layer L_i with γ^{L_i}. The markers are used in combination with the input values to address the index arrays (of the subsequent layers).

Remark: since arrays require positive integer numbers, all $\tilde{x_i}$ input values that do not satisfy this requirement are transformed into the positive integer values $\tilde{x_i}$ using (Eq.(6.3)).

The index array of the first layer (Λ^{L_0}) consists of a single row, and addressed with the first attribute value ($\tilde{x}_0$). The rest of the layers Λ^{L_i} ($\forall i \in [1, N-1]$ the corresponding attribute value $\tilde{x}_i$ and the index marker gained in the previous layer (let us denote it with η_{i-1} provide the inputs, to gain the marker $\Lambda^{L_i}_{x_i,\eta_{i-1}}$ that is used in the next step (for the next layer).

For each index array Λ^{L_i}, there is a fuzzy array M^{L_i} of equal size in the same layer. It assigns fuzzy membership function values to each marker, as well as to the areas around them in a ρ range (but only within the same row as the marker). The fuzzy value at each marker is 1 (i.e. 100%), while in its ρ area it decreases gradually. Thus, similarly to the case of FHMs, for SFITs the M^{L_i} arrays implement a certain level of generalization ability, with which the classifier can recognize samples that are very similar to the ones it has been trained with.

Furthermore, *additional statistical information* is stored in the form of matrices G^{L_i} and column-vectors υ^{L_i} for all non-root layers. The former keeps a count for each class that is associated to each marker in the previous layer (effectively, the classes distribution), while the latter notes the most regularly occurring class for each marker in the previous layer (i.e. the class with the highest number in the corresponding row of G^{L_i}).

If there are multiple classes that share the same occurrence number in the same row, the first such class is chosen. By using this statistical information, the system becomes more robust and achieves a certain level of anytime operation [114]: stopping the evaluation step at any of these layers, the most likely answer is always available.

Fig. 7.2 shows the architecture of the classifier. It depicts an example of a 3D, 3-class problem. The structure has been trained with the 4 samples that can be seen on the right hand side: [5 1 6 | α_2], [5 1 3 | α_0], [2 6 0 | α_2] and [1 3 1 | α_1], where the first 3 values (attribute values $\tilde{x}_0$, $\tilde{x}_1$ and $\tilde{x}_2$) are the attribute values and the assigned class label (κ) is denoted by α_i ($i \in [0, 2]$). Certain areas in the arrays are colored to better illustrate the changes that each sample caused in the structure during the training.

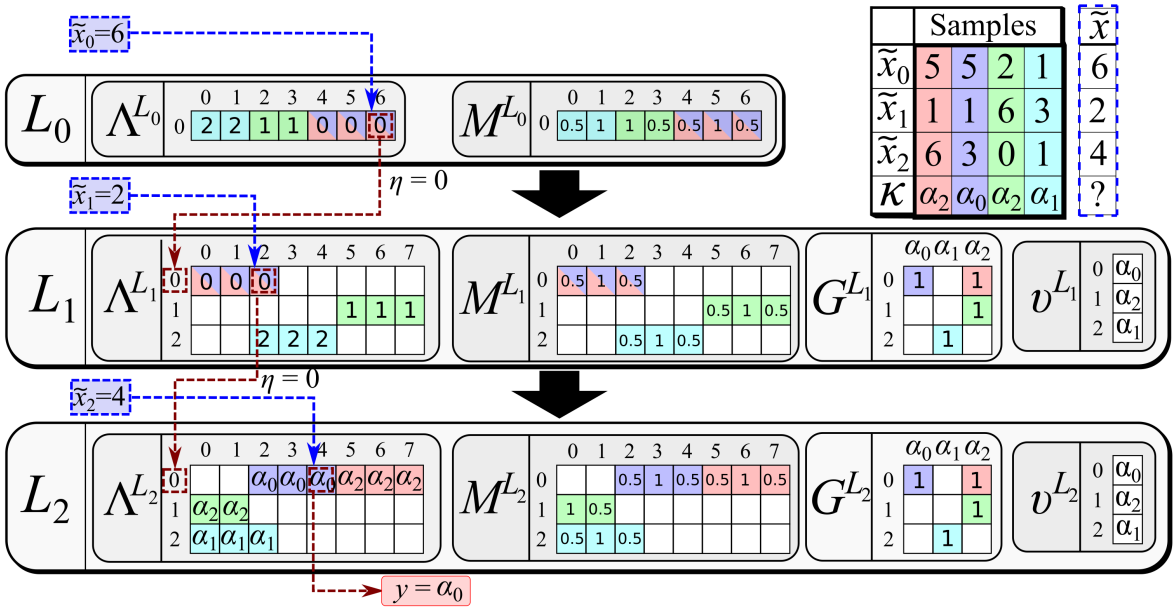


Figure 7.2: The structure of the c-SFIT classifier for a 3-dimensional, 3 class problem. The structure of the 3 layers is showcased: the index arrays (Λ^{L_i}), the fuzzy arrays (M^{L_i}) and the statistical arrays G^{L_i} and υ^{L_i}. The training samples can be seen on the upper right corner, while the evaluation of a sample is showcased in blue color.

The figure also illustrates the evaluation of the system for input [6 2 4 | ?]. The index value (marker) for $\tilde{x}_0 = 6$ is looked up in the index matrix of the first layer, gaining the row index of the second layer. Combining it with $\tilde{x}_1 = 2$ the index value for the last layer is gained. In the last layer, $\Lambda^{L_2}_{0,4}$ is checked, and class label α_0 is found, and subsequently returned as output.

Remark: if $\Lambda^{L_2}_{0,4}$ had no known value associated to it, the system would have given an estimated answer, due to the stored statistical information: $y = \upsilon^{L_2}_0 = \alpha_0$.

7.1.1.2 The Training Procedure of c-SFITs

The training of the c-SFIT classifier consists of 3 main phases. The first phase involves a quick analysis on the training data set in order to determine the necessary coefficients required to build the structure. Since the matrices in each layer need positive integer coordinates, the input data (x) values are mapped to a suitable domain, which can be done with a simple linear transform function, then rounding it to the closest integer number ($\tilde{x}_i$), using Eq.(6.3).

The size of the domain associated with each attribute is calculated (D_i for attribute i):

$$D_i \approx a_i \left(max\left(X_i\right) - b_i \right) \tag{7.1}$$

where X_i is the set of all values of attribute i of all available training samples (X), while a_i and b_i are linear coefficients associated with attribute i. They are determined by analyzing of the training data: $b_i = min(x_i)$ $(\forall x_i \in X_i)$, while a_i is set manually depending on the problem. The latter also provides a tool to tune the sensitivity of the system (as fine details in the data might be lost during the rounding step (e.g. the floating point values x_0=2.1 and x_1=2.3 are both rounded to the same integer value ($\tilde{x}$=2), but with a=10 they are treated as two different values ($\tilde{x}_0$=21 and $\tilde{x}_1$=23))), and also manipulate the size of the domains (D_i) directly, so larger domains can be manageable with limited memory storage. For integer valued problems this step can also be skipped.

With the domain sizes known, the arrays in each layer are initialized next. This can be done in two ways. One way is starting with 1D arrays with length D_i that are dynamically extended by adding new rows whenever it is necessary during the training. This makes the training time longer, but the arrays are of optimal size all the time. The other way is done by estimating the number of rows for each array, which can be done by analyzing the amount of unique values for each layer above; then the unused rows are simply deleted after training. This results in a faster training phase, but the estimation is needed to be more or less accurate.

In the second phase, the training samples are processed one by one. For each layer L_i ($i \in [1..N-1]$), it is checked if it i is the only attribute. If it is not the last unprocessed attribute, then the value of fuzzy matrix $M^{L_i}_{\eta,\tilde{x}_i}$ is checked, where η is the marker from the previous layer. For the first layer (L_0), the initial value of η is 0 since Λ^{L_0} and M^{L_0} are 1D arrays.

If the fuzzy value is 1, then this value already has a marker associated to it (which addresses a row in the next layer), so the corresponding index value can be determined:

$$\eta = \Lambda^{L_i}_{\eta,\tilde{x}_i} \tag{7.2}$$

which is used to note which row will be observed in the next layer.

If the value of the fuzzy array M^{L_i} is less than one, then there is no marker that is associated directly with the attribute value combinations thus far, so one is created by incrementing the number of records (γ^{L_i}) by one:

$$\gamma^{L_i} = \gamma^{L_i} + 1 \tag{7.3}$$

Then, the area around $(\eta, \tilde{x}_i)$ in Λ^{L_i} and M^{L_i} is updated in the following way:
for $\forall j \in [\tilde{x}_i - \rho, \tilde{x}_i + \rho]$:

$$\mu(j) = 1 - \frac{|\tilde{x}_i - j|}{\rho} \tag{7.4}$$

$$\Lambda_{\eta,j}^{L_i} = \begin{cases} \gamma^{L_i}, & if\ \mu(j) > M_{\eta,j}^{L_i} \\ \Lambda_{\eta,j}^{L_i}, & otherwise \end{cases} \tag{7.5}$$

$$M_{\eta,j}^{L_i} = \begin{cases} \theta, & if\ \mu(j) > M_{\eta,j}^{L_i} \\ M_{\eta,j}^{L_i}, & otherwise \end{cases} \tag{7.6}$$

where the fuzzy membership function value (i.e. the extent of how much a given value belongs to a given index marker in Λ^{L_i} ($i \in [0..N-1]$)) is calculated by $\mu(j)$ (in this case, a simple linear measure from the location of said index marker in the corresponding row of Λ^{L_i}). E.g. if $\tilde{x}_i = 5$, $\rho = 3$ and the position in question is $j = 4$, then $\mu(j) = 0.66$ (i.e., 66%).

The index and fuzzy matrices are only updated where the new fuzzy value is bigger than the old one, thus it does not interfere with previously absolved knowledge (i.e. fuzzy sets belonging to already existing markers). After that, the statistical matrix in the next layer ($G^{L_{i+1}}$) is updated and the index marker is set to the new value and the process goes on to the next attribute and the layer L_{i+1}.

$$G_{\eta,t}^{L_{i+1}} = G_{\eta,t}^{L_{i+1}} + 1 \tag{7.7}$$

$$\eta = \gamma^{L_i} \tag{7.8}$$

When the last attribute is reached, then the class ID belonging to the training sample is noted in index array $\Lambda^{L_{N-1}}$ with its fuzzy value in $M^{L_{N-1}}$, as well as the area around it (for $\forall j \in [\tilde{x}_i - \rho, \tilde{x}_i + \rho]$).

The last phase of the training is calculating statistical vectors υ^{L_i} for $i \in [1, N-1]$, for each row k:

$$\upsilon_k^{L_i} = \underset{\forall j}{\operatorname{argmax}}(G_{k,t}^{L_i}) \tag{7.9}$$

7.1.1.3 The Evaluation Procedure of c-SFITs

The evaluation step of the classifier is as follows: first, if the input data values are not in positive integer format, then they are transformed using the coefficients determined at the start of the training:

$$\tilde{x}_i \approx \max_{\forall i}(1, \min_{\forall i}(a_i x_i - b_i, D_i)) \tag{7.10}$$

for all attribute i, in order to ensure that the transformed values stay in their respective domains ($\tilde{x}_i \in [0, D_i]$).

For each attribute $i \in [0, N-1]$, similarly to the training, first it is checked if the currently examined attribute is the last one or not. If it is not the last one, then the fuzzy value of $M^{L_i}_{\eta,\tilde{x}_i}$ is checked (η=1 in the first layer). If the fuzzy value is 0, then the evaluation ends and returns with *unknown* class if $i = 1$, or outputs $\upsilon^{L_i}_{\eta}$ if $i \in [1, N-2]$. If it is larger than 0, then the corresponding new index value η is determined using Eq. (7.2), then the next layer is examined using the next attribute value ($i = i + 1$).

For the last attribute ($i = N - 1$) the value of fuzzy class matrix $M^{L_i}_{\eta,\tilde{x}_i}$ is checked. If its value is not 0, then the algorithm results in the class label stored in $\Lambda^{L_{N-1}}$:

$$y = \Lambda^{L_{N-1}}_{\eta,\tilde{x}_{N-1}} \tag{7.11}$$

If the fuzzy value is zero, then the classifier can still provide an answer due to the statistical data:

$$y = \upsilon^{L_i}_{\eta} \tag{7.12}$$

Fig. 7.2 illustrates the evaluation step with the path highlighted by dashed arrows, for input [6 2 4 | ?]. The index marker for $\tilde{x}_0$ = 6 is looked up in the index matrix of the first layer, gaining the row index $\eta = 0$ of the second layer. Combining it with $\tilde{x}_1$ = 2 the index value $\eta' = 0$ for the last layer is gained. In the last layer, $\Lambda^{L_2}_{\eta',\tilde{x}_2} = \Lambda^{L_2}_{0,4}$ is checked, and the system returns with $y = \alpha_0$.

7.1.2 Variable Length SFITs

In order to make the classifier capable of efficiently processing data with variable length samples, I have extended the c-SFIT architecture: a pair of new arrays are added to each layer in order to store the class labels ($\widehat{\Lambda}$) and the fuzzy membership function values ($\widehat{M}$) that belong to them.

