Int. Computer Symposium, Dec. 15-17, 2004, Taipei, Taiwan.

A Genetic Functional-Link Network for Camera Movement Classification

C. L. Philip Chen
Department of Elec. Engineering
The University of Texas at San Antonio
San Antonio 78249
Email: pchen@utsa.edu

Chandrakumar Bhumireddy
Department of Elec. Engineering
The Univ. of Texas at San Antonio
San Antonio 78249

Pavan K. Darvemula
Department of Elec. Engineering
The Univ. of Texas at San Antonio
San Antonio 78249

Abstract— **In this paper a Genetic Functional-Link Network (GFLN) that employs Gaussian functions in a feedforward functional-link network (FLN) for classifying camera movement for compressed videos is proposed. The parameters in GFLN are adjusted using genetic evolutionary approach. GFLN provides feature selection capability by selecting the links between input layer and functional nodes dynamically. Genetic coding is used for combining evolution of weights and Gaussian parameters in a single chromosome. Seven categories of camera movement: *static, pan-right, pan-left, tilt-up, tilt-down, zoom-in,* and *zoom-out* decoded from the MPEG-1 video stream are used for neural classification. Our aim is to rapidly extract and process motion vector information from MPEG video without full frame decompression. Video streams with aforementioned classes of camera movement have been successfully classified.**

Keywords: Camera movement classification, Functional-link networks, Genetic Algorithms, MPEG.

I. Introduction

The Moving Picture Expert Group (MPEG) defines the MPEG-1 and MPEG-2 ISO/IEC standards. In order to achieve high compression ratio, MPEG-1 uses hybrid coding techniques to reduce both spatial redundancy and temporal redundancy. The syntax of MPEG-1 defines three main types of coded pictures: intra coded I frames, forward predicted P frames, forward and backward predicted B frames arranged in a repetitive pattern called a GOP. A GOP always starts with an I frame followed by any number of I an P frames (referred to as anchor frames) with any number of B frames between a pair of anchor frames.

Many algorithms for video-shot segmentation have been proposed in both the uncompressed domain and compressed domain. A good survey paper on video segmentation in uncompressed domain can be found in [1]. In compressed domain various methods exist for video-shot segmentation. Especially, DC coefficients in Discrete Cosine Transform (DCT) have been extensively used. Other categories of video segmentation include model-based and feature-based. In model-based classification a physical model of the camera is modelled in 3-D space, with a mapping projection that maps the object on to the camera's image plane. The camera operations, usually, can be estimated from the parameters associated with the projected model. Tan et al. [2] estimated the camera movement based on the model of the camera and 3-D coordinate transforms involved in the mapping. Feature-based methods develop an *'N'* dimensional feature vector combining various statistical features based on the motion vector orientation and magnitude along with some other features extracted from the DCT coefficients [3]. Statistical information of motion vectors such as motion smoothness [4] (ratio of blocks with significant motion to blocks whose motion vector has changed significantly), motion activity in video frames [5], and motion histograms [6] have also been used in shot detection. All the above-mentioned approaches are limited to shot detection and do not consider *shot classification*. Some methods also require computationally expensive reconstruction of DC terms. Furthermore, these approaches are based on many adjustable thresholds and hence cannot be considered robust. Hence, recently neural methods and evolving neuro-methods have been proposed using motion vector (MV) information which has overcome many problems associated with adjustable thresholds. Furthermore, neuro-methods are able to detect and classify shot transitions. For example, Koprinska et al. [7] have proposed a neural-network-based (Linear vector quantization) classifier and an Evolving Fuzzy Neural Network-based (EFuNN) classifier for detecting and classifying camera motion in MPEG video streams [8]. Seok-Woo et al. [9] have defined a rule set for denazifying and classifying the camera shots using a fuzzy associative memory. However, linguistic feature of fuzzy computation takes additional step in mapping for real-time video stream applications.

