Collective Network of Evolutionary Binary Classifiers for Content-Based Image Retrieval

Serkan Kiranyaz, Stefan Uhlmann, Jenni Pulkkinen
and Moncef Gabbouj
Dept. of Signal Processing
Tampere University of Technology
Tampere, Finland
{firstname.lastname}@tut.fi

Turker Ince
Faculty of Computer Science
Izmir University of Economics
Izmir, Turkey
Email: turker.ince@ieu.edu.tr

Abstract—**The content-based image retrieval (CBIR) has been an active research field for which several feature extraction, classification and retrieval techniques have been proposed up to date. However, when the database size grows larger, it is a common fact that the overall retrieval performance significantly deteriorates. In this paper, we propose collective network of (evolutionary) binary classifiers (CNBC) framework to achieve a high retrieval performance even though the training (ground truth) data may not be entirely present from the beginning and thus the system can only be trained incrementally. The CNBC framework basically adopts a "Divide and Conquer" type approach by allocating several networks of binary classifiers (NBCs) to discriminate each class and performs evolutionary search to find the optimal binary classifier (BC) in each NBC. In such an evolution session, the CNBC body can further dynamically adapt itself with each new incoming class/feature set without a full-scale re-training or re-configuration. Both visual and numerical performance evaluations of the proposed framework over benchmark image databases demonstrate its scalability; and a significant performance improvement is achieved over traditional retrieval techniques.**

Keywords- evolutionary classifiers, content-based image retrieval, multi-dimensional particle swarm optimization

I. INTRODUCTION

For content-based image classification and retrieval, the key questions, e.g. 1) how to select certain features so as to achieve highest discrimination over certain classes, 2) how to combine them in the most effective way, 3) which distance metric to apply, 4) how to find the optimal classifier configuration for the classification problem in hand, 5) how to scale/adapt the classifier if large number of classes/features are incrementally introduced and finally, 6) how to train the classifier efficiently to maximize the classification accuracy, still remain unanswered. The current state-of-the-art classifiers such as SVMs, Bayesian, Artificial Neural Networks (ANNs), etc. cannot cope with such requirements since a single classifier, no matter how powerful and well-trained it may be, cannot discriminate efficiently a vast amount of classes, over an indefinitely large set of features. Furthermore, since both classes and features are not static, rather dynamically varying, as a natural consequence of image repositories, static and fixed-structured single classifiers cannot scale such changes without proper configuration updates and a full-scale re-training.

In order to address these problems and hence to maximize the classification accuracy which will in turn boost the retrieval performance, in this paper, we shall focus on a global framework design that embodies a collective networks of evolutionary classifiers. Specifically in this approach, the following objectives will be targeted:

I. *Evolutionary Search*: Seeking for the optimum network architecture among a collection of configurations (the so-called Architecture Space, AS).

II. *Evolutionary Update in the AS*: Keeping only "the best" individual configuration in the AS among indefinite number of evolution runs.

III. *Feature Scalability*: Support for varying number of features. Any feature can be dynamically integrated into the framework without requiring a full-scale initialization and re-evolution.

IV. *Class Scalability*: Support for varying number of classes. Any class can dynamically be inserted into the framework without requiring a re-evolution.

V. *High efficiency* for the evolution (or training) process: Using as compact and simple classifiers as possible in the AS.

VI. *Online (incremental) Evolution*: Continuous online/incremental training (or evolution) sessions can be performed to improve the classification accuracy.

VII. *Parallel processing*: Classifiers can be evolved using several processors working in parallel.

In this way, we shall achieve as compact classifiers as possible, which can be evolved and trained in a much more efficient way than a single but complex classifier, and the optimum classifier for the classification problem in hand can be *searched* with an underlying evolutionary technique, e.g. as in [1]. At a given time, this allows creation and designation of a dedicated classifier for discriminating a certain class type from the others based on a single feature. Each incremental evolution session will "learn" from the current best classifier configurations and can improve them further, possibly as a result of an (incremental) optimization, which may find another configuration in the architecture space (AS) as the "optimal". Moreover, with each incremental evolution, new classes/features can also be introduced which signals the collective classifier network to create new corresponding networks and classifiers within to adapt dynamically to the change. In this way the collective classifier network will be able to dynamically *scale* itself to the indexing requirements of the image database whilst striving for maximizing the classification and retrieval accuracies thanks to the dedicated classifiers within. In order to achieve all these objectives, we adopt a *Divide and Conquer* type of approach, which is based on a novel framework encapsulating a *network of (evolutionary) binary classifiers* (NBCs). Each NBC is devoted to a unique class and further encapsulates a set of *evolutionary Binary Classifiers* (BCs), each of which is optimally chosen within the AS, discriminating the class of the NBC with a unique feature set (or sub-feature). The optimality *therein* can be set with a user-defined criterion. The proposed *Collective NBC* (CNBC) framework currently supports two common ANN types, the Multi-Layer Perceptrons (MLPs) and the Radial Basis Function (RBF) networks. Besides the exhaustive search with the numerous runs of the Back-Propagation method, the recently proposed multi-dimensional Particle Swarm

147

Optimization (MD-PSO) [2], [1] is used as the primary evolution technique. In the current work, due to space limitations we shall restrict only on the CNBC design with evolutionary MLPs.

The rest of the paper is organized as follows. Section II presents evolutionary artificial neural networks. The proposed CNBC framework along with the evolutionary update mechanism is explained in detail in Section III. Section IV provides an extensive set of classification and retrieval results over two benchmark image repositories along with evaluations of the proposed incremental CNBC evolution. Finally, Section V concludes the paper and discusses topics for future work.

II. EVOLUTIONARY NEURAL NETWORKS

In this section we shall discuss the methodology for achieving the first objective that is the evolutionary search for the optimal classifier configuration. First the evolutionary technique, MD-PSO, will be briefly explained and then its application over feed-forward ANNs (the MLPs) shall be introduced.