7.1.2.1 The Architecture of v-SFITs

Thus, *Variable length SFITs* (v-SFITs) are an extension of c-SFITs. Aside from the attribute values and the class, the *length* of each sample (N_x) is explicitly taken into account. The number of layers is set to $N_{textmax}$ (i.e. the length of the longest sample in the input dataset X). The main difference between c-SFITs and v-SFITS is that since the route of evaluation (illustrated by the dashed arrays in Fig. 7.2) can stop before the last layer, the class labels are stored in a separate array $\widehat{\Lambda}^{L_i}$ instead in each layer L_i. Furthermore, to secure the same

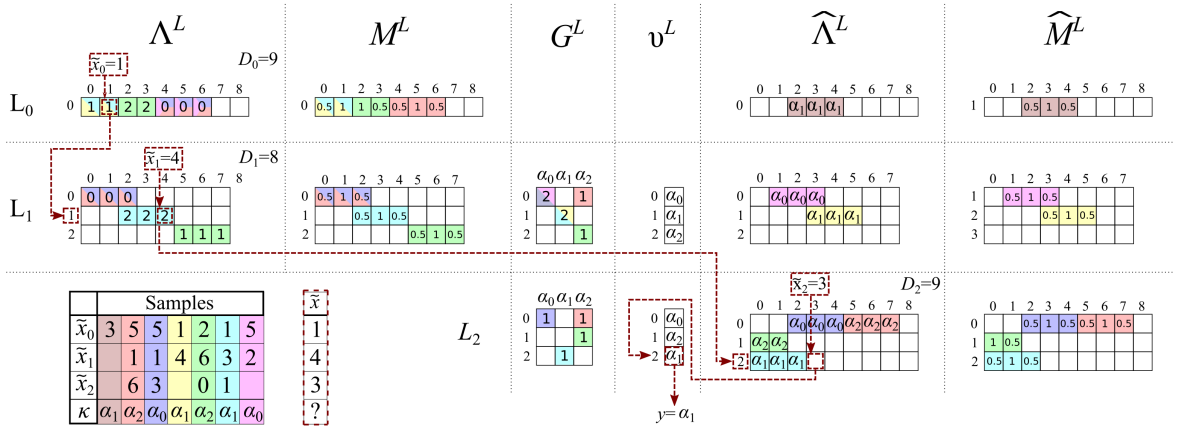


Figure 7.3: The structure of the v-SFIT classifier, for an up to 3 dimensional problem with 3 classes and variable length samples.

level of generalization ability of the system as that of the c-SFITs, an additional fuzzy array $\widehat{M}^{L_i}$ in each layer assigns fuzzy values to each non-zero element in $\widehat{\Lambda}^{L_i}$. The size of the two additional arrays are the same as that of the index array in the same layer.

Fig. 7.3 illustrates the structure and operation of v-SFITs for a 3-dimensional, 3 class problem (ρ=1). The dataset that can be seen in Fig. 7.2 is extended with 3 new data samples: [3 | α_1], [1 4 | α_1], and [5 2 | α_0]. As it can be seen by the placement of the colored areas in the arrays (still colored accordingly to the samples that first affected those array cells), the order of the training samples is significant and can change the content of the structure.

7.1.2.2 The Training of v-SFITs

The training of the v-SFIT classifier is summarized in Fig. 7.4. Most of the algorithm is the same as that of the constant length case: data analysis and the determination of the structure parameters, as well as the processing of the first attribute for each layer (steps marked with yellow color in the figure), then the processing of intermediary layers (marked with cyan).

The algorithm differs when the last attribute of the training sample ($i = N_x$-1) is reached (marked with brown color): the array elements in the area around $(\eta, \tilde{x}_i)$ in $\widehat{\Lambda}^{L_i}$ and $\widehat{M}^{L_i}$ is updated for $\forall j \in [\tilde{x}_i - \rho, \tilde{x}_i + \rho]$ using Eq. (7.4) and

$$\widehat{\Lambda}^{L_i}_{\eta,j} = \begin{cases} \gamma^{L_i}, & if\ \mu(j) > \widehat{M}^{L_i}_{\eta,j} \\ \widehat{\Lambda}^{L_i}_{\eta,j}, & otherwise \end{cases} \tag{7.13}$$

$$\widehat{M}^{L_i}_{\eta,j} = \begin{cases} \theta, & if\ \mu(j) > \widehat{M}^{L_i}_{\eta,j} \\ \widehat{M}^{L_i}_{\eta,j}, & otherwise \end{cases} \tag{7.14}$$

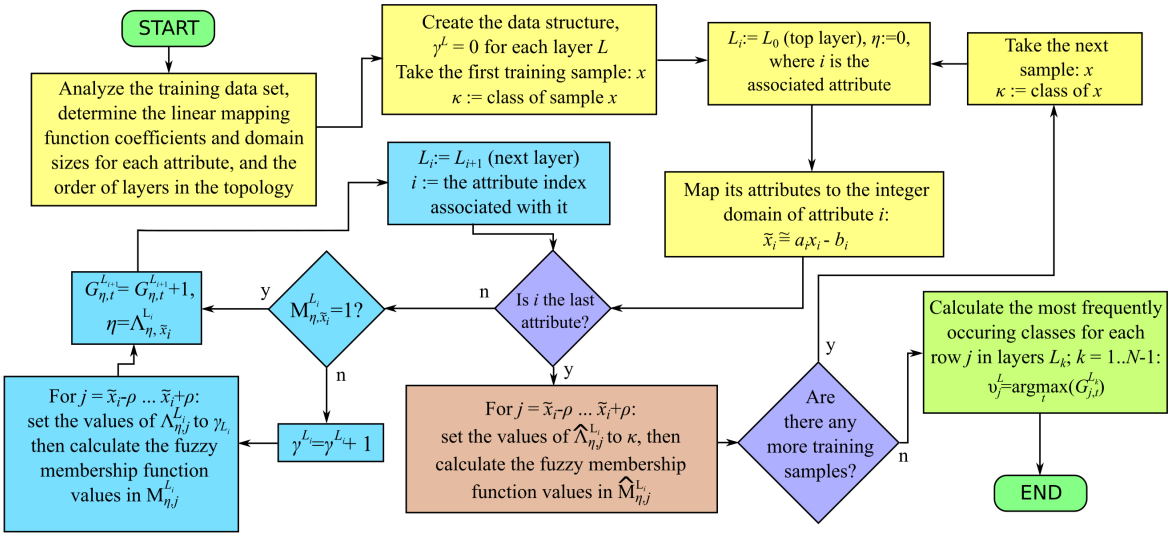


Figure 7.4: The training algorithm of the v-SFIT classifier.

The last phase of the training is, just like in the constant length case, calculating statistical vectors $\upsilon_k^{L_i}$ for $i \in [1, N-1]$, for each row k using Eq. (7.9).

7.1.2.3 The Evaluation of v-SFITs

The evaluation step of the v-SFIT classifier for attributes $i \in [0, N-2]$ is the same as that of the c-SFITs: first the data is transformed into an appropriate domain (positive integers, e.g. with Eq. (7.10)), then for each attribute i ($i \in [1, N_x]$), it is checked if the currently examined attribute is the last one or not considering the length (N_x) of the sample. If attribute i is not the last one, then the fuzzy value of $M^{L_i}_{\eta,\tilde{x}_i}$ is checked ($\eta = 0$ in the first layer). If the fuzzy value $M^{L_i}_{\eta,\tilde{x}_i}$ is 0, then the evaluation ends and returns with *unknown* class in the first layer ($i = 0$), or $\upsilon^{L_i}_{\eta}$ if $i \in [1, N-2]$.
If the fuzzy value is more than 0, then the corresponding new index value η is determined using Eq. (7.2), then the next layer is examined using the next attribute value ($i+1$).

While processing the last attribute ($i = N_X - 1$, if the evaluation have not ended in a previous layer), the value of fuzzy class matrix $\widehat{M}^{L_i}_{\eta,\tilde{x}_i}$ is checked. If its value is not 0, then the algorithm gives the class label stored in $\widehat{\Lambda}^{L_i}$:

$$y = \widehat{\Lambda}^{L_i}_{\eta,\tilde{x}_i} \tag{7.15}$$

If the fuzzy value is zero or if the input data has more attributes than the size of the structure ($N_X > N$), then the classifier can still provide an answer due to the statistical data:

$$y = \upsilon^{L_i}_{\eta} \tag{7.16}$$

Fig. 7.3 also illustrates the steps of the evaluation for input sample [1 4 3 | ?]. The index value (marker) for $\tilde{x}_0$ = 1 is looked up in the index matrix $\Lambda_{0,1}^{L_0}$ of the root layer, gaining the row index of the second layer (L_1). Combining it with $\tilde{x}_1$ = 4, the index value for the last layer (L_2) is gained. In the last layer, $\widehat{\Lambda}_{2,3}^{L_2}$ is checked, and since it has no known value associated to it, the system returns with the estimated answer: $\upsilon_2^{L_2} = \alpha_1$.

7.2 Experimental Results

In order to test the performance (classification accuracy and speed) of the SFIT classifier, the system has been tested on two benchmark data sets. The experiments have been conducted in Matlab™ R2013a (on PC-8). Furthermore, in order to provide a comparison between the performance of the proposed classifier and other methods in literature, three fuzzy classifiers are also investigated. The program codes can be found in the File Exchange of Matlab Central. The conducted experiments have been done in Matlab™ R2012a (on PC-16).

7.2.1 Constant Length SFIT Experimental Results

The first classifier is the Fuzzy K-Nearest Neighbor (FKNN, [115]) algorithm (Matlab code by [116]). It is based on the K-Nearest Neighbor method that uses clustering to select k samples of the training data set, and during the evaluation phase the input data gets classified as the class of the trained sample closest to it. The fuzzy improvement of the method assigns a membership function (in function of the distance from the given trained samples) instead of a crisp label, thus making the process more robust.

The second classifier (Matlab code by [117]) is a similarity-based classifier as well [33]. The classification is done by comparing the inputs to so-called ideal vectors and the assigned class is the class of the vector to which the input is the most similar (i.e. has the highest similarity value). Lukasiewicz structures [118] with fuzzy values are used in order to get similarity values. To aggregate the gained vector into a similarity value Ordered Weighted Averaging (OWA) operators are applied. The weight parameters are calculated with a quantifier guided aggregation. Four different quantifiers are used: basic, quadratic, exponential and trigonometric linguistic quantifiers.

The third classifier [34] (code made by [119]) is a Neuro-Fuzzy (NF, [120]) approach. It proposes a fast scaled conjugate gradient (SCG, [121]) method for the training of NF classifiers, by estimating the first order gradient of the parameters with a least square estimator (LSE, [122]) instead of calculating it from the training data. This is claimed to reduce the training time of the NF classifier by 20-50%, making large-scale problems more computable with personal computers.

Remark: for the three classifiers the implementations are only available with the training and testing phases merged in a single function, therefore the average processing time of the two phases are measured as a single operation.

7.2.1.1 The Wisconsin Breast Cancer Dataset

The first c-SFIT experiment uses the *Wisconsin Breast Cancer* (WBC, [123]) data set. It consists of 683 data samples, each sample having 9 attributes and can be associated with two possible classes. The classifiers are investigated for various parameter values and (complementary) training to testing set ratios (the samples are selected randomly in each training phase). The F-KNN is investigated for various cluster numbers (1, 3, 5, 10, 15, 20, 25) and its classification accuracy and required times is averaged. The OWA-based similarity classifier is investigated for its 4 different quantifiers, and the best performing one is noted with respect of the considered input data.

Table 7.1: Average Training and Testing Speed and Classification Accuracy for the Wisconsin Breast Cancer Dataset

Training to testing set ratio	**Classifier**	**Average classification accuracy (%)**	**Training time (s)**	**Testing time (s)**
70 : 30%	F-KNN	*96.68*	*0.23061*	
	OWA, exponential	*96.09*	*0.0983*	
	Neuro-Fuzzy	*95.18*	*1.0933*	
	c-SFIT (δ=0.5)	*94.04*	*0.1237*	*0.0253*
80 : 20%	F-KNN	*96.28*	*0.1701*	
	OWA, exponential	*96.05*	*1.2920*	
	Neuro-Fuzzy	*96.10*	*1.0776*	
	c-SFIT (δ=0.4)	*93.72*	*0.1313*	*0.0165*
90 : 10%	F-KNN	*97.22*	*0.0965*	
	OWA, exponential	*96.76*	*0.3558*	
	Neuro-Fuzzy	*96.27*	*1.2086*	
	c-SFIT (δ=0.4)	*94.70*	*0.1566*	*0.0083*

The Neuro-Fuzzy classifier is tested for 100 epochs (during the experiments, using more than that generally did not result in a better classification rate, the system could converge to the highest possible performance considering the training data within 100 epochs).

The results of the first experiment are summarized in Table 7.1. The F-KNN classifier provides the best performance generally in terms of accuracy (~96-97%), while the other three perform only slightly worse (~94-96%). The Neuro-Fuzzy classifier is generally the slowest, while the F-KNN and c-SFIT are the fastest (with the exception of the OWA classifier with exponential quantifier in the 70 to 30% case.)

7.2.1.2 The Seismic Bumps Dataset

The second c-SFIT experiment [124] is a real-time case study. The available data is about mining operations in Polish coal mines in the Upper Silesian coal basin. The goal of the classification is seismic hazard assessment, based on the data of various seismic sensors.

Table 7.2: Average Training and Testing Speed and Classification Accuracy for the Seismic Bumps Dataset

Training to testing set ratio	**Classifier**	**Average classification accuracy (%)**	**Training time (s)**	**Testing time (s)**
70 : 30%	F-KNN	*91.57*	*0.3337*	
	OWA, exponential	*93.41*	*4.0339*	
	Neuro-Fuzzy	*92.73*	*3.1275*	
	c-SFIT (δ=0.001)	*91.38*	*0.942381*	*0.1355*
80 : 20%	F-KNN	*91.53*	*0.2455*	
	OWA, exponential	*94.54*	*3.0183*	
	Neuro-Fuzzy	*92.97*	*2.9943*	
	c-SFIT (δ=0.0005)	*91.21*	*0.7967*	*0.0917*
90 : 10%	F-KNN	*91.38*	*0.1402*	
	OWA, exponential	*93.41*	*2.0032*	
	Neuro-Fuzzy	*93.40*	*3.7734*	
	c-SFIT (δ=0.0005)	*91.70*	*0.8963*	*0.0455*

The data set consists of 2584 multivariate data with 15 attributes (although 18 attributes are listed, of which the last 3 are constant zeros, therefore they can be neglected) and 2 classes: *'hazardous'* and *'non-hazardous'* situation. With an appropriate and fast classifier, such situations can be detected in time and withdraw the workers from the danger zone. The way of training is the same as in the previous experiment.

The results can be seen in Table 7.2. In general, the OWA classifier with quadratic quantifier provides the best accuracy (~94%), followed by the Neuro-Fuzzy (~93%) and the F-KNN and c-SFIT classifiers (~91%). However, both of the two most accurate classifiers take ~2-4 seconds to operate in general, while the F-KNN and the c-SFIT tale about or less than 1 second to learn the training data and evaluate the testing data. Furthermore, it can be seen that the c-SFIT classifier requires a relatively large fuzzy set width (δ) in the previous experiment due to the relatively small attribute domains, in this experiment some domains (sensory variables) are very large thus a very small width parameter is more advantageous.

7.2.2 Variable Length SFIT Experimental Results

To evaluate the performance of the proposed v-SFIT classifier, it has been tested on two problems with varying length data. The experiments have been conducted in Matlab™ R2012a, on PC-16.