Due to the fact that neural networks have been drawing increasing attention as powerful tools to solve different tasks of engineering and scientific problems. In this work, we propose using a Genetic Functional-Link Network (GFLN) for camera movement classification. The application can be easily applied to camera-vision-based mobile robot navigation and security surveillance. Functional-Link Neural Network, proposed by [10] and improved by others [11] [12] is essentially a single-layer functional layer, or a flat network that takes advantages of fast learning. This paper presents a novel approach to the learning task of traditional functional-link networks. As referred in [11], a flat functional-link network has the capability of rapid learning and incremental learning. This paper extends the model presented in [11] by incorporating efficient genetic-neuro learning algorithm.

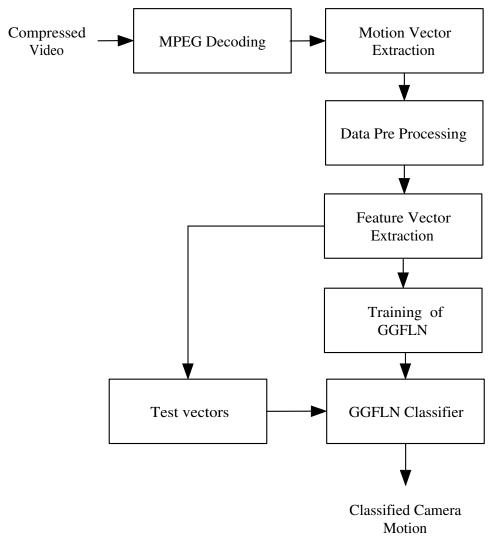


Fig. 1. Overview of proposed method

Here, we adapt radial basis function (RBF) such as Gaussian function in the functional layer. In this model, weights, i.e., links between input layer and functional layer are adjustable in the process of training. A fine grain continuous genetic algorithm [13] is used for training the network parameters. In GFLN the parameters to be adjusted are weights between input layer and functional layer, and parameters of Gaussian functions in the functional layer.

GFLN is trained with feature vectors based on motion vectors decoded from the MPEG-1 video stream. We have used the following seven categories of video streams of camera movement: *static, pan-right, pan-left, tilt-up, tilt-down, zoom-in* and *zoom-out* for neural classification.

The proposed approach for camera movement classification is shown in Figure 1. The structure of this paper is as follows. Section II describes the architecture of GFLN followed by the introduction of the genetic evolutionary training algorithm in section III. Section IV explains implementation of GFLN. Then section V reports simulation results of GFLN. Finally, section VI draws the conclusions.

II. Architecture of GFLN

The neural network proposed in this paper is shown in Figure 2, which consists of three parts: input layer, functional layer and output layer. In this model, the process of training adjusts weights from input nodes to functional layer. We use a Gaussian function as the activation function in the functional neuron layer. Let $\mathbf{x} = [x_1, x_2, \cdots, x_n]$ be the input vector and $\mathbf{u} = [u_1, u_2, \cdots, u_n]$ be the output vector of enhancement node layer (functional layer). Output of a neuron in the enhancement layer is given as follows.

$$u_i = e^{-r_i^2(a_i - c_i)^2} \tag{1}$$

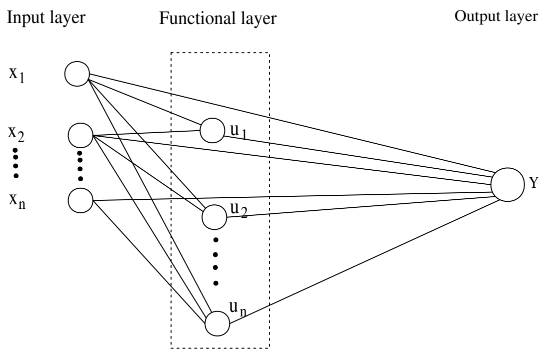


Fig. 2. Architecture of GFLN

where u_i is the output of i^{th} enhancement node and a_i is the linear combination of input elements for i^{th} neuron, c_i and r_i are the parameters of Gaussian function. Once the output of the functional layer is computed, the output vector u is added as extra input layer to the network.