A. Multi-Dimensional Particle Swarm Optimization

As the evolutionary method, we shall use the multi-dimensional (MD) extension of the basic PSO (bPSO) method, the MD-PSO, recently proposed in [2]. Instead of operating at a fixed dimension N, the MD-PSO algorithm is designed to seek both positional and dimensional optima within a dimension range, $\{D_{min}, D_{max}\}$. In order to accomplish this, each particle has two sets of components, each of which has been subjected to one of the two independent and consecutive processes. The first one is a regular positional PSO, i.e. the traditional velocity updates and due positional shifts in N dimensional search (solution) space. The second one is a dimensional PSO, which allows the particle to navigate through dimensions. Accordingly, each particle keeps track of its last position, velocity and personal best position (pbest) in a particular dimension so that when it re-visits the same dimension at a later time, it can perform its regular "positional" update using this information. The dimensional PSO process of each particle may then move the particle to another dimension where it will remember its positional status and will be updated within the positional PSO process at this dimension, and so on. The swarm, on the other hand, keeps track of the gbest particle in each dimension, indicating the best (global) position so far achieved. Similarly, the dimensional PSO process of each particle uses its personal best dimension in which the personal best fitness score has so far been achieved. Finally, the swarm keeps track of the global best dimension, dbest, among all the personal best dimensions. The gbest particle in the dbest dimension represents the optimum solution and dimension, respectively.

In a MD-PSO process at time (iteration) t, each particle a in the swarm with S particles, $\xi = \{x_1, .., x_a, .., x_S\}$, is represented by the following characteristics:

$xx_{a,j}^{xd_a(t)}(t)$: j^{th} component (dimension) of the position of particle a, in dimension $xd_a(t)$

$vx_{a,j}^{xd_a(t)}(t)$: j^{th} component (dimension) of the velocity of particle a, in dimension $xd_a(t)$

$xy_{a,j}^{xd_a(t)}(t)$: j^{th} component (dimension) of the personal best position of particle a, in dimension $xd_a(t)$

$gbest(d)$: Global best particle index in dimension d

$x\hat{y}_j^d(t)$: j^{th} component (dimension) of the global best position of swarm, in dimension d

$xd_a(t)$: Dimension of particle a

$vd_a(t)$: Dimensional velocity of particle a

$x\tilde{d}_a(t)$: Personal best dimension component of particle a

Let f denote the fitness function that is to be optimized within a certain dimension range, $\{D_{min}, D_{max}\}$. Without loss of generality assume that the objective is to find the minimum of f at the optimum dimension within a multi-dimensional search space. Assume that the particle a visits (back) the same dimension after T iterations (i.e. $xd_a(t) = xd_a(t+T)$), then the personal best position can be updated in iteration $t+T$ as follows,

$$xy_{a,j}^{xd_a(t+T)}(t+T) = \begin{cases} xy_{a,j}^{xd_a(t)}(t) & if \ f(xx_a^{xd_a(t+T)}(t+T)) > f(xy_a^{xd_a(t)}(t)) \\ xx_{a,j}^{xd_a(t+T)}(t+T) & else \end{cases} \quad (1)$$

$$j = 1, 2, ..., xd_a(t+T)$$

Furthermore, the personal best dimension of particle a can be updated in iteration $t+1$ as follows,

$$x\tilde{d}_a(t+1) = \begin{cases} x\tilde{d}_a(t) & if \ f(xx_a^{xd_a(t+1)}(t+1)) > f(xy_a^{x\tilde{d}_a(t)}(t)) \\ xd_a(t+1) & else \end{cases} \quad (2)$$
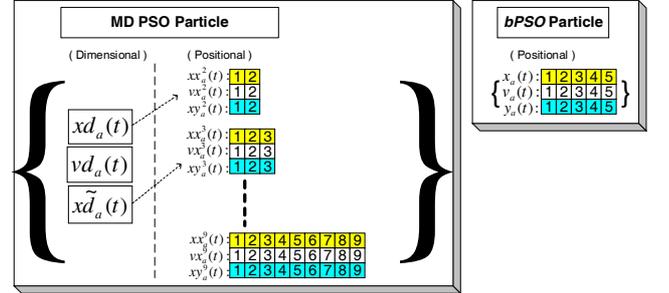

Figure 1: Sample MD-PSO (left) vs. *bPSO* (right) particle structures. For MD-PSO $\{D_{min} = 2, D_{max} = 9\}$ and at time t, $xd_a(t) = 2$ and $x\tilde{d}_a(t) = 3$.

Figure 1 shows sample MD-PSO and *bPSO* particles denoted as a. Particle a in *bPSO* particle is at a (fixed) dimension, $N=5$, and contains only positional components; whereas in MD-PSO particle a contains both positional and dimensional components, respectively. In the figure the dimension range for MD-PSO is given by $\{D_{min}, D_{max}\} = \{2, 9\}$, therefore the particle contains 9 sets of positional components. In this example the particle a currently resides at dimension 2 ($xd_a(t) = 2$); whereas its personal best dimension is 3 ($x\tilde{d}_a(t) = 3$). Therefore, at time t a positional PSO update is first performed over the positional elements, $xx_a^2(t)$ and then the particle may move to another dimension with respect to the dimensional PSO. Recall that each positional element, $xx_a^2(t)$, represents a potential solution in the search space of the problem.

B. MD-PSO for Evolving MLPs

As a stochastic search process in multi-dimensional search space, MD-PSO seeks (near-) optimal networks in an architecture space (AS), which can be defined over any type of ANNs with any properties. All network configurations in the AS are enumerated into a hash table with a proper hash function, which ranks the networks with respect to their complexity, i.e. associates higher hash indices to networks with higher complexity. MD-PSO can then use each index as a unique dimension of the search space where particles can make inter-dimensional navigations to seek an optimum dimension (dbest) and the optimum solution on that dimension, $x\hat{y}^{dbest}$. The former

corresponds to the optimal architecture and the latter encapsulates the optimum network parameters (connections, weights and biases). Suppose for the sake of simplicity, a range is defined for the minimum and maximum number of layers, $\{L_{min}, L_{max}\}$ and number of neurons for the hidden layer l, $\{N_{min}^l, N_{max}^l\}$. The sizes of both input and output layers are determined by the problem and hence fixed. The AS can then be defined only by two range arrays, $R_{min} = \{N_i, N_{min}^1, ..., N_{min}^{L_{max}-1}, N_o\}$ and $R_{max} = \{N_i, N_{max}^1, ..., N_{max}^{L_{max}-1}, N_o\}$, one for minimum and the other for the maximum number of neurons allowed for each layer of a MLP. The size of both arrays is naturally $L_{max}+1$ where the corresponding entries define the range of the lth hidden layer for all those MLPs, which can have an lth hidden layer. The size of the input and output layers, $\{N_i, N_o\}$, is fixed and is the same for all configurations in the AS. $L_{min} \geq 1$ and L_{max} can be set to any value meaningful for the problem encountered. The hash function then enumerates all potential MLP configurations into hash indices, starting from the simplest MLP with $L_{min}-1$ hidden layers, each of which has minimum number of neurons given by R_{min}, to the most complex network with $L_{max}-1$ hidden layers, each of which has a maximum number of neurons given by R_{max}.