7.2.2.1 Classification of English words

In the first v-SFIT experiment, the data consists of common English words (verbs, nouns and adjectives) and the task is to classify them based on their part of speech. Since many words fall into two or three categories at the same time, the combined classes count as additional classes and thus, it is considered a 7 class problem. The distribution of the data samples can be seen in Table 7.3. There are overall 6075 words in the dataset and the longest word is 17 letters long. The ASCII code of each letter is used as input data, thus the problem can be regarded as a general variable length pattern recognition problem. For example, the word "act" corresponds to (97 99 116 | 3), considering it being both a verb and a noun (thus belonging to class 3).

In order to provide a comparison between the performance of the proposed classifier, a speed-optimized LSTM RNN is investigated as well. It has been proposed and implemented by [125], in which a matrix-based batch learning paradigm with backpropagation through time is proposed in order to accelerate the training of the LSTM RNN. It makes vectored calculation for each layer thus, speeding up the training procedure. It applies padding with zeros to fill in empty spaces due to varying amount of samples. It has been demonstrated

in the chapter that this method provides a faster and more accurate convergence over the traditional stochastic gradient descent (SGD) methods.

Five different testing sets are generated from the training set of words. The first three are emulating the effect of noise, randomly adding or subtracting a certain amount (δ) from the value of the characters. To make each testing sample, the generator function takes the base set (the training data), selects a random sample then adds a random value z ($z \in [-\delta, \delta]$) to its attribute values. The fourth set (shortened, see Table 7.4) is made by deleting the last character of randomly selected samples (making them shorter), while the last set (elongated) is produced by adding a random character to random samples (also selected from the training samples). Both classifiers are tested with 5000-5000 samples (randomly generated the way described above) for each kind of testing sets and the results are averaged. Since the training data are not independent from each other (one word can be the extension of another), Type II v-SFIT is used.

As described in Section 7.1, the training of the classifier begins with the analysis of the input data set, or more precisely, the values of each attribute. In this case, the input data is integer valued, so the transformation (from x to $\tilde{x}$) can be skipped. The domain (D) of each attribute is 256 (as the ASCII code of each letter is considered), which determines the column dimension of the arrays in each layer. The row number comes from estimating the number of markers (γ) in each layer. These numbers can also be estimated by the quantity of training samples (in this case, 6075) if it is not overly high (too big arrays can cause memory deficiency), then after the training the unused records can be cropped from the arrays. The optimized LSTM network takes ~60 seconds on average to learn the problem (6075 training samples, using 2 memory cells per block, two gate layers, and batches with size of 405).

On the other hand, the v-SFIT processes the training samples in ~1.1 seconds.

Table 7.3: The Distribution of Training Data in the First Problem.

Classes	**Training Sample Quantity**	**Class ID**
Verb	*585*	*1*
Noun	*3947*	*2*
Verb and noun	*238*	*3*
Adjective	*1157*	*4*
Adjective and verb	*22*	*5*
Adjective and noun	*120*	*6*
Adjective, noun, and verb	*6*	*7*

The output of the LSTM network is rounded to the closest integer. The accuracy of both classifiers is measured in the ratio between the amount of correctly classified samples and the total amount of samples.

Table 7.4 shows the performances of the two classifiers. The LSTM network performs with 68-69% accuracy on average, while the v-SFIT performs ~9-13% better on the shortened, elongated, and lightly perturbed data set. Both perform with about the same accuracy on the moderately perturbed data set, while the LSTM works better on the noisiest data set. This is due to that in the problem the markers (the unique values, in this case the letters) in each layer tend to be very close to each other (well within the size of the width of the fuzzy sets (ρ), thus leaving little or no room for the generalization ability of the system) considering their values. Otherwise, the v-SFIT can correctly classify any new words that are similar to known ones (the distance between their letters is not larger than ρ).

Due to the matrix-based batch testing, the LSTM performs faster (~0.032 seconds) than the online, sample-by-sample method that v-SFIT applies (~0.34 - 0.378 seconds). However, the latter also has the advantage that the further training of the network can be done quickly, because it simply adds to the already learned data without the need to retrain the structure as in the case of neural networks (as long as the new data values are still within the original domains).

Table 7.4: Comparison Between the Performance of v-SFIT and LSTM RNN Classifiers in the First Problem

Classifier	**Testing data**				
	Added noise			***Shortened***	***Elongated***
	$\boldsymbol{\delta = 1}$	$\boldsymbol{\delta = 2}$	$\boldsymbol{\delta = 3}$		
Optimized LSTM	69%	69%	68%	69%	68%
v-SFIT	82%	70%	63%	79%	80%

7.2.2.2 Classification of Shape Descriptors

In the second v-SFIT experiment, an image processing problem is considered. 36 shape images (Fig. 7.5) have been selected from the MPEG-7 image database [126], then processed by a shape extractor system [P. 43] that extracted and analyzed the interesting points (e.g. corners) of the shapes. The interesting points break the shape down to segments, then each segment is classified considering their length and the angles between them, after that each segment is classified (ranging from 0 to 149) by a simple fuzzy classifier. The output of the algorithm is a sequence of these integer numbers, each sequence being unique to a given shape (while being rotation and scale invariant). The resulting dataset consists of 36 samples with lengths varying between 5 and 21. For example, the extractor algorithm has found 5 interesting points on the triangle shape, labeling them [119 31 148 101 31]. (For more information about the extractor system, please see [P. 43].) The samples have been used to train an Artificial Neural Network (ANN [46]) and the v-SFIT classifier. The number of neurons used for the ANN is 100, and the average training time is 1.76 seconds, the classifier converges in 4 iterations. The v-SFIT takes 0.03 seconds to train.

The testing data has been made with the same method as in the previous experiment. 1000 samples have been generated for each testing cycle. The average testing times are 0.0075 seconds and 0.117 seconds for the ANN and v-SFIT, respectively. Table 7.5 shows the accuracy of the classifiers. ANN performs with a 72.2-75.3% accuracy on the noisy data, while the v-SFIT achieves a 97.7-98.74% average accuracy. The ANN has performed significantly worse (~ 25%) on the elongated testing set and could not classify the shortened testing set at all. On the other hand, due to using additional statistical data, the v-SFIT achieved a 100% recognition rate on both. This clearly shows the strength of the classifier when it comes to variable length pattern recognition even from a low amount of training samples, which is generally a problem for neural networks.

Remark: this experiment was done using a regular RNN and a *Non-Linear Autoregressive Neural Network* with external inputs (NARX [127]) as well (both classifiers are part of the Machine Learning toolbox of Matlab 2012b), but neither networks managed to provide an accuracy better than 38% in the conducted experiments.

Figure 7.5: The shapes used in the second problem, from the MPEG-7 data set. The grouped images belong to the same class.

Table 7.5: Comparison Between the Performance of v-SFIT and Artificial Neural Networks (ANNs)

Testing data	Accuracy	
	ANN	**v-SFIT**
Added noise, δ=1	*75.3%*	*98.74%*
Added noise, δ=2	*73.7%*	*97.88%*
Added noise, δ=3	*72.2%*	*97.9%*
Shortened	*0%*	*100%*
Elongated	*25.5%*	*100%*

7.3 Applicability of the SFIT classifier

7.3.1 Time Complexity of the SFIT classifier

The training procedure of the SFIT classifier consists of allocating the arrays, then filling the structure. The latter involves the modification of $2\rho + 1$ array elements for each attribute and all samples, which adds up to a time complexity of

$$O(\rho \cdot N \cdot P) \tag{7.17}$$

This shows that the training speed of the classifier is linearly dependent on both the number of attributes and the amount of input samples. The evaluation is even faster, as only one array cell is needed to be accessed (in the index/class array, as well as the fuzzy arrays):

$$O(N \cdot P) \tag{7.18}$$

This is true for both c-SFITs and v-SFITs (although the allocation of the structure in the memory takes twice as much time for v-SFIT due to the additional arrays, but that step is generally much faster than the filling of the structure).

To put it into perspective, the training and the evaluation of the FKNNs (one of the classifiers used in Section 7.2.1 and have proven to have about the same classification accuracy as the SFIT, albeit its training is faster because it takes advantage of the batch learning method (learning over groups of patterns)) has a time complexity of $O(P^2 \cdot \log(k))$ and $O(P \cdot \log(k))$, respectively (according to [128]), where P is the number of samples, and k is the clustering variable (although the same paper have proposed a faster algorithm that reduces them to $O(P^2)$ and $O(P)$, respectively).

7.3.2 Spatial Complexity of the SFIT classifier

Another important aspect to note is the size of the structure.

Considering the WBC problem, in order to keep the whole problem workspace in the memory, one would need to store a 9D Fuzzy Hypermatrix (10^9 elements), while the c-SFIT architecture only requires to store 58345 elements in the form of 24 2D (the index, fuzzy and statistical arrays G^{L_i} for $i \in [0, N-1]$) and 8 1D arrays ($\upsilon_k^{L_i}, \forall i \in [1, N-1]$). The formula to calculate the memory requirements ($S_{\text{c-SFIT}}$, in bytes) of the appropriate FHM (Eq. (6.15)) and the proposed c-SFIT classifier structure:

$$S_{\text{c-SFIT}} = (S_\Lambda + S_M)(D_0 + \sum_{j=1}^{N-1} \gamma_{j-1} D_j) + S_G(\sum_{j=1}^{N-1} \gamma_{j-1} \cdot Q_\kappa + \gamma_j) \tag{7.19}$$

where Q_κ is the number of classes in the problem, N is the amount of attributes, D_i is the domain size of attribute i, D_i is the domain size of the attribute associated with layer L_i, and γ_j is the number of rows in layer L_j. Finally, S_Λ, S_M and S_G are the bit sizes of the applied variable types for the index, fuzzy and statistical arrays, respectively. This depends on the size of the largest attribute domain in the given problem (e.g. if all $D_i \in [0, 65535]$, then they can be stored in an *unsigned short* type array (each array cell taking 2 bytes), while the type *unsigned int* takes 4 bytes for each array cell but can store a range of values up to $4.29 \cdot 10^9$. The size of v-SFITs is roughly twice of the size of c-SFITs, since they maintain additional class and fuzzy arrays:

$$S_{\text{v-SFIT}} = 2 \cdot (S_\Lambda + S_M)(D_0 + \sum_{j=1}^{N-1} \gamma_{j-1} D_j) + S_G(\sum_{j=1}^{N-1} \gamma_{j-1} \cdot Q_\kappa + \gamma_j) \tag{7.20}$$

While the column dimension for each array can be calculated (D_i for attribute i), the row dimension depends on the number of markers in the previous layer (γ_{i-1}), which is not easy to estimate without counting all occurring value-combinations. However, an upper limit can be estimated: γ will never be higher than the number of input samples (P). The spatial complexity therefore for both SFIT architectures:

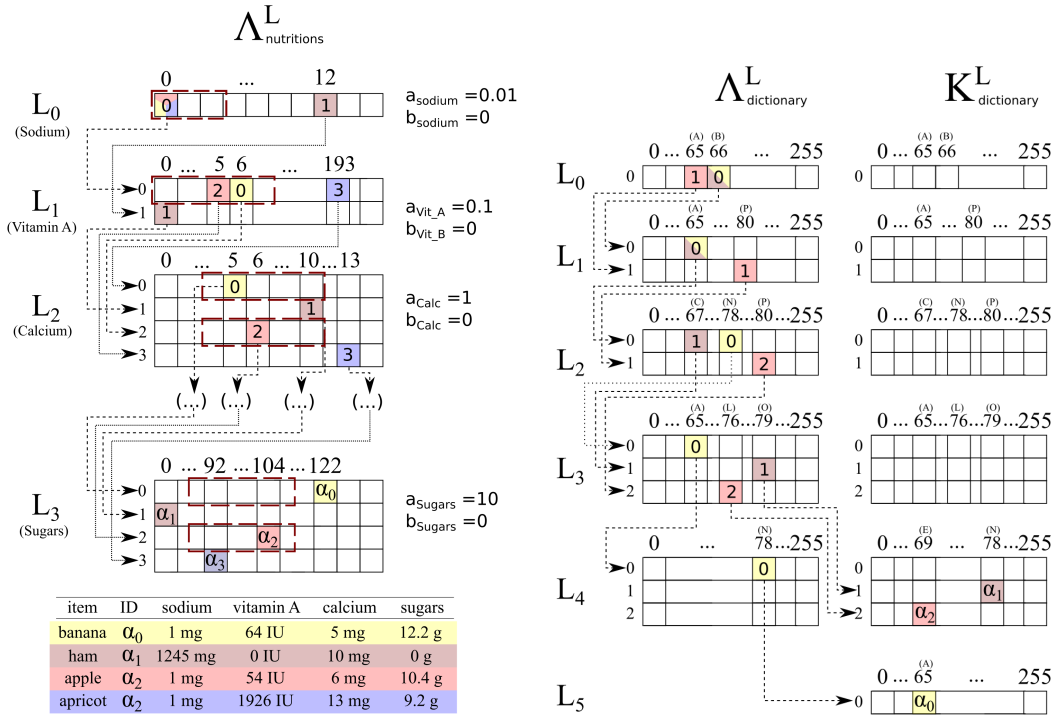
$$O(N \cdot P \cdot D_{max} + N \cdot P \cdot K) \tag{7.21}$$

This leads to a significant reduction in the size of the structure compared to FHMs. For example, regarding the experiment in Subsection 7.2.1.1 (on the WBC dataset), using Eq. (6.15) and Eq. (7.19), we can see that the amount of memory to store the structure of the FHM (S_T=1, D_i=10, N=9, K=2) is ~3 GB, while for the proposed classifier S_M=1, S_Λ=2, S_G=2, N=9, K=2, D_j=10 and γ_j=~222 for all $j \in [0, N-1]$) it is ~64 KB. It clearly

shows that the classifier succeeds in that goal to surpass the capabilities of the system it was based upon.

7.3.3 A Practical Application

A practical application of the SFIT classifier has been used in a dietary assistant application in [P. 29] and [P. 30], where the goal is to implement a system that can help its user achieve a healthier diet. The base idea is taking the current eating habits (the food and drink items) of the user, calculate its nutrition values, compare them to the recommended daily intake (RDI) then find the item that either contributes the lowest values, or causes at least one value to exceed the RDI values. For this, the system calculates a fitness function over the stored data to find the lowest scoring item. Then calculates the new current nutrition values by taking the nutrition values of the lowest scoring item out, then uses the deficit from the RDI to apply an interval search on the database of the items.



item	ID	sodium	vitamin A	calcium	sugars
banana	α_0	1 mg	64 IU	5 mg	12.2 g
ham	α_1	1245 mg	0 IU	10 mg	0 g
apple	α_2	1 mg	54 IU	6 mg	10.4 g
apricot	α_2	1 mg	1926 IU	13 mg	9.2 g

Figure 7.6: The SFIT classifier as an indexed data storage: c-SFIT to store nutrition values (left) and v-SFIT to act as a dictionary (right). In the former, crisp data is stored (no need for generalization) so there is no need for fuzzy matrices. In the latter, fuzzy matrices are used but they have been emitted from this figure for better visibility. Remark: IU stands for International Units.