III. Training Algorithm

Genetic learning is used for training the neural network for number of neurons needed (i.e., u_i), the values of weights (i.e., w_i), and the parameters of Gaussian functions (i.e., c_i and r_i). First, we briefly introduce genetic algorithms, and the representation followed by the genetic operators used in the algorithm.

A. Genetic Algorithms

Genetic algorithms are a family of computational models inspired by Darwin's theory of evolution. Genetic algorithms have been quite successfully applied to optimization problems. More precisely, genetic algorithms maintain a population of structures that evolve according to rules of selection and other operators such as crossover and mutation. Each individual is evaluated, assigning a measure of its fitness in the population. In our learning, we use similar approach as a fine-grained continuous genetic algorithm learning proposed in [14]. In this model, each member of population performs crossover with its immediate neighbors, where neighborhood is defined by the topology and some distance parameter. We used square topology with a distance parameter of three.

B. Representation

Majority of traditional genetic algorithms use binary representation for solving the problem. Here we use real coding that is conceptually closest to the problem space and allows for an easy and efficient implementation. Each chromosome consists real part for evolution of network parameters and binary part for selection of the links between input layer and functional layer. In binary string a '1' means a link is present '0' means absence of link. If all the links corresponding to an input are zero then that input is removed from the input layer. Similarly, if all the input links corresponding to a RBF node are

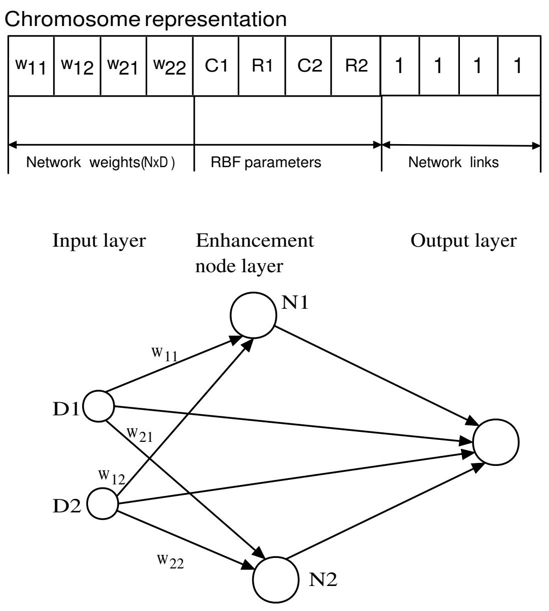


Fig. 3. Chromosome representation of GFLN

zero then output of that node will be constant. This node does not contribute to the process of learning, hence it can be removed from the network architecture. Figure 3 shows the representation of GFLN.

C. Genetic operators

Uniform crossover and non-uniform mutation operator are used as genetic operators.

1) Crossover: One offspring is generated from two parents by arithmetical crossover. It is a linear combination of the two parents. If P_1 and P_2 are the two parents to be crossed, offspring is generated as follows.

$$P_f = P_1(1-a) + aP_2 \tag{2}$$

where a is a random value in $[0,1]$.

2) Mutation: The generated offspring is mutated before reinsertion into the population. For the offspring, P_f, if q is any element and let upper bound (UB) and lower bound (LB) of q be $[LB, UB]$. The result after mutation, $q\prime$, is given by

$$q\prime = \begin{cases} q + \Delta(t, UB - q) & \text{if a random binary digit is zero} \\ q - \Delta(t, q - LB) & \text{if a random binary digit is one} \end{cases} \tag{3}$$

The function $\Delta(t, y)$ returns a value in the range of $[0, y]$ such that probability of being close to 0 increases as t increases (t is the iteration number). This property causes this operator search space uniformly initially (while t is small), and very locally at later stages. The function $\Delta(t, y)$ is given as follows