Let N_h^l be the number of hidden neurons in layer l of a MLP with input and output layer sizes N_i and N_o, respectively. The input neurons are merely fan-out units since no processing takes place. Let F be the activation function applied over the weighted inputs plus a bias, as follows:

$$y_k^{p,l} = F(s_k^{p,l}) \quad where \quad s_k^{p,l} = \sum_j w_{jk}^{l-1} y_j^{p,l-1} + \theta_k^l \qquad (3)$$

where $y_k^{p,l}$ is the output of the kth neuron of the lth hidden/output layer when the pattern p is fed, w_{jk}^{l-1} is the weight from the jth neuron in layer l-1 to the kth neuron in layer l, and θ_k^l is the bias value of the kth neuron of the lth hidden/output layer. The training mean square error, MSE, is formulated as follows:

$$MSE = \frac{1}{2PN_o} \sum_{p \in T} \sum_{k=1}^{N_o} (t_k^p - y_k^{p,o})^2 \qquad (4)$$

where t_k^p is the target (desired) output and $y_k^{p,o}$ is the actual output from the kth neuron in the output layer, $l=o$, for pattern p in the training dataset T with size P, respectively. At a time t, suppose that particle a, has the positional component formed as, $xx_a^{xd_a(t)}(t) = \Psi^{xd_a(t)} \left\{ \{w_{jk}^0\}, \{w_{jk}^1\}, \{\theta_k^1\}, \{w_{jk}^2\}, \{\theta_k^2\}, ..., \{w_{jk}^{o-1}\}, \{\theta_k^{o-1}\}, \{\theta_k^o\} \right\}$ where $\{w_{jk}^l\}$ and $\{\theta_k^l\}$ represent the sets of weights and biases of the layer l of the MLP configuration, $\Psi^{xd_a(t)}$. Note that the input layer ($l=0$) contains only weights whereas the output layer ($l=o$) has only biases. By means of such a direct encoding scheme, particle a represents all potential network parameters of the MLP architecture at the dimension (hash index) $xd_a(t)$. As mentioned earlier, the dimension range, $\{D_{min}, D_{max}\}$, where MD-PSO particles can make inter-dimensional jumps, is determined by the AS defined. Apart from the regular limits such as (positional) velocity range, $\{V_{min}, V_{max}\}$, dimensional velocity range, $\{VD_{min}, VD_{max}\}$, the data

space can also be limited with some practical range, i.e. $X_{min} < xx_a^{xd_a(t)}(t) < X_{max}$. Setting MSE in Eq. (4) as the fitness function enables MD-PSO to perform evolutions of both network parameters and architectures within its native process.

Further details and an extensive set of experiments demonstrating the optimality of the networks evolved with respect to several benchmark problems can be found in [1].

III. THE CNBC FRAMEWORK

This section describes in detail the proposed framework: Collective Network of (Evolutionary) Binary Classifiers, the CNBC, which uses the training dataset to configure its internal structure and to evolve its binary classifiers (BCs) individually. Before going into details of CNBC, the evolutionary update mechanism, which keeps only the best networks within the AS of each BC will be detailed next.

A. Evolutionary Update in the Architecture Space

Since the evolutionary technique, MD PSO, is a stochastic optimization method, in order to improve the probability of convergence to the global optimum, several evolutionary runs can be performed. Let N_R be the number of runs and N_C be the number of configurations in the AS. For each run the objective is to find the optimal (the best) classifier within the AS with respect to a pre-defined criterion. Note that along with the best classifier, all other configurations in the AS are also subject to evolution and therefore, they are continuously (re-) trained with each run. So during this ongoing process, between any two consecutive runs, any network configuration can replace the current best one in the AS if it surpasses it. Besides the MD PSO evolutionary search, this is also true for the *exhaustive search*, i.e. each network configuration in the AS is trained by N_R BP runs and the same evolutionary update rule applies. Figure 2 demonstrates an evolutionary update operation over a sample AS containing 5 MLP configurations. The table shows the training MSE which is the criterion used to select the optimal configuration at each run. The best runs for each configurations are highlighted and the best configuration in each run is tagged with '*'. Note that at the end of the three runs, the overall best network with MSE = 0.10 has the configuration: 15x2x2 and thus used as the classifier for any classification task until any other configuration surpasses it in a future run. In this way, each BC configuration in the AS can only *evolve* to a better state, which is the main purpose of the proposed evolutionary update mechanism.
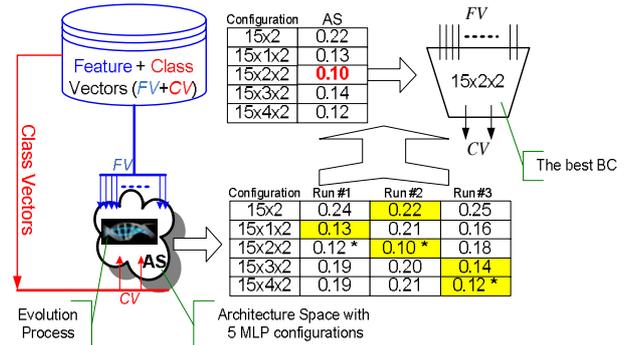

Figure 2: Evolutionary update in a sample AS for MLP configuration arrays $R_{min} = \{15,1,2\}$ and $R_{max} = \{15,4,2\}$ where $N_R = 3$ and $N_C = 5$. The best run for each configurations is highlighted and the best configuration in each run is tagged with '*'.