This item database is realized by a c-SFIT and stores 31 nutrition data ($N_{nutrition} = 31$) for each of the 8790 food items ($P_{items} = 8790$) from the *SR18 database* [129] (Fig. 7.6(left)), and takes up ~990 MB of memory.

Furthermore, a v-SFIT is used as a dictionary (Fig. 7.6(right)) holding the 8790 words with a maximum length of 60 ($N_{dictionary} = 60$) and taking ~176 MB. The size difference is due to the sizes of the domains: the domain of the dictionaries only encompasses the ASCII codes of the letters from 65 ('A') to 122 ('z'), which is generally much smaller than that of the nutrition data.

After the system has found the item that falls into the intervals gained from the deficit of the current nutrition values and the RDI, it recommends that item to the user as a substitute to the worst performing item. This way, the user can optimize their diet bit by bit, in their own pace.

7.4 Summary of New Results

In this chapter, I have presented **a new kind of LUT-based classifier that can achieve two goals: a very fast operation with using a more compact structure than that of Fuzzy Hypermatrices, and reliable variable-length sample classification.** For this end, I have developed a new pattern recognition method that is called *Sequential Fuzzy Indexing Tables* (SFIT). SFITs are multidimensional extensions of FHMs: instead of storing the whole problem space in a single array, they use a sequence of 1D and 2D arrays, composed into a layered architecture. This change makes it possible to implement **constant length SFITs** (c-SFIT) and **variable length SFITs** (v-SFIT).

SFITs in general have a layered structure, where each layer contains a lookup table that combines the value of an attribute belonging to that layer to the combination gained in the previous layers. Thus in each layer, the proposed method reduces the size of the problem space in which the class of the input pattern is. **This significantly reduces the arrays that are needed to be stored in the memory, compared to Fuzzy Hypermatrices,** another classifier that uses the Lookup Table principle to achieve a very fast operation but at the cost of an exponential spatial complexity.

The main difference between c-SFIT and v-SFIT is that the latter has additional arrays in order to store the class labels separately. Although this change effectively doubles the spatial complexity of the structure, but it makes the SFIT able to process samples with different length. Furthermore, **the dependence on the dimensionality of the problem is still only linear** (as opposed to that of the FHMs, which has an exponential spatial complexity).

Although the time complexity of the method is higher than that of the FHMs (N data

access is needed instead of just one), the structure itself is typically only a fraction of the size of FHMs. Experiments have shown that the SFIT method is still faster than many contemporary classifiers, at the cost of a slightly worse (by ~1-3%) classification accuracy. On the other hand, SFITs have been proven to be more robust against noise for variable length data than other examined methods (LSTM RNN and NARX ANNs).

The new classifier has also been used in a practical application of a dietary assistant system, where the goal is to help the user optimize their diet by changing the least nutritious items to more nutritious ones one step at a time. The c-SFIT is used to store food data in a way so it can be quickly searched based on nutrition values, while v-SFIT is used to implement a fast dictionary that is used to identify the food items themselves.

7.4.1 Thesis Group VI.

I have developed a new form of Lookup Table classifier for multi-dimensional classification problems, called Sequential Fuzzy Indexing Tables that uses a sequence of one 1D and *N* 2D fuzzy lookup tables in to achieve a more memory-efficient classification than FHMs on problems with a constant number of attributes. It has a layered structure, where each layer contains a lookup table that combines the value of an attribute belonging to that layer to the combination gained in the previous layers. Thus in each layer, the proposed method reduces the size of the problem space in which the class of the input pattern is. This significantly reduces the arrays that are needed to be stored in the memory, compared to FHMs.

7.4.1.1 Thesis Statement VI.1.

I have expanded upon the idea of Fuzzy Hypermatrices and developed a new classifier named Sequential Fuzzy Indexing Tables for real-time classification of complex problems with a constant number of dimensions (c-SFIT). The problem space is stored in a sequence of 1D and 2D arrays, each adding one more attribute value to the combination and thus restricting the problem space. Tests have confirmed that the method is generally faster than most other state of the art pattern recognition approaches, while the classification is only slightly less precise.
Related publications: [P. 27], [P. 28], [P. 10]

7.4.1.2 Thesis Statement VI.2.

I have extended the Sequential Fuzzy Indexing Tables classifier to work on variable length data (v-SFIT). This is done at the cost of doubling the structure, in order to hold the class values in different arrays. The proposed methods have also been successfully applied in an iSpace-based dietary assistant application that uses a c-SFIT for storing food items based on their nutrition data and a v-SFIT for storing the names of food items.

Related publications: [P. 11], [P. 29], [P. 30]
Number of independent citations as of May 2021, according to Google Scholar: 3

Chapter 8

Conclusions

With the advancements made in machine intelligence in the last two decades, the feasibility of smart environments have also increased to the point where implementing smart areas, may it be an open space (like a park) or a closed area (a room or a whole building), is not only possible but also getting more and more affordable. The Intelligent Space (iSpace) is one such smart environment where the emphasis is helping people in their every day life, extending the sensory capabilities of robots, etc.

In this thesis work, I have presented my research regarding methods and applications that aim to enhance the capabilities of iSpace.

First, **I have proposed the concept and described the implementation of a hand posture and gesture modeling and recognition system** that can be a foundation of a complex, natural command recognition system for the iSpace: an interface that allows human users to control Intelligent Space by hand gestures. The presented system has been shown to be able to classify 6 different simple hand postures and any hand gestures that consist of any combination of the predefined hand postures.

However, the training of the CFNNs used in the hand recognition system has proved to be slow even with the lowered complexity compared to basic neural networks. To solve this, **I have developed a novel training method for feed-forward neural network models used in classification problems**. The algorithm applies a time-reduced preclustering procedure. The idea behind the preclustering is that the quantity of the input data should be reduced to keep the complexity of the procedure in a handleable rate in such a way in which the training ability is preserved as much as possible. The usability of the proposed technique is illustrated in classification problems of different toughness. This method has also been useful for the determination of the structure (number of hidden neurons and parameters) of Radial Basis Function Neural Networks. In case of RBFNNs, the main role of the presented clustering step is not to reduce the required training time, since one of the advantages of using RBFNNs

is the fast training by default. By applying the quantity and cluster centers of the resulting clusters, the quasi-optimum number of hidden layer neurons can be determined and their center parameters can be set, respectively. **After that, I have presented an iSpace control system framework that is capable of interpreting and executing detected commands given by a human user, if the commands are given in a specific form.** The system is able to learn repetitive commands that are given under similar conditions with the application of heuristics and execute them when the conditions are fulfilled without requiring the user to give these commands again. During the simulations with the virtual implementation of the system, it has been able to learn all the instructions and prohibition commands given by the virtual inhabitant of the virtual room. The knowledge of the system is designed to be quasi independent of natural languages, given that the knowledge representation uses concepts as nodes which are identified by dictionaries, so the represented knowledge does not depend on natural languages. The number of languages in which the system can receive commands can by increased by simply adding more dictionaries and text preprocessing units to it. The proposed framework has a modular architecture and is managed apart from the sensors and executive agents. The advantage of using this flexibly designable knowledge base-based control procedure is that sensors, detectors, and executive agents can be added and removed anytime without reconfiguring the system; only a new entry or modification in the knowledge base is needed.

Furthermore, **I have proposed a new proximity-based classification method, the so-called Fuzzy Filter Network** that can be used for color-based skin region detection in vision-based iSpace applications. The approach is based on the classic RBFNN architecture, with some modifications in the output layer that change the behavior of the neural network into a fuzzy rule-based inference network. The resulting system can efficiently be used to filter numeric data into categories. The training of the network is solved by a clustering step; the operational rules of the network are derived from the resulting clusters. The improved filter network is theoretically capable of the classification for any number of classes. **Two new clustering algorithms, designed specifically for the filter network, are also presented**. The two methods approach the problem in different manners, the first processes the problems as the iterative cases, while the other follows a bottom-up attitude. In both cases the trained network is shown to be able to achieve high classification accuracy (~80-97%) on balanced datasets. The performance of the system is illustrated through image processing problems. **Moreover, I have developed and implementation a parallel version of the FFN**. The speed of both the training and evaluation phase are significantly increased compared to the sequential implementation, as the complexity is reduced from $O(N \times P^2)$ to $O(N \times P)$ during the training (with P

training data and N attributes), and from $O(P \times \omega)$ to $O(P)$ (considering ω clusters) during the evaluation phase.

However, even with the acceleration the FFN classifier is too slow for real-time color filtering. In order to solve this problem, I have proposed a new classifier technique called Fuzzy Hypermatrices. The idea is based on the principle of the lookup table method, which achieves a fast run-time operation by doing most of the computation during the offline training phase and storing the results in arrays so that they become readily available during the online evaluation phase. However, in contrast to the classical look up table methods of classification that use probabilities to calculate and store likelihood values, the presented method applies fuzzy logic to handle uncertainty. The fuzzy approach also adds a certain degree of generalization ability to the system. The precalculated fuzzy membership function values are stored in multi-dimensional arrays (so called hypermatrices), implicitly modeling the problem domain. It works best considering problems with integer values and low attribute quantity, like color filtering tasks in image processing problems. A simple training algorithm is also proposed and its performance is shown through experiments. The main advantages, and thus the importance of the new classification technique are its simplicity, easy implementation, and high speed compared to other classifiers. Latter is probably the most significant, as other (non-lookup table based) classifiers generally use multiple steps in order to calculate the output, while for FHMs the amount of necessary steps is bounded by only the number of classes in the problem. This is very important for time-sensitive problems where the classifier is operated in a loop, like real-time image processing. Similarly to the FFNs, I have also proposed a parallel version of FHMs that enhanced the operational speed from ~1.9 frames per second to ~142.8 frames per second on 1080p images. Through the conducted experiments it is shown that the parallel Fuzzy Hypermatrices provide real-time color filtering with an accuracy (rate of successfully distinguishing important color tones from background tones) of ~98% and precision of ~82% (rate of correctly identifying the class of the detected important color ton), which is enough to make it possible to find the sought objects based on the density of the marked pixels.

The disadvantage of FHMs, however, lies in the memory usage. Problems with many (N>4) dimensions can require a possibly unmanageable amount of memory space to work. As a solution for this problem, **I have proposed a new classification method called Sequential Fuzzy Indexing Tables.** SFITS are Lookup Table-based fuzzy data structures that make the classification of constant and variable length data quick and reliable, at the cost of memory usage. They use a sequence of 1D and 2D arrays (structured into layers) to map the known areas of the problem space based on the training data. Although the

complexity of the method is higher than that of the FHMs (N data access is needed instead of just one), the structure itself is typically only a fraction of the size of FHMs. Experiments have shown that the SFIT method is still faster than many contemporary classifiers, at the cost of a slightly worse (by ~1-3%) classification accuracy. On the other hand, SFITs have been proven to be more robust against noise for variable length data than other examined methods (LSTM RNN and NARX ANNs).

The applicability of the Theses have been confirmed by experiments and time complexity analysis.

Chapter 9

Possible Targets of Future Research

There are numerous ways to extend or improve the presented methods. In this chapter, I briefly summarize some of these directions.

In future work, I plan to extend the set of **FHPMs** from the examined six models with more models so a vocabulary can be built, for example, for sign languages. With an adequate sign language recognizer application, it would be possible for people with hearing or speech difficulties to be able to communicate with the iSpace. Another practical example is binding various hand gestures to commands, e.g. in a hospital setting, the nurse making the rounds during the night shift can just gesture with her hand to have the system automatically turn off the lights or open doors in her way, without the need to touch any surfaces, or make a classic control gesture (e.g. clapping) that would be noisy and bother the patients. The hand-posture identification method will be improved as well since the actual implementation of the system does not take into account the position of the hand, only the shape of it. Furthermore, more modern classification methods will be investigated, such as the application of SFITs. The hand gesture detection part of the system will also be improved to take posture orientation and relative position into account as well.

Another direction is investigating the feasibility of using varying fuzzy set sizes and shapes: the current fuzzy sets are triangular and have a common width in each model. However, using different set widths can potentially increase the classification robustness (e.g. model m_i). One easy way to achieve this is defining multiple variations ($m_{i,0}, m_{i,1}$ etc.) of the same model m_i that differ only in small changes in some of their fuzzy sets, and then make an average of their sets, incorporating them into a single 'general' model $\widehat{m}_i$.
Furthermore, a disadvantage of the rule-based classifier (the *Fuzzy Inference Machine*) used to find the closest FHMP to the detected one is that each rule is its sequential operation. In future work, I will explore the possibility of using pFNNs (introduced in Chapter 5) in order to increase the speed of the FIM on the 14-dimension problem.

The **clustering algorithm** proposed in Chapter 3 will also be improved. One possible direction is the development of a new algorithm for determining of the "optimal" clustering distance for given problems. In optimistic cases (where the overlapping of classes is not significant), the clustering distance can be e.g. the distance of the two samples that are the farthest from each other among all the samples of the class. If the classes overlap than further considerations are needed. An idea can be the application of multiple clustering distances, using small distance where the overlapping occurs and larger where it does not. In future work I will investigate the effectiveness of these ideas, as well as practical applications for the clustering method in iSpace applications, such as passive space exploration through activity monitoring (e.g. automatically discovering places with given functionalities in the home environments, hospitals, public parks, etc., by simply observing that the users have spent longer periods of time in such places). Traffic control is another important field in which a good estimation about places with higher load, along with the rough measure of the load can help alleviate the problems caused by high traffic.

There are numerous ways the presented **iSpace framework** can be extended. Some of further possible improvement directions can be:

- **More flexible command parsing.** In the current implementation the command given by the user needs to be in a predefined form, thus the user is limited in giving commands. This limitation needs to be decreased with the application of more intelligent parser algorithms.

- **Causality.** The system should realize the indirect desires of the human user. E.g., if the user states that he or she is cold, then the system should realize that the correct course of action is to close the windows if they are open, turn on the heating, etc. To achieve this, the improvement of the Knowledge Base and Interpreter Module is necessary. Interaction between the iSpace and the human user: if the system is unsure about giving out a learned command (the justification value of the trigger of the hypothesis is just at the threshold) or there are two very similar hypothesis triggered at the same time (e.g. `"make coffee!"` and `"make tea!"`), the system should ask the human user about what action should it take.