$$\Delta(t, y) = y.r.(1 - \frac{t}{T})^b \tag{4}$$

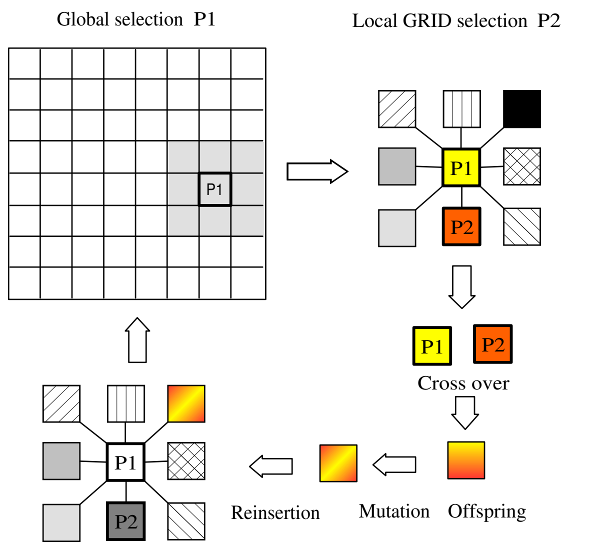


Fig. 4. Fine Grain Genetic Algorithm Cycle, where the original upper right corner cell has been replaced by the new offspring

where r is a random value in $[0,1]$, t is the iteration number, T is maximum number of iterations, and b is a parameter determining the degree of non-uniformity. The value of 5 is used in our implementation. Above-mentioned mutation operator is considered in [15].

3) Selection: In fine grain model selection is in two stages as shown in the Figure 4. The first step is global selection. An individual is selected globally based on its fitness value. Once an individual is selected, the second individual for crossover is selected from the local deme given by grid shape and distance parameter three, based on fitness value. Generated offspring after mutation is reinserted in the same deme replacing the worst individual in the local deme.

D. Error methods

Many different error methods can be used for training. Most commonly used is the root mean square error $(RMSE)$. Other error methods frequently used are mean square error (MSE), and average percentage error (APE) as indicated in the following.

1)

$$RMSE = \sqrt{\frac{1}{N * O}\sum_{o=1}^{O}\sum_{n=1}^{N}(y_{on} - y\prime_{on})^2} \tag{5}$$

where
N: Number of patterns
O: Number of outputs
y_{on}: Desired output of pattern
$y\prime_{on}$: Network output of pattern

2)

$$APE = \frac{100}{N}\sum_{n=1}^{N}\frac{|y_n - y\prime_n|}{y_n} \tag{6}$$

These error functions are used to compare the simulation results with previous approaches.

E. Fitness Function

Fitness or cost function is a measurement of performance of an individual in the population. A simple fitness function for genetic evolution of neural networks is root mean square error (RMSE) of the corresponding network. As indicated by [16] that the generalization performance of a network depends on the size of the weights rather than on the network size. Thus a cost function that considers the size of weights in a network guarantees better generalization performance. The cost function proposed for GFLN is as follows.

$$\text{Fitness } = E + \frac{W}{\alpha_1} + \alpha_2 \frac{n}{N} \tag{7}$$

where
E: Root mean square error (RMSE)
W: Cumulative of sum of network weights
α_1: A constant, a range between [1000,10000] is used
α_2: A constant, a range between $[10^{-1}, 10^{-3}]$ is used
n: Number of links present between input layer and functional layer i.e number of '1' in the binary part of chromosome
N: Length of binary chromosome

F. Learning algorithm

Learning algorithm of GFLN is as follows.

```
BEGIN
  1) Generate initial population of weight and RBF
     parameters in their domains.
  2) Compute error and fitness of each individual.
  3) Run genetic evolutions for given maximum number
     of iterations.
  4) Save the fittest individual of the evolution.
     IF Goal is true, STOP.
     ELSE add an extra enhancement node and go to
     Step 2.
END
```

The algorithm mentioned above is effective as the chromosomes are represents combination of weights and RBF parameters. Genetic algorithm is driven to find an optimal network that is the result of adjusting both network weights and RBF parameters. Feature selection is possible through the evolution of input links simultaneously. In $GFLN$, output network weights are computed through the SVD (Singular value decomposition). The cumulative sum of output weights found through the SVD is considered along with $RMSE$ of desired and network output values in computing the fitness of the network as shown in Eq. (7). Based on [16], the generalization capacity of a network depends on the size of network weights rather than the number of weights in the network. From Eq. (7) it is clear that the evolution is driven in a direction of finding a network with smaller network weights consequently better performance. Designed algorithm guarantees the $GFLN$ having a good generalization capability and better performance.