B. Collective Network of Binary Classifiers

1) The Topology

To achieve the third and fourth objectives mentioned earlier, i.e. the scalability with respect to a varying number of classes and features, a novel framework encapsulating a *network of binary classifiers* (NBCs) is developed, where NBCs can *evolve* continuously with the ongoing evolution sessions i.e. using the training dataset which is in practice obtained by cumulating the ground truth data (GTD) from several relevance feedback sessions. Each NBC corresponds to a *unique* image class and shall contain varying number of evolutionary binary classifiers (BCs) in the input layer where each BC performs binary classification using a single (sub-) feature. Therefore, whenever a new feature is extracted, its corresponding BC will be created, evolved (using the available GTD so far), and inserted into each NBC, yet keeping each of the other BCs "as is". On the other hand, whenever an existing feature is removed, the corresponding BC is simply removed from each NBC in the system. In this way *scalability* with respect to any number of features is achieved and the overall system can avoid re-evolutions from scratch.
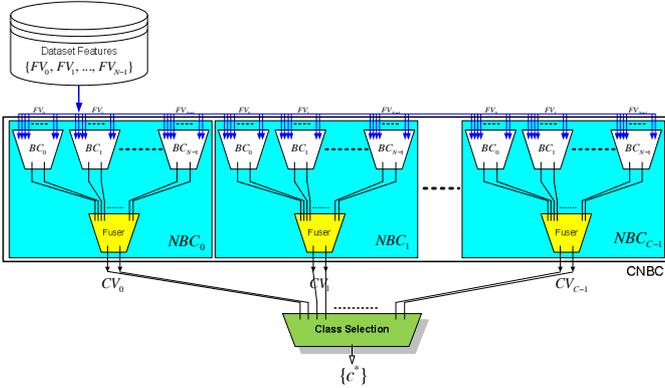

Figure 3: Topology of the proposed CNBC framework with *C* classes and *N* FVs (sub-features).

Each NBC has a "fuser" BC in the output layer, which collects and fuses the binary outputs of all BCs in the input layer and generates a single binary output, indicating the relevancy of each FV to the NBC's corresponding class.

Furthermore, CNBC is also *scalable* to any number of classes since whenever a new class is defined by the user, a new NBC can simply be created (and evolved) only for this class without requiring any need for change or update the other NBCs unless their performance significantly deteriorates with the introduction of the new class This way the overall system dynamically adapts to varying number of image classes.

As shown in Figure 3, the main idea in this approach is to use as large number of classifiers as necessary, so as to divide a massive learning problem into many NBC units along with the BCs within, and thus prevent the need of using complex classifiers as the performance of both training and evolution processes degrades significantly as the complexity rises due to the *curse of dimensionality*. A major benefit of our approach with respect to efficient training and evolution process is that the configurations in the AS can be kept as *compact* as possible avoiding unfeasibly large storage and training time requirements. This is a significant advantage especially for the training methods performing local search, such as BP since the amount of deceiving local minima is significantly low in the error space for such simple and compact ANNs. Furthermore,

when BP is applied exhaustively, the probability of finding the optimum solution is significantly increased.

In order to maximize the classification accuracy, we applied a dedicated class selection technique for CNBC. We used *1-of-n* encoding scheme in all BCs, and the output layer size of all BCs is always two (i.e. $n = 2$). Let $CV_{c,1}$ and $CV_{c,2}$ be the first and second output of the cth BC's class vector (CV). The class selection in *1-of-n* encoding scheme can simply be performed by comparing the individual outputs, e.g. say a positive output if $CV_{c,2} > CV_{c,1}$, and vice versa for negative. This is also true for the fuser BC, the output of which makes the output of its NBC. FVs of each dataset item are fed to each NBC in the CNBC. Each FV is propagated through its corresponding BC in the input layer of the NBC. The outputs of these BCs are then fed to the fuser BC of each NBC to produce all CVs. Finally, the class selection block shown in Figure 3 collects them and selects the positive class(es) of the CNBC as the final outcome. This selection scheme, first of all, differs with respect to the dataset class type, i.e. the dataset can be called as "uni-class", if an item in the dataset can belong to *only* one class, otherwise called as "multi-class". Therefore, in a uni-class dataset there must be *only* one class, the c^*, selected as the positive outcome whereas in a multi-class dataset, there can be one or more NBCs, $\{c^*\}$, with a positive outcome. In the class selection scheme the *winner-takes-all* strategy is utilized. Therefore, for uni-class datasets, the positive class index, c^*, ("the winner") is determined to be the class where the difference $CV_{c,2} - CV_{c,1}$ is maximal, i.e.,

$$c^* = \arg \max_{c \in [0, C-1]} (CV_{c,2} - CV_{c,1}) \qquad (5)$$

In this way the erroneous cases (false negative and false positives) where none or more than one NBC exists with a positive outcome can be properly handled. However, for multi-class datasets the winner takes all strategy can only be applied when no NBC yields a positive outcome, i.e. $CV_{c,2} \leq CV_{c,1} \ \forall c \in [0, C-1]$, otherwise multiple NBCs with positive outcome may indicate the multiple true-positives and hence cannot be further pruned. As a result, for a multi-class dataset the (set of) positive class indices, $\{c^*\}$, is selected as follows:

$$\{c^*\} = \begin{cases} \arg \max_{c \in [0,C-1]} (CV_{c,2} - CV_{c,1}) & if \ CV_{c,2} \leq CV_{c,1} \ \forall c \in [0,C-1] \\ \{\arg_{c \in [0,C-1]} (CV_{c,2} > CV_{c,1})\} & else \end{cases} \qquad (6)$$

2) Evolution of the CNBC

The evolution of a subset of the NBCs or the entire CNBC is performed for each NBC individually with a two-phase operation, as illustrated in Figure 4. As explained earlier, using the feature vectors (FVs) and the target class vectors (CVs) of the training dataset, the evolution process of each BC in a NBC is performed within the current architecture space (AS) in order to find the best (optimal) BC configuration with respect to a given criterion (e.g. training/validation mean-square-error (MSE) or classification error (CE)). During the evolution, only NBCs associated with those classes represented in the training dataset are evolved. If the training dataset contains new classes, which do not have a corresponding NBC yet, a new NBC is created for each, and evolved using the training dataset.