- **Improved dictionaries and interpretation.** Since one word or expression can have multiple meanings, one entry in the dictionary can reference to multiple nodes, as well. In such cases, the context of the word or expression can help in deciding the correct meaning in the interpretation phase.

- **Distinction between users.** A more sophisticated system shall be able to differentiate/recognize its users and apply their personalized knowledge bases with

their own built hypothesis storages or assign roles to them (like guest, administrator, etc.), with certain privileges. This way the executable commands that the users can give to the iSpace could be regulated. The current version of the system does not differentiate among the human users, for the identification of each user more complex detection methods have to be built in.

- **Further scalability.** The presented framework is meant for a single system, but it can be easily extended with optional modules where different iSCS applications can connect, e.g. in a smart hospital, every room and hallway can have its own micro-iSCS, while the whole hospital can make up a large, robust system that is a network of micro-iSCSs.

Another aspect of expanding the application of the iSCS framework is helping the work of robotic agents in the iSpace, e.g. coordinating the routes that cleaning or transportation robots take in a facility based on the layout of the rooms and the capabilities of each robot (e.g. can traverse stairs, etc.).

For the three classifiers that I have proposed in Chapters 6, 5 and 7, the most promising future development directions involve parallel computing, either parallelizing still sequential parts of their algorithms or improving upon the already parallelized ones in order to achieve a higher training and run-time speed and a better classification accuracy. The parallel evaluation algorithm for **FFNs** can be further improved with newer parallelization techniques, as the current implementation still has parts that are done in a sequential manner, although the main obstacle in the way of mass parallelization lies in the limitation of the GPU: the number of processes that can be used at once is finite, and for high number of processes the maintenance of these processes causes a significant overhead.

The parallelized **FHM** has proven to be very fast run-time, but the training is still done using the sequential algorithm. In future work, I plan to create a more efficient, parallel training algorithm for it, as well as investigate the automatic tuning of parameters such as the range (ρ) of the fuzzy sets. Furthermore, I plan to further develop it in order to create an automatically adapting color-tone identifier for color-based object detection in the iSpace.

For **SFITs**, I plan to investigate the optimal setting of the arbitrary parameters of the system, conduct more experiments in order to further examine the performance compared to other classifiers, and attempt to improve the classifier to make it applicable for more complex problems (with more attributes), as well as further optimization options will be investigated regarding both the speed, accuracy and error estimation of the described algorithms. Furthermore, since the current structure for higher dimensional problems is generally too big for the graphics memory to handle, the structure itself will be improved upon by compressing it (as in the current form it usually have a significant amount of

unused cells, so ideally the structure could be reduced to only the markers and their immediate surroundings).

Chapter 10

New Results

10.1 Thesis Group I.

10.1.1 Thesis Statement I.1.

I have proposed a new fuzzy hand posture model that uses a fuzzy feature set consisting of 14 features over 3 feature groups. I have also developed a new hand posture detection method for the identification of hand postures. The input of the method is provided by an image processing system that uses two cameras and a Fuzzy Matching Algorithm to calculate the 3D coordinates of the shape of the hand. The 3D coordinates are processed by a set of Circular Fuzzy Neural Networks that map the coordinates into a set of 14 triangular fuzzy sets. The acquired fuzzy sets are processed by a fuzzy inference machine that provides the output of the method: the label of the hand posture model that is the most similar to the detected one. The performance of the method has been shown for 6 different hand postures.

10.1.1.1 Thesis Statement I.2.

I have developed a new neural network structure named Circular Fuzzy Neural Network. The structure of the new network is derived from interval-based Fuzzy Neural Networks by trimming the number of connections between the input and hidden layer neurons in order to achieve a faster yet more localized processing (one hidden layer neuron only processes the output of a given number of neighboring input layer neurons). The other significant difference comes from the circular shape of the new network, as the first neuron in each layer is regarded as a neighbor of the last neuron in the same layer. The new classifier is used to convert a set of 3D coordinates into 14 fuzzy features. It has been validated for 6 different hand postures.

10.1.2 Thesis Statement II.

In order to enhance the training speed of Artificial Neural Networks, I have introduced a new supervised Neural Network training method including an input sample clustering step. The proposed method generates clusters of similar input data samples and returns the centers of the clusters. The training is done by using the corresponding cluster centers instead of the input samples. The method has been shown to decrease training time significantly, at the cost of a slight decrease in classification precision. The new method has been validated on Artificial Neural Networks, Circular Fuzzy Neural Networks and Radial Basis Function Networks.

10.1.3 Thesis Statement III.

I have developed a new iSpace Control System (iSCS) framework consisting of intelligent detection modules, command processor modules, and autonomous action planner modules, together constituting a smart environment system that can provide services to users based on the available resources and agents. I have also designed and implemented a new graph-based knowledge representation system to describe a priori knowledge for control systems. The knowledge representation uses a graph-based structure that provides a flexible and easy to use architecture. The new knowledge representation system also enables automatic observation-based learning, in which hypotheses are used to model the observed a posteriori knowledge (gained by analyzing the details of commands issued by users), which are used for the automatized operation. The iSCS satisfies the following requirements: modularity, scalability, ease of integration, can be built from low cost components and easy to configure and maintain, mainly due to the wide availability of cheap Single Board Computers and sensors that can be used to implement the iSCS framework. The functionality of the system has been validated in a simulated environment.

10.1.4 Thesis Group IV.

10.1.4.1 Thesis Statement IV.1.

I have proposed a new fuzzy rule-based classification method called Fuzzy Filter Networks that is based on a topological modification of Radial Basis Function neural networks. Its training is done through clustering, where the clusters that cover the problem domain based on the training data are calculated. Said clusters are used as rules during the operation of the classifier, during which each hidden layer neuron calculates a Gaussian

fuzzy membership function value based on each rule. The output of the classification is the class label associated with the rule that is the closest to the input values. **I have developed a new clustering algorithm called Analytical Radial Representative clustering in order to train the new fuzzy rule-based classifier, which is based on a bottom-up approach.** The classification performance of the filter network using the proposed clustering algorithm as training has been evaluated through a color filtering experiment.

10.1.4.2 Thesis Statement IV.2.

I have designed a parallel realization of the new Fuzzy Filter Network classifier, which reduces the complexity of both the training and the operation of the classifier. The efficacy of the method is shown through a color filtering experiment.

10.2 Thesis Group V.

10.2.1 Thesis Statement V.1.

I have introduced Fuzzy Hypermatrices (FHMs), a new classification method that extends lookup table classifiers in order to achieve a fast and robust classifier. I have introduced generalization ability to the method by implementing Fuzzy logic implicitly into the structure. The method is shown to perform at high speed and high precision on low dimension problems, such as color filtering.

10.2.2 Thesis Statement V.2.

I have developed a parallel version of the FHM classifier that is able to process images with a significantly lower time complexity than that of the sequential version, and provide a more stable classification performance than pFFNs.

10.3 Thesis Group VI.

I have developed a new form of Lookup Table classifier for multi-dimensional classification problems, called Sequential Fuzzy Indexing Tables that uses a sequence of one 1D and N 2D fuzzy lookup tables in to achieve a more memory-efficient classification than FHMs on problems with a constant number of attributes. It has a layered structure, where each layer contains a lookup table that combines the value of an attribute belonging to that layer to the combination gained in the previous layers. Thus in

each layer, the proposed method reduces the size of the problem space in which the class of the input pattern is. This significantly reduces the arrays that are needed to be stored in the memory, compared to FHMs.

10.3.1 Thesis Statement VI.1.

I have expanded upon the idea of Fuzzy Hypermatrices and developed a new classifier named Sequential Fuzzy Indexing Tables for real-time classification of complex problems with a constant number of dimensions (c-SFIT). The problem space is stored in a sequence of 1D and 2D arrays, each adding one more attribute value to the combination and thus restricting the problem space. Tests have confirmed that the method is generally faster than most other state of the art pattern recognition approaches, while the classification is only slightly less precise.

10.3.2 Thesis Statement VI.2.

I have extended the Sequential Fuzzy Indexing Tables classifier to work on variable length data as well (v-SFIT). This is done at the cost of doubling the structure, in order to hold the class values in different arrays. The proposed methods have also been successfully applied in an iSpace-based dietary assistant application that uses a c-SFIT for storing food items based on their nutrition data and a v-SFIT for storing the names of food items.

List of Figures

List of Tables

List of Symbols

General Symbols

P	the cardinality of the input dataset
X	the input data set
X_i	a specific sample from the input data set X: the i^{th} sample in the set
x	a sample from the input data set X
x_i	the i^{th} value of sample x
$\tilde{x}$	non-negative integer value of sample x
$\tilde{x}_i$	the i^{th} value of sample $\tilde{x}$
y	the output of a given classifier
N	the dimension of a given problem, i.e. the number of data attributes
i	a given attribute
$d(u, v)$	the distance value between vectors u and v
κ_x	a classes label of sample x
Q_κ	the number of classes
$O(X)$	the *Big O notation* that describes how the space or time requirements of an algorithm grow in function of the given argument

Thesis I. Specific Symbols

$backproj(x, y)$	the back-projection function on an image
$H()$	hue histogram
$I(x, y)$	intensity value in a given image at coordinates x and y
μ_i	fuzzy membership function
a, b, c, d	the distance between the fingers (expressed in linguistic variables)
A, B, C, D, E	the distance between the fingers (expressed in linguistic variables)
$\alpha, \beta, \gamma, \delta, \epsilon$	the distance between the fingers (expressed in linguistic variables)
S	linguistic variable for SMALL
M	linguistic variable for MEDIUM
L	linguistic variable for LARGE
S_L	linguistic variable used for setting the width of the triangular fuzzy sets
d	identification value of an FHPM model
$center$	the center of a given fuzzy set
$width$	the width of a given fuzzy set
T	1D array, the numerical representation of an FHPM
d_H	the Hamming distance
n	a fixed number of attributes
I	number of neurons in the input layer of a neural network
J	number of neurons in the hidden layer of a neural network
K	number of neurons in the output layer of a neural network
ϱ	the number of epochs a neural network takes to learn
m	the number of hand posture models

Thesis II. Specific Symbols

y	the output of a given supervised learning model
d	the desired output of a given supervised learning model
c	criteria used to examine the difference between y and d
u	the input of a supervised learning model
u'	the clustered centers created from u
k	the number of clusters
A_i, B_i, C_i	three neural networks trained with different clustering distances
δ	clustering distance
S_L	linguistic variable used for setting the width of the triangular fuzzy sets
G_i	feature group
C	number of neurons
ϕ	matrix used to store the calculated distances between each neuron and input sample in the given RBF implementation
D	the array of the desired outputs (targets)
g_0	bias value
$g_i()$	Gaussian activation function of neuron i
c_i	the center parameter belonging to neuron i
σ_i	the width parameter belonging to neuron i
w_i	weight parameter belonging to the output of neuron i

Thesis III. Specific Symbols

A, B, C	given concepts
N_A	a node belonging to concept A
CV	current value
mV	minimum value
MV	maximum value
SEN	sensitivity value
$V_{A,CV}$	the current value of concept A
$V_{A,mV}$	the minimum value of concept A
$V_{A,MV}$	the maximum value of concept A
$V_{A,SEN}$	the sensitivity value of concept A
U	a so-called environmental variable, a concept node that has current, minimum, maximum and sensitivity values defined (in the form of $V_{U,CV}$, etc. meta edges)
u	the current value of environmental variable U
$M_{A,B}(x)$	fuzzy membership function value between concept A and B, at x value
μ_B	fuzzy membership function associated to concept B
N_{obj}	an object node
N_{act}	an action node
N_{exec}	an executive node
E_Z^z	a z type edge ($z \in ability, instance, association$) connecting nodes Z ($Z \subseteq N_exec, N_act, N_obj$)

Thesis IV. Specific Symbols

$g_i()$	Gaussian activation function of neuron i
c_i	the center parameter belonging to neuron i
σ_i	the width parameter belonging to neuron i
w_i	weight parameter belonging to the output of neuron i
Ψ_j	the status of cluster j: 0 - inactive, 1 - active
R	a set of rules
r_j	the radius of cluster j
ρ	range value
Λ_i	the index of closest adversarial sample to sample i
ω	the final number of rules (clusters used for classification))
θ	an arbitrary threshold value to restrict the output of the classifier above the given value

Thesis V. Specific Symbols

Ψ	the problem space or domain of a given problem
$\Psi^{\mathbb{N}\cup\{0\}}$	the non-negative valued problem space or domain of a given problem
ψ_u	a given subspace of $\Psi^{\mathbb{N}\cup\{0\}}$ around element u
κ	a given class label
C	a multidimensional class matrix marking the positions of classes
M	a multidimensional fuzzy matrix assigning fuzzy membership function values for each element of C
M_κ	the fuzzy matrix associated with class κ
y	the output of the classifier
λ	the class with the largest
θ	the threshold value of the classification
$\bar{C}$	the aggregated class matrix for an illustration
$\bar{M}$	the aggregated fuzzy matrix for an illustration
μ	a fuzzy membership value
ρ	the width of the fuzzy sets
a_i	the scaling coefficient of the linear mapping function
b_i	the offset coefficient of the linear mapping function
$\breve{C}$	the temporary class array for pFHMs
$\breve{M}$	the temporary fuzzy array for pFHMs
$\widehat{C}$	the final class array for pFHMs
$\widehat{M}$	the final fuzzy array for pFHMs
D_j	the size of the domain of attribute j
Q_κ	the number of classes
$A(u)$	an address function that converts 1D array u into a single value
$\text{FHM}_{\rho=v}$	an FHM classifier where the width of all fuzzy sets is v
P'	the cardinality of the input dataset after redundancy and inconsistency filtering
L	the number of non-zero values in C and $\widehat{C}$
S_M	the size of the FHM structure (in bytes)

Thesis VI. Specific Symbols

L_i	a layer assigned to attribute i
Λ^{L_i}	the index array belonging to layer L_i, contains index values
$\Lambda^{L_i}_{u,v}$	an element of index array Λ^{L_i} at column u and row v
M^{L_i}	the fuzzy array belonging to layer L_i, assigns fuzzy membership function values for each element of Λ^{L_i}
$M^{L_i}_{u,v}$	an element of fuzzy array M^{L_i} at column u and row v
G^{L_i}	2D statistical array belonging to layer L_i, counts the class occurrences for each row in the index array Λ^{L_i}
$G^{L_i}_{u,v}$	an element of statistical array G^{L_i} at column u and row v
υ^{L_i}	1D statistical array belonging to layer L_i, indicates the most likely class for each row in the index array Λ^{L_i}
$\upsilon^{L_i}_{v}$	an element of statistical array υ^{L_i} at row v
κ	a given class label
γ_i	the number of index markers in Λ^{L_i}
η_i	the index value acquired in layer L_i
ρ	the width of the fuzzy sets
α_i	the class marker belonging to class i
D_i	the domain size of attribute i
a_i	the scaling coefficient of the linear mapping function
b_i	the offset coefficient of the linear mapping function
$\widehat{\Lambda}^{L_i}$	the class array belonging to layer L_i, contains class labels in v-SFITs
$\widehat{\Lambda}^{L_i}_{u,v}$	an element of class array $\widehat{\Lambda}^{L_i}$ at column u and row v
$\widehat{M}^{L_i}$	the fuzzy array belonging to layer L_i, assigns fuzzy membership function values for each element of $\widehat{\Lambda}^{L_i}$ in v-SFITs
$\widehat{M}^{L_i}_{u,v}$	an element of fuzzy array $\widehat{M}^{L_i}$ at column u and row v
N_x	the length of variable length sample x (i.e. the number of attributes in x)
y	the output of the system
δ	variable to set the magnitude of noise in the example
z	a random value in $[-\delta, \delta]$

$S_{\text{c-SFIT}}$	the size of the c-SFIT structure (in bytes)
$S_{\text{v-SFIT}}$	the size of the v-SFIT structure (in bytes)
S_{Λ}	the size of a single element in Λ (in bytes)
S_{M}	the size of a single element in M (in bytes)
S_{G}	the size of a single element in G (in bytes)

Appendix A

Statistical Measures

In this chapter, a concise description of the statistical measures can be found that were used in the thesis work.