IV. Camera movement classification using GFLN

A. Decoding and Preprocessing of motion vectors

Our aim is to rapidly extract and process motion vector information from MPEG videos without full frame decompression. We have used MPEG -1 video streams for camera movement classification in this work. However, our algorithm could be readily extended to MPEG-2 and MPEG-4 video streams. The typical GOP structure (15 frame sequence IPPPPPPPPPPPPPP) as show in Figure 5 is used in the encoding at a rate of 30 frames/sec. In this paper we **only** use motion vectors from P frames without any loss of motion information. Using both P and B frames tends to yield better accuracy but at a much higher computational cost. We also assume that the camera movement is the only dominant motion in our test video streams. We encoded JPEG (352X288 pixels) image sequences using Berkely's MPEG-1 encoder software with the GOP structure described above at a rate of 30 frames/sec. We then extracted the motion vectors for each frame by partial decompression.

The extracted motion vector field portrays a characteristic pattern in the direction corresponding to the seven classes of camera movement:

1) Static: The MV field contains majority of zero motion blocks. The MV field may also contain some random MVs due to uniform background.
2) Pan-Left: The MV field predominantly consists of motion vectors pointing to the left.
3) Pan-Right: The MV field predominantly consists of motion vectors pointing to the right.
4) Tilt-UP: The MV field predominantly consists of motion vectors pointing upwards.
5) Tilt-Down: The MV field predominantly consists of motion vectors pointing downwards.
6) Zoom-In: The MV field contains a point of expanding focus with motion vectors predominantly pointing outwards from a focal point.
7) Zoom-Out: The MV field contains a point of contracting focus with motion vectors predominantly pointing inwards from a focal point.

The extracted MV fields of each frame then go through series of data preprocessing and feature extraction to be classified by our genetic-neuro classifier (GFLN). Raw motion vector field is filtered using 3x3 vector median filter. Figure 6 shows noisy and smoothed motion vector field after filtering the noise. Figure 6a shows the frames of zoom-in and tilt-up video, Fig 6b shows corresponding raw motion vectors after decoding. Raw motion vectors have some random motion vectors which are smoothed by applying the median filter. Figure 6c shows the filtered motion vectors free of noise.

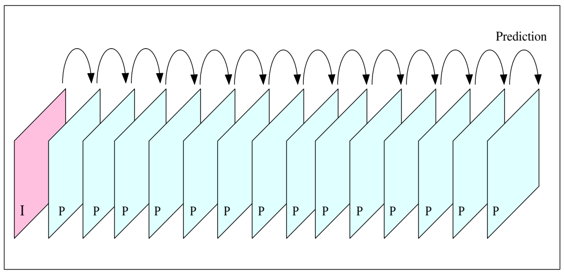
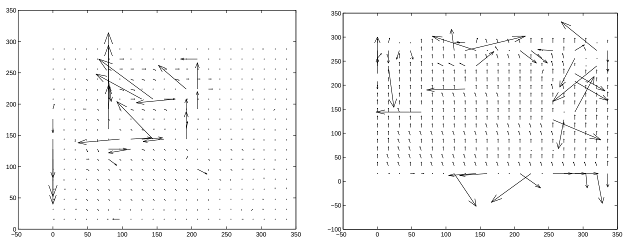
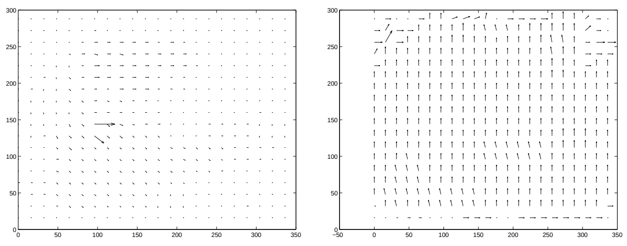