In Phase 1, see top of Figure 4, the BCs of each NBC are evolved given an input set of FVs and a target CV. Recall that each CV is associated with a unique NBC. The fuser BCs are not used in

this phase. Once an evolution session is over, the AS of each BC is then recorded so as to be used for potential (incremental) evolution sessions in the future. Recall that each evolution process may contain several runs and according to the aforementioned evolutionary update rule, the best configuration achieved will be used as the classifier. Hence once the evolution process is completed for all BCs in the input layer (phase 1), the best BC configurations are used to forward propagate all FVs of the items in the training dataset to compose the FV for the fuser BC from their output CVs, so as to evolve the fuser BC in the second phase. Apart from the difference in the generation of the FVs, the evolutionary method (and update) of the fuser BC is same as any other BC has in the input layer. In this phase, the fuser BC *learns* the *significance* of each individual BC (and the corresponding sub-feature) for the discrimination of that particular class. This can be viewed as the adaptation of the entire feature space to discriminate a specific class in a large dataset, or in other words, as a way of applying an efficient *feature selection* scheme as some FVs may be quite discriminative for some classes whereas others may not and the fuser, if properly evolved and trained, can "weight" each BC (with its FV), accordingly. In this way the usage of each feature (and its BC) shall optimally be "fused" according to their discrimination power of each class. Similarly, each BC in the first layer shall in time learn the significance of individual feature components of the corresponding FV for the discrimination of its class. In short the CNBC, if properly evolved, shall learn the significance (or the discrimination power) of each FV and its individual components.
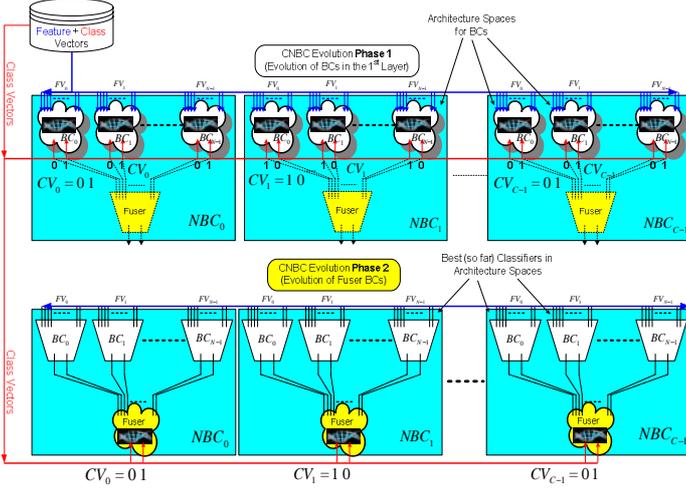

Figure 4: Illustration of the two-phase evolution session over BCs' architecture spaces in each NBC.

3)   Incremental Evolution of the CNBC

To accomplish yet another major objective, the proposed CNBC framework is designed for continuous "incremental" evolution sessions where each session may further improve the classification performance of each BC using the advantage of the "evolutionary updates". The main difference between the initial and the subsequent incremental evolution sessions is the initialization of the evolution process: the former uses *random* initialization for each configuration in the AS whereas the latter starts from the best parameters found for each classifier configuration in the last AS for each BC. Note that the training dataset used for the incremental evolution sessions may be different from the previous ones, and each session may contain several runs. Thus the evolutionary update rule compares the performance of the last recorded and the *current* (after the run) network over the *current* training dataset. During each incremental evolution phase, existing NBCs are (incrementally) evolved *only if*

they cannot accurately classify the training dataset of the new (emerging) classes. In that, an empirical threshold level (e.g. 95%) is used to determine the level of classification accuracy required. The NBCs for the new classes are obviously due for evolution without any such verification.

Consequently, the proposed MD PSO evolutionary technique used for evolving MLP configurations is initialized with the current AS parameters of the network. That is the swarm particles are randomly initialized (as in the initial evolutionary step) except that one of the particles (without loss of generality we assume the first particle with $a=0$) has its personal best set to the optimal solution found in the previous evolutionary session. For MD PSO evolution over MLPs, this can be expressed as,

$$xy_0^d(0) \leftarrow \Psi^d \left\{ \{w_{jk}^0\}, \{w_{jk}^1\}, \{\theta_k^1\}, \{w_{jk}^2\}, \{\theta_k^2\}, ..., \{w_{jk}^{O-1}\}, \{\theta_k^{O-1}\}, \{\theta_k^O\} \right\}$$
$$\forall d \in [1, N_C] \tag{7}$$

where $\{w_{jk}^l\}$ and $\{\theta_k^l\}$ represent the sets of weights and biases of the layer l of the MLP network, Ψ^d, which is the dth (MLP) configuration retrieved from the last AS record. It is expected that especially at the early stages of the MD PSO run, the first particle is likely to be the *gbest* particle in every dimension, guiding the swarm towards the last solution otherwise keeping the process independent and unconstrained. Particularly if the training dataset is considerably different in the incremental evolution sessions, it is quite probable that MD PSO can converge to a new solution whilst taking the past solution (experience) into account.

In the alternative evolutionary technique, the exhaustive search via repetitive BP training of each network in the AS, the first step of an incremental training will simply be the initialization of the weights w_{jk}^l and biases θ_k^l with the parameters retrieved from the last AS of that BC. Starting from this as the initial point, and using the current training dataset with the target CVs, the BP algorithm can then perform its gradient descent in the error space to converge to a new solution.

IV.   EXPERIMENTAL RESULTS

In this section, we first detail the benchmark datasets used and the feature extraction techniques performed for the extensive set of classification and content-based image retrieval experiments. We then investigate the classification performance of the proposed CNBC framework using different feature combinations and we also compare the results obtained with batch training and incremental evolutions. Finally we shall demonstrate the performance gain in terms of improved retrieval accuracy that can be achieved using the proposed CNBC framework as compared to the traditional (dis-) similarity based retrievals.

A.   Database Creation and Feature Extraction

We used MUVIS framework [4], to create and to index the following two image databases by extracting 14 (sub-) features for each.

1)   *Corel_10* Image Database: There are 1000 medium resolution (384x256 pixels) images obtained from Corel repository [5] covering 10 diverse classes: 1 - *Natives*, 2 - *Beach*, 3 - *Architecture*, 4 - *Bus*, 5 - *Dino Art*, 6 - *Elephant*, 7 - *Flower*, 8 - *Horse*, 9 - *Mountain*, and 10 - *Food*.