- **True positive (TP)**: an outcome where the classifier correctly predicts the positive class.
- **False positive (FP)**: an outcome where the classifier incorrectly predicts the positive class.
- **True negative (TN)**: an outcome where the classifier correctly predicts the negative class.
- **False negative (FN)**: an outcome where the classifier incorrectly predicts the negative class.
- **true positive rate (TPR, *sensitivity, recall or hit rate)***: The proportion of correctly identified positives:

$$TPR = \frac{TP}{TP + FN} \tag{A.1}$$

- **true negative rate (TNR, *specificity, selectivity*)**: The proportion of correctly identified negatives:

$$TNR = \frac{TN}{TN + FP} \tag{A.2}$$

- **positive predictive value (PPV, *precision*)**: The proportion of actually correct positive identifications:

$$PPV = \frac{TP}{TP + FP} \tag{A.3}$$

- **false negative rate (FNR, *miss rate*)**: . It measures type I error, which is the rejection of a true null hypothesis:

$$FNR = \frac{FN}{FN + TP} \tag{A.4}$$

the probability of falsely rejecting the null hypothesis for a particular test

- **false positive rate (FPR, fall-out ratio)**: Measures the ratio of falsely rejected positives:

$$FPR = \frac{FP}{FP + TN} \tag{A.5}$$

- **classification accuracy (ACC)**: It measures the performance of the system on a balanced dataset:

$$ACC = \frac{TP + TN}{TP + TN + FP + FN} \tag{A.6}$$

- **balanced accuracy (BA)**: It measures the performance of the system on an imbalanced dataset:

$$BA = \frac{TPR + TNR}{2} \tag{A.7}$$

Appendix B

References

B.1 Publications Strictly Related to the Thesis Work

B.1.1 Journal Articles

[P. 1] A.R. Várkonyi-Kóczy, B. Tusor. Human-Computer Interaction for Smart Environment Applications Using Fuzzy Hand Posture and Gesture Models, In *IEEE Trans. on Instrumentation and Measurement*, 60(5):1505-1514, **2011**. **Impact factor: 2.10; Q2, SJR:0.62**

[P. 2] B. Tusor, A.R. Várkonyi-Kóczy, G. Klie, G. Kocsis. Human-Machine Cooperation in an iSpace Robot Room. In *International Journal of Advanced Computational Intelligence and Intelligent Informatics (JACIII)*, 16(6):723-732, 2012. **Q4, SJR:0.181**

[P. 3] B. Tusor, A.R. Várkonyi-Kóczy. A Hybrid Fuzzy-RBFN Filter for Data Classification. In *Advanced Materials Research*, 1117:261-264, 2015. **Q3, SJR:0.115**
DOI:10.4028/www.scientific.net/AMR.1117.261

[P. 4] B. Tusor, A.R. Várkonyi-Kóczy. Fuzzy Classifier Hyper-matrices for Rapid Data Classification. In *Japanese Journal of Applied Physics (JJAP)*, 4:011610-1-6, 2016. **Impact factor: 0.327, Q3, SJR:0.327** DOI:10.7567/JJAPCP.4.011610

B.1.2 Book Chapters

[P. 5] A.A. Tóth, B. Tusor, and A.R. Várkonyi-Kóczy. Fuzzy Hand Posture Models in Man-Machine Communication. In *I. Rudas, J. Fodor, J. Kacprzyk, (eds.), Computational Intelligence in Engineering (Ser. Studies in Computational Intelligence), Springer Verlag, Berlin, Heidelberg*, pages 229-245, 2010. **Q4**

[P. 6] A.R. Várkonyi-Kóczy, B. Tusor. Improving the Model Convergence Properties of

Classifier Feedforward MLP Neural Networks. In *L.A. Zadeh, A.M. Abbasov, R.R. Yager, S.N. Shahbazova, M.Z. Reformat (eds.), Recent Developments and New Directions in Soft Computing, Springer Verlag, Berlin, Heidelberg*, pages 281-293, 2014.

[P. 7] A.R. Várkonyi-Kóczy, B. Tusor, I.J. Rudas. Knowledge representation in ISpace based man-machine communication, In *Jamshidi, M., Kreinovich, V., Kacprzyk, J. (eds), Advanced Trends in Soft Computing (Ser. Studies in Fuzziness and Soft Computing, 312), Springer Verlag, Berlin, Heidelberg*, ISBN:978-3-319-03673-1, pages 241-251, 2014. **Q3**

[P. 8] A.R. Várkonyi-Kóczy, B. Tusor, Rudas, I.J. Data Classification based on Fuzzy-RBF Networks. In *Valentina Emilia Balas, Lakhmi C Jain, Branko Kovačević (eds.) Advances in Intelligent Systems and Computing 357, Springer Verlag, Berlin, Heidelberg*, pages 829-840, 2015. ISBN:978-3-319-18415-9

[P. 9] B. Tusor, A.R. Várkonyi-Kóczy, J. Bukor. A Parallel Fuzzy Filter Network for Pattern Recognition. In *Recent Advances in Technology Research and Education, Springer International Publishing*, pages 275-282, January 2019.
DOI:10.1007/978-3-319-99834-3_36

[P. 10] A.R. Várkonyi-Kóczy, B. Tusor, J.T. Tóth: A Multi-Attribute Classification Method to Solve the Problem of Dimensionality. *In book: Advances in Intelligent Systems and Computing; (519), Berlin; Heidelberg: Springer Verlag*, pages 403-410, January 2017. ISBN: 978-3-319-46490-9

[P. 11] B. Tusor, A.R. Várkonyi-Kóczy and Tóth, J. T.: A Fuzzy Data Structure for Variable Length Data and Missing Value Classification. In *D. Luca, L. Sirghi, C. Costin (eds.) Recent Advances in Technology Research and Education, ser. Advances in Intelligent Systems and Computing (660), Berlin; Heidelberg: Springer*, pages 297-304, January 2017. ISBN: 978-3-319-67458-2

B.1.3 Conference Papers

[P. 12] A.A. Tóth, B. Tusor, and A.R. Várkonyi-Kóczy. New Possibilities in Human-Machine Interaction. In Proc. of the *10th International Symposium of Hungarian Researchers on Computational Intelligence and Informatics, CINTI'2009*, Budapest, Hungary, pages 327-338, 2009.

[P. 13] A.R. Várkonyi-Kóczy, B. Tusor. Man-Machine Communication via CFNN Based Hand Sign Models. In Proc. of the *8th Slovakian-Hungarian Joint Symposium on Applied Machine Intelligence, SAMI2010, Herlany, Slovakia*, pages 329-334, 2010.

[P. 14] B. Tusor, A.R. Várkonyi-Kóczy. Circular Fuzzy Neural Network Based Hand Gesture

and Posture Modeling. In CD-ROM Proc. of the *2010 IEEE Int. Instrumentation and Measurement Technology Conference, I2MTC'2010, Austin, USA*, pages 815-820, 2010.

[P. 15] A.R. Várkonyi-Kóczy, B. Tusor. Improved Back-Propagation Algorithm for Neural Network Training. In CD-ROM Proc. of the *7th IEEE Int. Symposium on Intelligent Signal Processing, WISP'2011, Floriana, Malta*, pages 66-73, 2011.

[P. 16] B. Tusor, A.R. Várkonyi-Kóczy, I.J. Rudas, G. Klie, G. Kocsis. An Input Data Set Compression Method for Improving the Training Ability of Neural Networks. In CD-ROM Proc. of the *2012 IEEE Int. Instrumentation and Measurement Technology Conference, I2MTC'2012, Graz, Austria*, pages 1775-1783, 2012.

[P. 17] A.R. Várkonyi-Kóczy, B. Tusor, A. Dineva. Determination of the Complexity Fitted Model Structure of Radial Basis Function Neural Networks. In Proc. of the *17th IEEE Int. Conference on Intelligent Engineering Systems, INES'2013, Costa Rica,* pages 237-242, 2013.

[P. 18] B. Tusor, A.R. Várkonyi-Kóczy. Intelligent Hotel Room Assistant, In Proc. of the *8th Int. Conference on Informatics in Control, Automation, and Robotics, ICINCO'2011, Noordwijkerhout, The Netherlands*, pages 182-187, 2011.

[P. 19] B. Tusor, A.R. Várkonyi-Kóczy. ISpace Robot Room for Improved Life Comfort. In CD-ROM Proc. of the *12nd Int. Symp. of Hungarian Researchers on Computational Intelligence and Informatics, CINTI'2011, Budapest, Hungary*, pages 199-205, 2011.

[P. 20] B. Tusor, A.R. Várkonyi-Kóczy. Virtual Eto-Environment in iSpace. In Proc. of the *International Workshop on Advanced Computational Intelligence and Intelligent Informatics, IWACIII'2011, Suzhou, China*, pages G2-5.1-8, 2011.

[P. 21] A.R. Várkonyi-Kóczy, B. Tusor. Efficient Knowledge Representation in Intelligent Human-Robot Co-operation. In Proc. of the *16th IEEE Int. Conference on Intelligent Engineering Systems INES'2012, Lisbon, Portugal*, pages 25-30, 2012.

[P. 22] B. Tusor, A.R. Várkonyi-Kóczy. A Rule-based Filter Network for Multiclass Data Classification. In Proc. of the *2015 IEEE International Instrumentation and Measurement Technology Conference, I2MTC'2015, Pisa, Italy*, pages 1102-1107, 2015.

[P. 23] A.R. Várkonyi-Kóczy, B. Tusor, and J.T. Tóth: A Fuzzy Hypermatrix-based Skin Color Filtering Method. In Proc. of the *2015 IEEE 19th International Conference on Intelligent Engineering Systems, INES'2015, Bratislava, Slovakia*, pages 173-178, 2015.

[P. 24] Várkonyi-Kóczy A.R., B. Tusor: Classification with Fuzzy Hypermatrices. In Proc. of the *2016 IEEE International Instrumentation and Measurement Technology Conference, I2MTC'2016, ISBN: 978-1-4673-9220-4, Taipei, Taiwan*, pages 990-995, 2016.

[P. 25] B. Tusor, A.R. Várkonyi-Kóczy, J.T. Tóth. Fuzzy Hypermátrix Alapú Osztályozó Gépészeti Osztályozási Problémákhoz. In *XXIV. Nemzetközi Gépészeti Találkozó. OGET'2016, Déva, Romania*, ISSN: 2068-1267, pages 463-466, 2016.

[P. 26] B. Tusor, Bukor, J., A.R. Várkonyi-Kóczy: Parallelized Fuzzy RBF and FHM based Color Filtering for Real-Time Image Processing. In Proc. of the *2019 IEEE International Instrumentation and Measurement Technology Conference, I2MTC'2019, Auckland, New Zealand*, pages 1-6, 2019.

[P. 27] B. Tusor, A.R. Várkonyi-Kóczy, J.T. Tóth. Active Problem Workspace Reduction with a Fast Fuzzy Classifier for Real-Time Applications. In Proc. of the *IEEE International Conference on Systems, Man, and Cybernetics, SMC'2016, Budapest, Hungary*, pages 4423-4428, 2016. ISBN: 978-1-5090-1819-2

[P. 28] A.R. Várkonyi-Kóczy, B. Tusor, J.T. Tóth. Robust Variable Length Data Classification with Extended Sequential Fuzzy Indexing Tables. In CD-ROM Proc. of the *2017 IEEE International Instrumentation and Measurement Technology Conference (I2MTC'2017), Torino, Italy*, pages 1881-1886, 2017.

[P. 29] B. Tusor, Simon-Nagy, G., A.R. Várkonyi-Kóczy, and J.T. Tóth. Personalized Dietary Assistant - An Intelligent Space Application. In Proc. of the *21st IEEE Int. Conf. on Intelligent Engineering Systems (INES'2017), Larnaca, Cyprus*, pages 27-32, 2017.

[P. 30] B. Tusor, A.R. Várkonyi-Kóczy, J. Bukor. An ISpace-based Dietary Advisor. In Proc. of the *7th IEEE Int. Symposium on Intelligent Signal Processing, WISP'2011, Rome, Italy*, pages 1-6, 2018.

B.2 Further Publications of the Author

[P. 31] B. Tusor, Várkonyi-Kóczy A.R. Fast Regular and Interval-based Classification, using parSITs. *ACTA POLYTECHNICA HUNGARICA*, 18(6):107-126, 2021.
Impact factor: 1.510, Q2

[P. 32] B. Tusor, J.T. Tóth, Várkonyi-Kóczy A.R. Parallelized Sequential Indexing Tables for Fast High-Volume Data Processing. In *2020 IEEE International Instrumentation and Measurement Technology Conference (I2MTC)*, pages 1-6, 2020.