Fig. 5. GOP structure

(a) Original Frames



(b) Motion vectors before filtering the noise



(c) Smoothed motion vectors

Fig. 6. Motion vectors before and after filtering the noise, left column shows motion vectors of a zoom-in video and right column is of tilt-up video

B. Extraction of feature vector

A good feature vector should reduce the dimension of the problem data without discarding the valuable information. Magnitude and direction of the motion vectors are the key features for classification of camera movement. Direction of motion vectors plays a more significant role in MV pattern recognition than magnitude. A 22-dimensional feature vector is extracted from the motion vectors of each frame. First component (V_1) of the feature vector is fraction of macro blocks that have no motion signifying staticness in the frame.

$$V_1 = \frac{B_z}{B_n}$$

where
B_z is number of blocks with zero motion
B_n is the total number of blocks in the frame

Motion vector field of a frame is divided into 7 vertical slices and for each slice 3 parameters are computed resulting in the remaining 21 components of the feature vector as follows:

1) 1st parameter of 7 slices, ($V_2 - V_8$), i.e., average magnitude of each slice: Represents amount of motion in the slice.

$$V_k = \frac{\sum_{i=1}^{N}(R_i)}{N} \qquad 2 \le k \le 8$$

where
V_k is average magnitude of MVs in the slice $k-1$
N is total number of motion vectors in a slice
R_i is the magnitude of i^{th} MV in the slice $k-1$

2) 2nd parameter of 7 slices, ($V_9 - V_{15}$), i.e., average direction of each slice: Represents spatial distribution of MV.

$$V_k = \frac{\sum_{i=1}^{N}(\Theta_i)}{N} \qquad 9 \le k \le 15$$

where
V_k is the average directions of MVs in the slice $k-8$
N is total no of motion vectors in a slice
Θ_i is the direction of i^{th} MV in the slice $k-8$

3) 3rd parameter of 7 slices, ($V_{16} - V_{22}$), i.e., standard deviation of directions of each slice: Captures the uniformity in the direction of MV.

$$V_k = std(\Theta_1, \Theta_2, .., \Theta_i, ...\Theta_N) \qquad 16 \le k \le 22$$

where
V_k is the standard deviation of directions in the slice $k-15$
N is total no of motion vectors in a slice
Θ_i is the direction of i^{th} MV in the slice $k-15$

V. Simulation Results

Simulations are performed to show the effectiveness of GFLN for camera movement classification. We have collected various videos with different camera movement from open-video website (www.open-video.org). Our experiment consists of classification of camera movement into one of of 7 categories: static, pan-left, pan-right, tilt-up, tilt-down, zoom-in or zoom-out by training the GFLN with feature vectors obtained from the video sequences. A total of 1400 frames are collected of which each class contributes to 200 frames of data. Using the data preprocessing methods explained in the section IV, 1400 feature vectors are extracted from motion vectors corresponding to 1400 frames. Dimension of a feature vector is of size 23 including the class variable. Each output class is given a value from 1 to 7 representing its output class.

GFLN is trained with 10-fold cross validation i.e., feature data is split into 10 subsets of equal size and network is trained with 10 times each time leaving one of subsets from training but using only the omitted subset for testing the network. Average performance of GFLN over each subset is listed in the Table I.

TABLE I
10-FOLD CROSS VALIDATION RESULTS OF GFLN USING 15 NODES

subset no	Accuracy on training subset	Accuracy on testing subset
1	79.80	73.68
2	78.93	85.52
3	78.63	75.26
4	79.89	77.10
5	79.56	77.36
6	79.23	82.10
7	79.09	80.78
8	79.86	74.73
9	80.21	70.78
10	77.45	85.00

TABLE II
RESULTS FOR DIFFERENT NUMBER OF ENHANCEMENT NODES

no of nodes	Accuracy on training subset	Accuracy on testing subset
5	78.80	75.68
10	79.93	83.52
15	81.63	85.26
20	82.89	87.10

Table I shows that GFLN is effective in capturing patterns in camera movement. Performance on each subset is consistent for different number of simulation runs. Simulations are performed by changing the number of enhancements nodes in the GFLN architecture. The average performance of GFLN for different number of enhancement nodes is evaluated and listed in Table II. Results show that increasing the number of nodes increases performance at the cost of time complexity. Performance of GFLN for individual category is shown in the Table III using 15 nodes.