2)   *Corel_Caltech_30* Image Database: There are 4245 images from 30 diverse classes that are obtained from both Corel and Caltech [6] image repositories.

As detailed in Table 1, some of the basic color (e.g. MPEG-7 Dominant Color Descriptor, HSV color histogram and Color Structure Descriptor [7]), texture (e.g. Gabor [8], Local Binary Pattern [9], and Ordinal Co-occurrence Matrix [10]) and edge (e.g. Edge Histogram Direction [7]) features, are extracted. Some of them are created with different parameters to extract several sub-features and the total feature dimension is obtained as 2335. Such a high feature space dimension can thus give us opportunity to test the performance of the proposed CNBC framework against the "curse of dimensionality" and *scalability* with the varying number of features.

Table 1: 14 Features extracted per MUVIS database.

FV	Feature	Parameters	Dim.
1	HSV Color Histogram	H=6, S=2, V=2	24
2		H=8, S=4, V=4	128
3	Dominant Color	$N_{DC}^{max}=6, T_A=2\%, T_S=15$	27
4		$N_{DC}^{max}=8, T_A=2\%, T_S=15$	35
5	Color Structure	32 bins	32
6		64 bins	64
7		128 bins	128
8		256 bins	256
9		512 bins	512
10		1024 bins	1024
11	Local Binary Pattern		16
12	Gabor	scale=4, orient.=6	48
13	Ordinal Co-occurrence	d=3, o=4	36
14	Edge Histogram Dir.		5

B. Classification Results

Both databases are partitioned in such a way that the majority (55%) of the items is spared for testing and the rest was used for evolving the CNBC. To demonstrate the feature scalability property of the CNBC, we evolved two CNBCs individually using 7 (FVs 1, 3, 5, 11, 12, 13 and 14 in Table 1 with total dimension of 188) and 14 (all FVs with total dimension of 2335) features. Therefore, the first CNBC has 7+1=8 BCs and the second has 14+1=15 BCs in each NBC. The evolution (and training) parameters are as follows: For MD-PSO, we use the termination criteria as the combination of the maximum number of iterations allowed (*iterNo* = 100) and the cut-off error ($\varepsilon_C = 10^{-4}$). Other parameters were empirically set as: the swarm size, S=50, $V_{max}=x_{max}/5=0.2$ and $VD_{max}=5$, respectively. For exhaustive BP, the learning parameter is set as $\lambda = 0.01$ and iteration number is 20. We use the typical activation function: *hyperbolic tangent* ($\tanh(x)=\dfrac{e^x-e^{-x}}{e^x+e^{-x}}$). For the AS, we used simple configurations with the following range arrays: $R_{min}=\{N_i, 8, 2\}$ and $R_{max}=\{N_i, 16, 2\}$, which indicate that besides the single layer perceptron (SLP), all MLPs have only a single hidden layer, i.e. $L_{max}=2$, with no more than 16 hidden neurons. Besides the SLP, the hash function enumerates all MLP configurations in the AS, as shown in Table 2. Finally, for both evolution methods, the exhaustive BP and MD PSO, $N_R = 10$ independent runs are performed. Note that for exhaustive BP, this corresponds to 10 runs for each configuration in the AS.

Table 3 presents the classification performances achieved for *Corel_10* database by both evolutionary techniques over the sample AS given in Table 2. The results indicate that MD PSO achieves the lowest MSE and CE levels (and hence the best results)

within the training set whereas vice versa is true for the exhaustive BP within the test set. CNBC in general demonstrates a solid robustness against the major feature dimension increase (i.e. from 7 (188-D) to 14 sub-features (2335-D)) since the classification performance does not show any deterioration, on contrary, with both techniques a better performance is achieved with enhanced generalization ability. This is an expected outcome since CNBC can benefit from the additional discrimination capability of each incoming (sub-) feature thanks to its "Divide and Conquer" type design where an efficient feature selection scheme is embedded.

Table 2: The architecture space used for MLPs.

Dim.	*Conf.*	Dim.	*Conf.*	Dim.	*Conf.*
0	N_i x 2	6	N_i x6x2	12	N_i x12x2
1	N_i x1x2	7	N_i x7x2	13	N_i x13x2
2	N_i x2x2	8	N_i x8x2	14	N_i x14x2
3	N_i x3x2	9	N_i x9x2	15	N_i x15x2
4	N_i x4x2	10	N_i x10x2	16	N_i x16x2
5	N_i x5x2	11	N_i x11x2		

Table 3: Classification performance of each evolution method per feature set for *Corel_10* database.

Feature Set	Evol. Method	Train MSE	Train CE %	Test MSE	Test CE %
7 sub-features	**MDPSO**	0.32	3.55	1.61	20.18
	BP	0.45	4.88	**1.27**	**18.36**
14 sub-features	**MDPSO**	0.28	3.18	1.46	18.63
	BP	0.34	3.33	**1.25**	**14.54**

Table 4 presents the confusion matrix of the best classification result over the test set, i.e. achieved by the exhaustive BP method using 14 sub-features. It is worth noting that the major source of error results from the confusion between the 2nd (*Beach*) and 9th (*Mountain*) classes where low-level features cannot really discriminate due to excessive color and texture similarities among those classes. This is also true for the 6th class (*Elephant*) from which the background of some images share a high similarity with both classes.

Table 4: Confusion matrix of the evolution method, which gave the best (lowest) test CE in Table 3.