[P. 33] B. Tusor, A.R. Várkonyi-Kóczy. Memory Efficient Exact and Approximate Functional Dependency Extraction with ParSIT. In *Szakál, Anikó (szerk.) 2020 IEEE 24th International Conference on Intelligent Engineering Systems (INES)*, pages 133-138, 2020.

[P. 34] B. Tusor, J.T. Tóth, A.R. Várkonyi-Kóczy. An Indexed Rule-Based Fuzzy Color Filtering Method. In *L. Kovács, T. Haidegger, A. Szakál. (eds.) Recent Advances in*

Intelligent Engineering. Topics in Intelligent Engineering and Informatics, 14:307-312, 2020.

[P. 35] B. Tusor, O. Takac, A. Molnár, S. Gubo, A.R. Várkonyi-Kóczy. Shape Recognition in Drone Images Using Simplified Fuzzy Indexing Tables. In *Szakál, Anikó (eds.) IEEE 18th World Symposium on Applied Machine Intelligence and Informatics (SAMI 2020), Kosice, Slovakia*, pages 1-6, 2020.

[P. 36] B. Tusor, Š. Gubo, T. Kmeť, J. T. Tóth. Augmented Smart Refrigerator - An Intelligent Space Application. In *Varkonyi-Koczy, Annamaria R. (eds.) Engineering for Sustainable Future: Selected papers of the 18th International Conference on Global Research and Education, Inter-Academia2019, Springer*, pages 171-178, 2020.

[P. 37] B. Tusor, J. Bukor, L. Végh, O. Takáč. Domain Reduction Techniques for Sequential Fuzzy Indexing Tables – A Case Study. In *A.R. Várkonyi-Kóczy(eds.): Engineering for Sustainable Future: Selected papers of the 18th International Conference on Global Research and Education, Inter-Academia2019, Cham, Springer*, pages 179-190, 2020.

[P. 38] B. Tusor, J.T. Tóth, Várkonyi-Kóczy A.R. Approximate Functional Dependency Mining with Sequential Indexing Tables. In *Szakál, Anikó (szerk.) IEEE Joint 19th International Symposium on Computational Intelligence and Informatics and 7th IEEE International Conference on Recent Achievements in Mechatronics, Automation, Computer Sciences and Robotics: CINTI-MACRo 2019, Szeged, Hungary*, pages 119-124, 2019.

[P. 39] B. Tusor, J.T. Tóth, A.R. Várkonyi-Kóczy. SIT-based Functional Dependency Extraction. *ACTA POLYTECHNICA HUNGARICA*, 16(10):65-81, 2019.
Impact factor: 1.510, Q2

[P. 40] B. Tusor, J.T. Tóth, A.R. Várkonyi-Kóczy. Functional Dependency Detection with Sequential Indexing Tables. In *Szakál, Anikó (szerk.) IEEE 23nd Int. Conf. on Intelligent Engineering Systems (INES 2019), Budapest, Hungary*, pages 1-6, 2019.

[P. 41] A. Dineva, B. Tusor, I. Vajda. Interval Type-2 Fuzzy System in Personalized Driving Cycle Forecasting. *AIP CONFERENCE PROCEEDINGS*, 1982(1):020027, 2018.

[P. 42] Dineva, A., B. Tusor, P. Csiba, A.R. Várkonyi-Kóczy. Robot Control in ISpace by Applying Weighted Likelihood Functions. In *D. Luca, L. Sirghi, C. Costin: Recent Adv. in Tech. Research and Education, Springer Berlin Heidelberg*, pages 243-248, 2018.

[P. 43] Várkonyi-Kóczy, A., B. Tusor, J.T. Tóth. *A Fuzzy Shape Extraction Method. Studies in Fuzziness and Soft Computing*, pages 383-395, 2018.

[P. 44] B. Tusor, A.R. Várkonyi-Kóczy, M. Takács, J.T. Tóth. A Fast Fuzzy Decision Tree for Color Filtering. In *IEEE International Symposium on Intelligent Signal Processing, WISP2015*, pages 46-51, 2015.

Bibliography

[1] J-H. Lee and H. Hashimoto. Intelligent space. In *Proc. Int. Conf. on Intelligent Robots and Systems, IROS 2000*, volume 2, pages 1358–1363, 2000.

[2] J-H. Lee, K. Morioka, N. Ando, and H. Hashimoto. Cooperation of distributed intelligent sensors in intelligent environment. *IEEE/ASME Trans. on Mechatronics*, 9(3):535–543, 2004.

[3] M. Weiser. The computer for the twentyfirst century. *Scientific American*, 265(3):94–104, 1991.

[4] K. Morioka, J.-H. Lee, and H. Hashimoto. Human centered robotics in intelligent space. In *Proceedings 2002 IEEE International Conference on Robotics and Automation, Washington, DC, USA*, pages 2010–2015, 2002.

[5] D. Brscic and H. Hashimoto. Mobile robot as physical agent of intelligent space. *Journal of computing and information technology*, 17(1):81–94, 2009.

[6] T.-S. Jin, J.-M-Lee, and H. Hashimoto. Position control of mobile robot for human-following in intelligent space with distributed sensors. *International Journal of Control, Automation, and Systems*, 4(2):204–216, 2006.

[7] C.-L. Hwang and L.-J. Chang. Trajectory tracking and obstacle avoidance of car-like mobile robots in an intelligent space using mixed h2/h? decentralized control. *IEEE/ASME Trans. on Mechanics*, 12(3):345–352, 2007.

[8] P. Galambos and P. Baranyi. Virca as virtual intelligent space for rt-middleware. In *Proc. of the 2011 IEEE/ASME Int. Conference on Advanced Intelligent Mechatronics (AIM2011)*, pages 140–145, 2011.

[9] R. Vanijjirattikhan, M-Y. Chow, P. Szemes, and H. Hashimoto. Mobile agent gain scheduler control in inter-continental intelligent space. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation, ICRA 2005*, pages 1115–1120, 2005.

[10] L.A. Jeni, Z. Istenes, P. Korondi, and H. Hashimoto. Mobile agent control in intelligent space using reinforcement learning. In *International Symposium of Hungarian Researchers on Computational Intelligence (HUCI)*, 2006.

[11] L.A. Jeni, Z. Istenes, P. Korondi, and H. Hashimoto. Hierarchical reinforcement learning for mobile robot navigation using the ispace concept. In *11th IEEE Int. Conference on Intelligent Engineering Systems (INES)*, 2007.

[12] L.A. Jeni, P. Korondi, Z. Istenes, and H. Hashimoto. Safe mobile robot control in the ispace environment. In *9th IFAC Symposium on Robot Control (SYROCO)*, 2009.

[13] P. Zanaty, P. Korondi, G. Sziebig, and L.A. Jeni. Image based automatic object localisation in ispace environment. In *In 10th International Symposium of Hungarian Researchers on Computational Intelligence and Informatics (CINTI)*, pages 501–512, 2009.

[14] L.A. Jeni, H. Hashimoto, and T. Kubota. Robust facial expression recognition using near infrared cameras. *Journal of Advanced Computational Intelligence and Intelligent Informatics, Fujipress*, 16(2):1–8, 2012.

[15] K. Yokoi, M. Niitsuma, and H. Hashimoto. Localization of human hand by using inertial sensors. In *2008 SICE Annual Conference, Tokyo*, pages 1818–1822, 2008.

[16] M. Niitsuma, H. Hashimoto, and H. Hashimoto. Spatial memory: an aid system for human activity in intelligent space. In *Proceedings 2006 IEEE International Conference on Robotics and Automation, ICRA 2006, Orlando, FL*, pages 4258–4263, 2006.

[17] M. Niitsuma, K. Kawaji, K. Yokoi, and H. Hashimoto. Extraction of human - object relations in intelligent space. In *RO-MAN 2008 - The 17th IEEE International Symposium on Robot and Human Interactive Communication, Munich*, pages 520–525, 2008.

[18] M. Niitsuma, Takeshi Sasaki, and H. Hashimoto. Enhancement of spatial memory using human-object relations. In *2008 SICE Annual Conference, Tokyo*, pages 3509–3513, 2008.

[19] M. Niitsuma and H. Hashimoto. Comparison of visual and vibration displays for finding spatial memory in intelligent space. In *RO-MAN 2009 - The 18th IEEE International Symposium on Robot and Human Interactive Communication, Toyama*, pages 587–592, 2009.

[20] T. Ichikawa, M. Yuki, P. Korondi, H. Hashimoto, M. Gácsi, and M. Niitsuma. Impression evaluation for different behavioral characteristics in ethologically inspired human-robot communication. In *IEEE RO-MAN: The 21st IEEE International Symposium on Robot and Human Interactive Communication, Paris, 2012*, pages 55–60, 2012.

[21] Y. Takahashi and M. Niitsuma. Enhancement of attachment behavior model for social robot to adapt in daily living environments. In *IECON 2015 - 41st Annual Conference of the IEEE Industrial Electronics Society, Yokohama*, pages 005142–005147, 2015.

[22] T. Ichikawa, M. Yuki, P. Korondi, H. Hashimoto, M. Gácsi, and M. Niitsuma. Nonverbal human-robot communication for ambient assisted living applications based on ethologically inspired social behavior model. In *IECON 2018 - 44th Annual Conference of the IEEE Industrial Electronics Society, 21-23 Oct. 2018, Washington, DC, USA*, pages 6045–6050, 2018.

[23] L. Palafox and H. Hashimoto. A human movement profile classifier using self organized maps in the 4w1h architecture. In *Japanese Robotic Society Conference*, 2009.

[24] T. Kohonen. Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, 43(1):59–69, 1982.

[25] L. Palafox, L. A. Jeni, and Hideki Hashimoto. 5w1h as a human activity recognition paradigm in the ispace. In *In 8th Asian Control Conference (ASCC)*, 2011.

[26] L. Palafox and H. Hashimoto. 4w1h and particle swarm optimization for human activity recognition. *Journal of Advanced Computational Intelligence and Intelligent Informatics*, 15(7):793–799, 2011.

[27] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.

[28] H. Tanaka H. Ishibushi. Fuzzy neural networks with fuzzy weights and fuzzy biases. In *Proc. IEEE Neural Network Conf, San Francisco*, volume 3, pages 1650–1655, 1993.

[29] L.A. Zadeh. Fuzzy sets. *Information and Control*, 8:338–353, 1965.

[30] D.H. Broomhead and D. Lowe. Multivariable functional interpolation and adaptive networks. *Complex Systems*, 2:321–355, 1988.

[31] E. Robson (eds.) M. Campbell-Kelly, M. Croarken. *The History of Mathematical Tables From Sumer to Spreadsheets*. NY, USA, 2003.

[32] B.D. Zarit, B.J. Super, and F.K.H. Quek. Comparison of five color models in skin pixel classification. In *Proc. of the International Workshop on Recognition, Analysis, and Tracking of Faces and Gestures in Real-Time Systems, Corfu, Greece, Sep. 26-27*, pages 58–63, 1999.

[33] P. Luukka and O. Kurama. Similarity classifier with ordered weighted averaging operators. *Expert Systems With Applications*, 40:995–1002, 2013.

[34] B. Cetisli and A. Barkana. Speeding up the scaled conjugate gradient algorithm and its application in neuro-fuzzy classifier training. *Soft Computing*, 14(4):365–378, 2010.

[35] A.A. Tóth and A.R. Várkonyi-Kóczy. A new man-machine interface for ispace applications. *Journal of Automation, Mobile Robotics and Intelligent Systems*, pages 187–190, 2009.

[36] Qing Chen, Marius Cordea, Emil Petriu, Annamaria Varkonyi-Koczy, and Thomas Whalen. Human - computer interaction for smart environment applications using hand gestures and facial expressions. *IJAMC*, 3:95–109, 01 2009.

[37] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. volume 1, pages I–511, 02 2001.

[38] Benjamin Bedregal, Antônio Carlos Rocha Costa, and Graçaliz Dimuro. Fuzzy rule-based hand gesture recognition. volume 217, pages 285–294, 08 2006.

[39] Nguyen Dang Binh, Shuichi Enokida, and Toshiaki Ejima. A new approach dedicated to real-time hand gesture recognition. volume 2, pages 481–488, 01 2006.

[40] A.R. Várkonyi-Kóczy and A.A. Tóth. Ispace? a tool for improving the quality of life, journal of automation. *Journal of Automation, Mobile Robotics and Intelligent Systems*, 3(4):41–45, 2009.

[41] A. Zisserman R. Hartley. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2003.

[42] Gary Bradski. Computer vision face tracking for use in a perceptual user interface. pages 1–15, 1998.

[43] G. Appenzeller, Joo-Ho Lee, and H. Hashimoto. Building topological maps by looking at people: an example of cooperation between intelligent spaces and robots. In *Proceedings of the 1997 IEEE/RSJ International Conference on Intelligent Robot and Systems. Innovative Robotics for Real-World Applications. IROS '97*, volume 3, pages 1326–1333, 1997.

[44] Shahzad Malik Malik. Real-time hand tracking and finger tracking for interaction csc 2503 f project report. pages 1–21, 2003.

[45] A.R. Várkonyi-Kóczy. Autonomous 3d model reconstruction and its intelligent application in vehicle system dynamics. In *5th International Symposium on Intelligent Systems and Informatics (SISY 2007), Subotica, Serbia*, pages 13–18, 2007.

[46] D. Rumelhart, G. Hinton, and R. Williams. Learning representations by back-propagating errors. *Nature*, 323:533?536, 01 1986.

[47] F. Rosenblatt. *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*. Spartan Books, 1961.

[48] Ming-Kuei Hu. Visual pattern recognition by moment invariants. *IRE Trans. Inf. Theory*, 8:179–187, 1962.

[49] Jennifer Schlenzig, Edward Hunter, and Ramesh C. Jain. Vision based hand gesture interpretation using recursive estimation. *Proceedings of 1994 28th Asilomar Conference on Signals, Systems and Computers*, 2:1267–1271 vol.2, 1994.

[50] Simon Conseil, Salah Bourennane, and Lionel Martin. Comparison of fourier descriptors and hu moments for hand posture recognition. *European Signal Processing Conference*, 09 2007.

[51] David Broomhead and David Lowe. Radial basis functions, multi-variable functional interpolation and adaptive networks. *ROYAL SIGNALS AND RADAR ESTABLISHMENT MALVERN (UNITED KINGDOM)*, RE-MEMO-4148, 03 1988.

[52] Daniel Aloise, Amit Deshpande, Pierre Hansen, and Preyas Popat. Np-hardness of euclidean sum-of-squares clustering. *Machine Learning*, 75:245–248, 05 2009.