TABLE III
CLASSIFICATION RESULTS

category type	Accuracy on training subset	Accuracy on testing subset
static	79.80	73.68
pan-right	82.93	85.52
pan-left	81.63	75.26
tilt-up	79.89	77.12
tilt-down	89.27	77.23
zoom-in	74.36	72.41
zoom-out	72.47	73.67

VI. CONCLUSION

We have proposed a method for the classification of camera movement in compressed domain using the motion vector patterns in MPEG video stream. Results show that GFLN is successful in classifying various camera motions. Note that motion vectors **only** from P frames is used in this paper. Using both P and B frames will tend to yield better accuracy but with the cost of higher computational cost. Future work will include modelling a robust feature vector from the compressed video frame and to include other complex camera effects such as fade-in, fade-out, dissolve etc. for classification.

REFERENCES

[1] G. Ananger and T. Little, "A survey of technologies for parsing and indexing digital video," *Journal of Visual Communications and Image representation*, vol. 7(1), pp. 28–43, Summer 1995.

[2] S. Y.-P. Tan, D.D Saur and P.J.Ramadge, "Rapid estimation of camera motion from compressed video with application to video annotation." *IEEE Trans. on Circuits and systems for video Tecnology*, 1999.

[3] Y. X. L. W. Xiangyang Xue, Xingquan Zhu, "Using mutual relationship between motion vectors for qualitative camera motion classification in mpeg video," in *Proc. of SPIE: Second International Conference on Image and Graphics (ICIG)*, Aug 2002, pp. 853–860.

[4] H. H. A Akutsu, Y. Tonomura and Y. Ohba, "Video indexing using mpeg motion compensated vectors," in *Proc. IEEE international conference on Multimedia computing and Systems*, 1999.

[5] H. S. A. Divakaran, H.Ito and T.Poon, "Scene change detection and feature extraction for mpeg 4 sequences," in *Proc. SPIE Conference on storage and Retrieval for Media Databases 2000*, Jan 2000, pp. 392–398.

[6] D. V.Kobla and C. Faloutsos, "Video trails: representing and visualizing structure in video sequences," in *Proc. ACm multimedia 97*, 1997, pp. 335–346.

[7] I. Koprinska and S. Carrato, "Hybrid rule-based/neural approach for segmentation of mpeg compressed video," in *Multimedia Tools and Applications*, 2000, pp. 187–212.

[8] L. K. Koprinska, "Evolving fuzzy neural network for camera operations recognition," in *pattern Recognition, 2000. Proceedings. 15th International Conference on*, Sept 2000, pp. 523 –526.

[9] H.-I. C. Seok-Woo Jang, Gye-Young Kim, "Detecting shot transitions for video indexing with fam," in *ICANN*, 2001, pp. 1020–1025.

[10] H. Pao and Y. Takefuji, "Functional-link net computing, theory, system architecture, and functionalities," *IEEE Computer*, vol. 3, 1991.

[11] C. L. P. Chen and J. Z. Wan, "A rapid learning and dynamic stepwise updating algorithm for flat neural networks and its application to time-series prediction," *IEEE Trans. on Systems, Man and Cybernetics*, vol. 29, 1999.

[12] J. A. M. A. Sierra and F. Corbacho, "Evolution of functional link networks," *IEEE Trans. Evolutionary Computing*, vol. 5, 2001.

[13] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*. Reading, MA: Addison-Wesley.

[14] M. Russo, "Genetic fuzzy learning," *IEEE Tran. On Evolutionary Computation*, vol. 4(3), pp. 259–273, September 2000.

[15] Z. Michalewicz, *Genetics Algorithms + Data Structures = Evolution Programs*. Reading, Heidelberg: Springer-Verlag, 1996.

[16] P. L. Bartlett, "For valid generalization, the size of the weights is more important than the size of the network," *Advances in Neural Information Processing Systems*, vol. 9, 1997.