Actual		1	2	3	4	5	6	7	8	9	10
	1	42	2	1	1	0	5	0	0	1	3
	2	2	37	4	1	0	0	0	1	9	1
	3	2	3	46	1	0	1	0	0	1	1
	4	2	0	0	53	0	0	0	0	0	0
	5	0	0	0	0	55	0	0	0	0	0
Truth	6	2	4	2	0	0	37	0	1	8	1
	7	1	0	0	0	0	0	53	1	0	0
	8	0	0	0	0	0	0	0	55	0	0
	9	0	8	1	0	0	0	0	0	46	0
	10	1	1	0	1	1	2	1	0	2	46

The CNBC evolutions so far performed are much alike to the (batch) training of traditional classifiers (such as ANNs, k-NN, Bayesian) where the training data (the features) and (number of) classes are all fixed and the entire GTD is used during the training (evolution). As detailed earlier, the CNBC can also be evolved incrementally, i.e. incremental evolutions can be performed whenever

new features/classes can be introduced and the CNBC can dynamically create new BCs and/or NBCs as the need arises. In order to evaluate the incremental evolution performance, the training dataset is divided into three distinct partitions, each of which contains 5 (classes 1-5), 3 (classes 6-8) and 2 (classes 9 and 10) classes, respectively. Therefore, three stages of incremental evolutions have been performed where at each stage the CNBC is further evolved only with the particular dataset partition, which belongs to the new classes in that partition. After the first phase, only three out of five existing NBCs were incrementally evolved over the training dataset of the three new classes (classes 6-8). Similarly, at the third phase, three out of 8 NBCs did not undergo for incremental evolution since their classification accuracy over the training dataset of those new classes (9 and 10) are already above the minimum classification accuracy threshold (95%) required.

Due to space limitations, we had to skip details on the classification performances achieved at the intermediate stages. Table 5 presents the final classification performance of each evolution method per feature set. The results indicate a few percent losses on both training and test classification accuracies, which can be expected since the incremental evolution was purposefully skipped for some NBCs whenever they surpass 95% classification accuracy over the training dataset of the new (emerging) classes. This means, for instance, some NBCs (e.g. the one corresponds to class 4, the *Bus*) evolved with only over a subset of the entire training dataset.

Table 5: Final classification performance of 3-stage incremental evolution per evolution method and feature set for *Corel_10* database.

Feature Set	Evol. Method	Train MSE	Train CE %	Test MSE	Test CE %
7 sub-features	MDPSO	1.36	6.89	2.61	28.63
	BP	**0.82**	**4.22**	**1.85**	**21.63**
14 sub-features	MDPSO	1.23	**6.66**	2.39	26.36
	BP	**0.91**	7.55	**1.83**	23.81

Finally, the CNBC evolution for *Corel_Caltech_30* database allows testing and evaluation of its classification performance when the database size and number of classes are significantly increased. For both evolution techniques, we used the same parameters as presented earlier except that the number of epochs (iterations) for BP and MD PSO were increased to 200 and 500 in order to compensate the increase in the database size. Table 6 presents the classification performances of each evolution method per feature set. Due to space limitations we had to skip the results from incremental evolutions in this dataset. As compared with the results from *Corel_10* database in Table 3, it is evident that both evolution methods achieved a similar classification performance in the training set (i.e. similar train CEs) whilst certain degradation occurs in the classification accuracy in test set (i.e. 10-15% increase in the test CEs). This is an expected outcome since the lack of discrimination within those low-level features can eventually yield a poorer generalization especially when the number of classes is tripled.

Table 6: Classification performance of each evolution method per feature set for *Corel_Caltech_30* database.

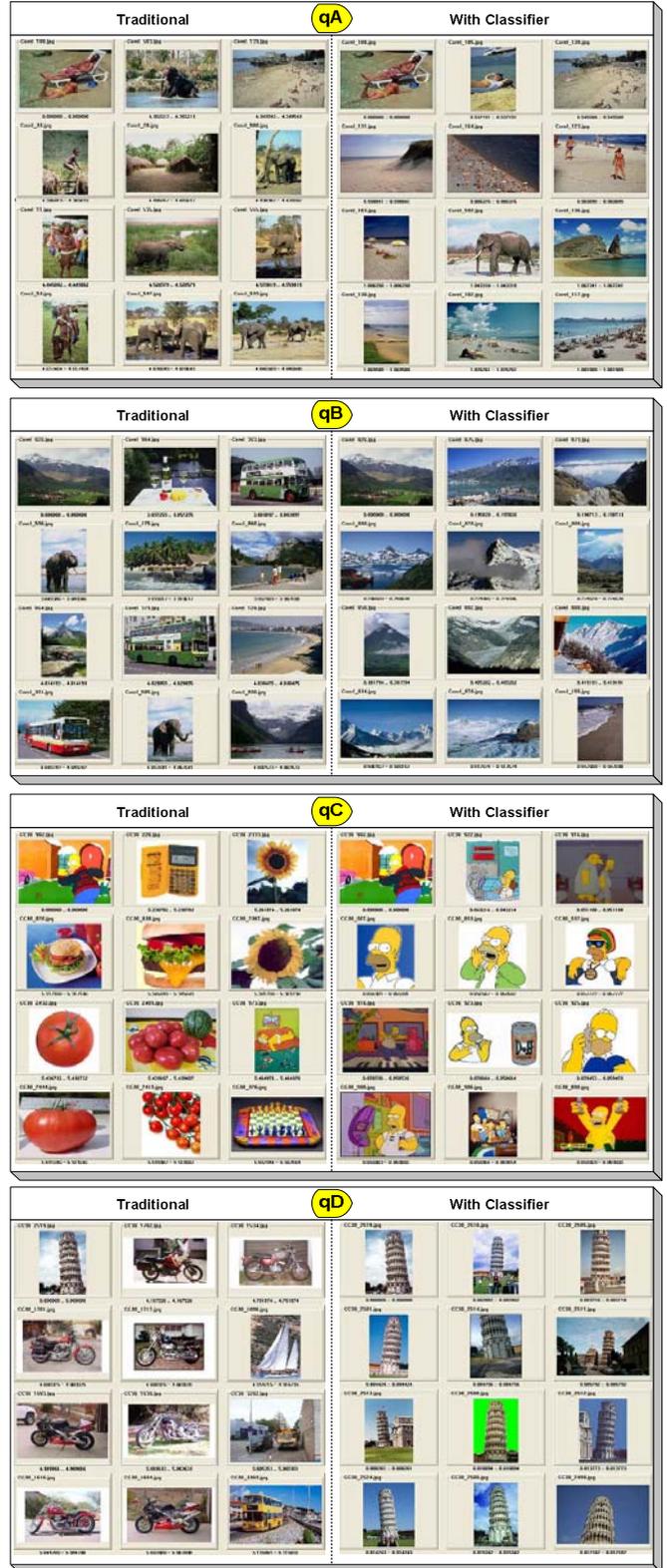| Feature Set	Evol. Method	Train MSE	Train CE	Test MSE	Test CE
7 sub-features	MD PSO	0.54	8.1	2.3	**33.40**
	BP	**0.24**	**2.95**	**2.16**	34.67
14 sub-features	MD PSO	0.33	5.47	**2.52**	36.33
	BP	**0.074**	**1.31**	2.69	**33.86**

Figure 5: 4x2 sample queries in *Corel_10* (qA and qB), and *Corel_Caltech_30* (qC and qD) databases Top-left is the query image.