[53] Shreeram S. Abhyankar. Hilbert's thirteenth problem. In *Algére non commutative, groupes quantiques et invariants (Reims, 1995), Séin. Congr., 2, Soc. Math. France, Paris*, pages 1–11, 1997.

[54] D. Hilbert. Mathematical problems. *Bulletin of the American Mathematical Society*, 8(10):437–479, 1902.

[55] A.N. Kolmogorov. On the representation of continuous functions of many variables by superpositions of continuous functions of one variable and addition. *Doklady Akademii Nauk SSSR*, 114:953–956, 1957.

[56] V.I. Arnold. On the functions of three variables. *Doklady Akademii Nauk SSSR*, 114(4):679–681, 1957.

[57] V.I. Arnold. On the representation of continuous functions of three variables by the superpositions of continuous functions of two variables. *Matem. Sbornik*, 48(1):3–74, 1959.

[58] Kurt Hornik, Maxwell B. Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–366, 1989.

[59] Erich Peter Klement, Laszlo Koczy, and Bernhard Moser. Are fuzzy systems universal approximators? *International Journal Of General System*, 28:259–282, 08 1999.

[60] J. Park and I. W. Sandberg. Universal approximation using radial-basis-function networks. *Neural Computation*, 3(2):246–257, 1991.

[61] Wiens. *Radial Basis Function Network*, 2014 (Retrieved June 1, 2020). MATLAB Central File Exchange, `https://www.mathworks.com/matlabcentral/fileexchange/22173-radial-basis-function-network`.

[62] Barry Brumitt, Brian Meyers, John Krumm, Amanda Kern, and Steven Shafer. Easyliving: Technologies for intelligent environments. pages 12–29, 09 2000.

[63] B. Johanson, A. Fox, and T. Winograd. The interactive workspaces project: Experiences with ubiquitous computing rooms. *IEEE Pervasive Comput.*, 1:67–74, 2002.

[64] Franco Cicirelli, Giancarlo Fortino, Andrea Giordano, Antonio Guerrieri, Giandomenico Spezzano, and Andrea Vinci. On the design of smart homes: A framework for activity recognition in home environment. *Journal of Medical Systems*, 40, 09 2016.

[65] J.F. Sowa. Conceptual graphs for a data base interface. *IBM J. of Research and Development*, 20(4):336–357, 1976.

[66] P. Zhang, J. Peng, and J. Xi. A concept graph based knowledge model for software architecture. In *WRI World Congress on Software Engineering 3*, pages 125–128, 2009.

[67] M.P. Palacio, D. Sol, and J. Gonzalez. Graph-based knowledge representation for gis data. In *4th Mexican International Conference on Computer Science*, pages 117–124, 2003.

[68] Q. Qu, J. Qiu, Ch. Sun, and Y. Wang. Graph-based knowledge representation model and pattern retrieval. In *5th International Conference on Fuzzy Systems and Knowledge Discovery 5*, pages 541–545, 2008.

[69] A. Zenonos, A. Khan, G. Kalogridis, S. Vatsikas, T. Lewis, and M. Sooriyabandara. Healthyoffice: Mood recognition at work using smartphones and wearable sensors. In *2016 IEEE International Conference on Pervasive Computing and Communication Workshops (PerCom Workshops)*, pages 1–6, 2016.

[70] R. A. Brooks. The intelligent room project. In *Proceedings Second International Conference on Cognitive Technology Humanizing the Information Age*, pages 271–278, 1997.

[71] Paulraj M P, C.R. Hema, Pranesh Krishnan, and Siti Radzi. Color recognition algorithm using a neural network model in determining the ripeness of a banana. 10 2009.

[72] R.Vigneshwar and Ms.V.Prema. Colour recognition in images using neural networks. volume 4, pages 2455–2460, 2 2016.

[73] Lidwine Gross, Sylvie Thiria, and Robert Frouin. Applying artificial neural network methodology to ocean color remote sensing. *Ecological Modelling*, 120(2):237 – 246, 1999.

[74] R. Yasir, M. A. Rahman, and N. Ahmed. Dermatological disease detection using image processing and artificial neural network. In *8th International Conference on Electrical and Computer Engineering*, pages 687–690, 2014.

[75] N. S. Altman. An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician*, 46(3):175–185, 1992.

[76] K. Adi, S. Pujiyanto, O. D. Nurhayati, and A. Pamungkas. Beef quality identification using color analysis and k-nearest neighbor classification. In *2015 4th International*

Conference on Instrumentation, Communications, Information Technology, and Biomedical Engineering (ICICI-BME), pages 180–184, 2015.

[77] N. Krithika and A. G. Selvarani. An individual grape leaf disease identification using leaf skeletons and knn classification. In *2017 International Conference on Innovations in Information, Embedded and Communication Systems (ICIIECS)*, pages 1–5, 2017.

[78] O. R. Indriani, E. J. Kusuma, C. A. Sari, E. H. Rachmawanto, and D. R. I. M. Setiadi. Tomatoes classification using k-nn based on glcm and hsv color space. In *2017 International Conference on Innovative and Creative Information Technology (ICITech)*, pages 1–6, 2017.

[79] Saman Sarraf. Hair color classification in face recognition using machine learning algorithms. *American Scientific Research Journal for Engineering, Technology, and Sciences*, 26:317–334, 12 2016.

[80] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, Sep 1995.

[81] H. Lin, S. Yen, J. Yeh, and M. Lin. Face detection based on skin color segmentation and svm classification. In *2008 Second International Conference on Secure System Integration and Reliability Improvement*, pages 230–231, 2008.

[82] M.A. Ebrahimi, M.H. Khoshtaghaza, S. Minaei, and B. Jamshidi. Vision-based pest detection based on svm classification method. *Computers and Electronics in Agriculture*, 137:52 – 58, 2017.

[83] Nakhoon Baek, Sun-Mi Park, Ku-Jin Kim, and Seong-Bae Park. Vehicle color classification based on the support vector machine method. In De-Shuang Huang, Laurent Heutte, and Marco Loog, editors, *Advanced Intelligent Computing Theories and Applications. With Aspects of Contemporary Intelligent Computing Techniques*, pages 1133–1139, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[84] Esraa Elhariri, Nashwa El-Bendary, Mohamed Mostafa M. Fouad, Jan Platoš, Aboul Ella Hassanien, and Ahmed M. M. Hussein. Multi-class svm based classification approach for tomato ripeness. In Ajith Abraham, Pavel Krömer, and Václav Snášel, editors, *Innovations in Bio-inspired Computing and Applications*, pages 175–186, Cham, 2014. Springer International Publishing.

[85] Muhammad Zubair Asghar, Aurangzeb Khan, Shakeel Ahmad, Maria Qasim, and Imran Ali Khan. Lexicon-enhanced sentiment analysis framework using rule-based classification scheme. *PLOS ONE*, 12(2):1–22, 02 2017.

[86] Srishti Vashishtha and Seba Susan. Fuzzy rule based unsupervised sentiment analysis from social media posts. *Expert Systems with Applications*, 138:112834, 2019.

[87] E. Soares, P. Angelov, B. Costa, and M. Castro. Actively semi-supervised deep rule-based classifier applied to adverse driving scenarios. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2019.

[88] Francis Bbosa, Ronald Wesonga, and Peter Jehopio. Clinical malaria diagnosis: rule-based classification statistical prototype. *SpringerPlus*, 5, 2016.

[89] Thien Le, Frederic Stahl, Mohamed Medhat Gaber, João Bártolo Gomes, and Giuseppe Di Fatta. On expressiveness and uncertainty awareness in rule-based classification for data streams. *Neurocomputing*, 265:127 – 141, 2017. New Trends for Pattern Recognition: Theory and Applications.

[90] Allah Bux Sargana, Xiaowei Gu, Plamen Angelov, and Zulfiqar Habib. Human action recognition using deep rule-based classifier. *Multimedia Tools and Applications*, 08 2020.

[91] T. Celik, H. Ozkaramanli, and H. Demirel. Fire pixel classification using fuzzy logic and statistical color model. In *2007 IEEE International Conference on Acoustics, Speech and Signal Processing - ICASSP '07*, volume 1, pages I–1205–I–1208, 2007.

[92] Spiros Fotopoulos, Antony Fotinos, and Socrates Makrogiannis. *Fuzzy Rule-Based Color Filtering Using Statistical Indices*, pages 72–97. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

[93] Ismat Beg and Samina Ashraf. Similarity measures for fuzzy sets. *Applied and Computational Mathematics*, 8:192–202, 03 2009.

[94] D. Ramyachitra and P. Manikandan. Imbalanced dataset classification and solutions : A review. volume 5, pages 1–29, 7 2014.

[95] R. Chandra. *Parallel Programming in OpenMP*. Morgan Kaufmann, 2001.

[96] Alaa Tharwat. Classification assessment methods. *Applied Computing and Informatics*, 2018.

[97] M. Buckland and F. Gey. The relationship between recall and precision. *J. Am. Soc. Inf. Sci.*, 45:12–19, 1994.

[98] Mohammad Akhbarizadeh and Mehrdad Nourani. Hardware-based ip routing using partitioned lookup table. *Networking, IEEE/ACM Transactions on*, 13:769 – 781, 09 2005.

[99] Ronan O'Dowd. Automated pxi-based screening and characterisation of tunable lasers. pages 84 – 89, 11 2004.

[100] Kiran George and Henry Chen. Configurable and expandable fft processor for wideband communication. pages 1 – 6, 06 2007.

[101] H.F.-W Sadrozinski and J. Wu. *Applications of Field-Programmable Gate Arrays in Scientific Research*. 04 2016.

[102] E.F. Krause. *Taxicab Geometry*. Dover Publications, 1987.

[103] You-Dong Liang and B. A. Barsky. A new concept and method for line clipping. *ACM Trans. Graph.*, 3(1):1–22, January 1984.

[104] Vasileios Drakopoulos and Dimitrios Matthes. A simple and fast line-clipping method as a scratch extension for computer graphics education. 07:40–47, 06 2019.

[105] Carl Looney. A fuzzy classifier network with ellipsoidal epanechnikovs. pages 1–20, Oct. 2002.

[106] Takaya Saito and Marc Rehmsmeier. The precision-recall plot is more informative than the roc plot when evaluating binary classifiers on imbalanced datasets. *PloS one*, 10:e0118432, 03 2015.

[107] A. Graves, M. Liwicki, S. Fernández, R. Bertolami, H. Bunke, and J. Schmidhuber. A novel connectionist system for unconstrained handwriting recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(5):855–868, 2009.

[108] A. I. Moustapha and R. R. Selmic. Wireless sensor network modeling using modified recurrent neural networks: Application to fault detection. *IEEE Transactions on Instrumentation and Measurement*, 57(5):981–988, 2008.

[109] A. Malhi, R. Yan, and R. X. Gao. Prognosis of defect propagation based on recurrent neural networks. *IEEE Transactions on Instrumentation and Measurement*, 60(3):703–711, 2011.

[110] I. Kamwa, R. Grondin, V. K. Sood, C. Gagnon, Van Thich Nguyen, and J. Mereb. Recurrent neural networks for phasor detection and adaptive identification in power system control and protection. *IEEE Transactions on Instrumentation and Measurement*, 45(2):657–664, 1996.

[111] Michael C. Mozer. *A Focused Backpropagation Algorithm for Temporal Pattern Recognition*, page 137–169. L. Erlbaum Associates Inc., USA, 1995.

[112] Y. Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council*, 5:157–66, 02 1994.

[113] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.

[114] Annamaria Varkonyi-Koczy. *Model Based Anytime Soft Computing Approaches in Engineering Applications*, volume 196, pages 63–92. 03 2009.

[115] J. M. Keller, M. R. Gray, and J. A. Givens. A fuzzy k-nearest neighbor algorithm. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-15(4):580–585, 1985.

[116] E. Akbas. *Fuzzy k-NN*, 2007 (Retrieved June 1, 2020). MATLAB Central File Exchange, `https://www.mathworks.com/matlabcentral/fileexchange/13358-fuzzy-k-nn`.

[117] P. Luukka. *Similarity classifier with OWA operators*, 2012 (Retrieved June 1, 2020). MATLAB Central File Exchange, `https://www.mathworks.com/matlabcentral/fileexchange/38871-similarity-classifier-with-owa-operators`.

[118] Pasi Luukka, Kalle Saastamoinen, and V. Kononen. A classifier based on the maximal fuzzy similarity in the generalized lukasiewicz-structure. pages 195 – 198, 02 2001.

[119] Bayram Cetisli. *Neuro-fuzzy classifier*, 2010 (Retrieved June 1, 2020). MATLAB Central File Exchange, `https://www.mathworks.com/matlabcentral/fileexchange/29043-neuro-fuzzy-classifier`.

[120] Chuen-Tsai Sun and Jyh-Shing Roger Jang. A neuro-fuzzy classifier and its applications. *[Proceedings 1993] Second IEEE International Conference on Fuzzy Systems*, pages 94–98 vol.1, 1993.

[121] Martin Møller. Moller, m.f.: A scaled conjugate gradient algorithm for fast supervised learning. *Neural Networks*, 6:525–533, 12 1993.

[122] J. Steinier, Y Termonia, and J. M. Deltour. Smoothing and differentiation of data by simplified least square procedure. *Analytical chemistry*, 44(11):1906–1909, 1972.

[123] O. L. Mangasarian and W. H. Wolberg. Cancer diagnosis via linear programming. *SIAM News*, 23(5):1–18, 1990.

[124] Marek Sikora and Ł Wróbel. Application of rule induction algorithms for analysis of data collected by seismic hazard monitoring systems in coal mines. *Archives of Mining Sciences*, 55:91–114, 01 2010.

[125] Qi Lyu and Jun Zhu. Revisit long short-term memory : An optimization perspective. 2014.

[126] R. Lakamper L. J. Latecki and T. Eckhardt. Shape descriptors for nonrigid shapes with a single closed contour. pages 424–429, 2000.

[127] Stephen Billings. Nonlinear system identification: Narmax methods in the time, frequency, and spatio-temporal domains. *Nonlinear System Identification: NARMAX Methods in the Time, Frequency, and Spatio-Temporal Domains*, page 574, Oct. 2013.

[128] Hamed Nikdel, Yahya Forghani, and S. Moattar. Increasing the speed of fuzzy k-nearest neighbours algorithm. *Expert Systems*, 35(4):e12254, Oct. 2017.

[129] Nutrient Data Laboratory US Department of Agriculture, Agricultural Research Service. Usda national nutrient database for standard reference [electronic resource], 2016.