C. Retrieval Results

The traditional retrieval process in MUVIS is based on the query by example (QBE) operation. The (sub-) features of the query item are used for (dis-) similarity measurement among all the features of the visual items in the database. Ranking the database items according to their similarity distances yields the retrieval result. The traditional (dis-) similarity measurement in MUVIS is accomplished by applying a distance metric such as L2 (*Euclidean*) between the feature vectors of the query and each database item. When a CNBC is used for the purpose of retrieval, the same (L2) distance metric is now applied to the class vectors at the output layer of the CNBC (10x2=20-D for *Corel_10* and 30x2=60-D for *Corel_Caltech_30* databases). In order to evaluate the retrieval performances with and without CNBC, we use average precision (*AP*) and average normalized modified retrieval rank (ANMRR) measures, both of which are computed querying *all* images in the database (i.e. batch query) and within a retrieval window equal to the number of ground truth images, $N(q)$ for each query q. This henceforth makes the *AP* identical to average recall and average F1 measures, too.

Over each database, four batch queries are performed to compute the average retrieval performances, two with and two without using the CNBC. Whenever used, the CNBC is evolved with the MD PSO and the exhaustive BP, the former with 7 and the latter with 14 sub-features, respectively. As listed in Table 7, it is evident that the CNBC can significantly enhance the retrieval performance regardless of the evolution method, the feature set and the database size. The results (without CNBC) in the table also confirm the enhanced discrimination obtained from the larger feature set, which led to better classification performance and in turn, leads to a better retrieval performance.

Table 7: Retrieval performances (%) of the four batch queries in each MUVIS databases.

Feature Set	Evol. Method	Corel_10		Corel_Caltech_30	
		ANMRR	AP	ANMRR	AP
7 sub-features	MD PSO	33.09	64.01	43.04	54.47
	None	55.81	42.15	60.21	37.80
14 sub-features	BP	22.21	76.20	32.00	65.37
	None	47.19	50.38	62.94	34.92

For visual evaluation, Figure 5 presents four typical retrieval results with and without using the proposed CNBC framework. All query images are selected among the test set and the query is processed within the entire database.

V. CONCLUSIONS

In this paper, a novel CNBC framework is introduced to address the problem of efficient and accurate content-based classification and retrieval within large image databases. CNBC is a *Divide and Conquer* type of approach, which reduces both feature and class vector dimensions for individual classifiers significantly to enable the use of as compact classifiers as possible. Such compact classifiers can be evolved and trained better than a single yet more complex classifier.

The optimum classifier for each classification problem at hand can be searched separately and at a given time, this allows to create new dedicated classifiers (BCs) for discriminating a certain class type from the others with the use of a single (sub-) feature to accommodate new features or to create a new NBC to allow introduction of new classes. Each (incremental) evolution session "learns" from the current best classifier and can improve it further, possibly using another configuration in the AS. Moreover, when trained properly, the fuser BC can correct the erroneous classification of any BC in the input layer, which further increases the classification accuracy. Thus classification performance, the main advantages of the proposed framework are the improved classification performance and the efficient solution provided to the problems of *scalability* and *dynamic adaptability* by allowing both feature space dimensions and the number of classes in a database to be unlimited and dynamic (incremental). Furthermore, the CNBC framework is designed for both online (incremental) and offline (batch) evolutions, which can be performed in multiple runs. During each run, any new configuration can replace the current one in the AS if it outperforms it. Such an evolutionary update mechanism ensures that the AS containing the best configurations, is always kept intact and that only the best configuration at any given time is used for classification and retrieval.

Although the results indicate that all the aforementioned objectives have been successfully fulfilled, even higher accuracy levels can still be expected from the CNBC framework with the addition of new powerful features with superior discrimination and content description capabilities.

REFERENCES

[1] S. Kiranyaz, T. Ince, A. Yildirim and M. Gabbouj, "Evolutionary Artificial Neural Networks by Multi-Dimensional Particle Swarm Optimization", Neural Networks, vol. 22, pp. 1448 – 1462, doi:10.1016/j.neunet.2009.05.013, Dec. 2009.

[2] S. Kiranyaz, T. Ince, A. Yildirim and M. Gabbouj, "Fractional Particle Swarm Optimization in Multi-Dimensional Search Space", *IEEE Trans. on Systems, Man, and Cybernetics – Part B*, pp. 298 – 319, vol. 40, No. 2, 2010.

[3] Y. Chauvin and D. E. Rumelhart, "Back Propagation: Theory, Architectures, and Applications," Lawrence Erlbaum Associates Publishers, UK, 1995.

[4] MUVIS. http://muvis.cs.tut.fi/

[5] Corel Collection / Photo CD Collection (www.corel.com)

[6] L. Fei-Fei, R. Fergus and P. Perona, "Learning generative visual models from few training examples: an incremental Bayesian approach tested on 101 object categories", *IEEE CVPR Workshop on Generative-Model Based Vision*, vol. 12, pp.178, 2004.

[7] B. S. Manjunath, J.-R. Ohm, V. V. Vasudevan, and A. Yamada, "Color and Texture Descriptors", *IEEE Trans. On Circuits and Systems for Video Technology*, vol. 11, pp. 703-715, Jun. 2001.

[8] B. Manjunath, P. Wu, S. Newsam, H. Shin, "A texture descriptor for browsing and similarity retrieval", Journal of Signal Processing: Image Communication, vol. 16, pp. 33-43, Sep. 2000.

[9] T. Ojala, M. Pietikainen, D. Harwood, "A comparative study of texture measures with classification based on feature distributions", Pattern Recognition, vol. 29, pp. 51-59, 1996.

[10] M. Partio, B. Cramariuc, M. Gabbouj, "An Ordinal Co-occurrence Matrix Framework for Texture Retrieval", EURASIP Journal on Image and Video Processing, vol. 2007, Article ID 17358, 15 pages, 2007. doi:10.1155/2007/17358.