Selection of an optimal neural network architecture for computer-aided detection of microcalcifications—Comparison of automated optimization techniques

Metin N. Gurcan,[a] Berkman Sahiner, Heang-Ping Chan, Lubomir Hadjiiski, and Nicholas Petrick
Department of Radiology, University of Michigan, Ann Arbor, Michigan 48109

(Received 27 March 2001; accepted for publication 26 June 2001)

Many computer-aided diagnosis (CAD) systems use neural networks (NNs) for either detection or classification of abnormalities. Currently, most NNs are ‘‘optimized’’ by manual search in a very limited parameter space. In this work, we evaluated the use of automated optimization methods for selecting an optimal convolution neural network (CNN) architecture. Three automated methods, the steepest descent (SD), the simulated annealing (SA), and the genetic algorithm (GA), were compared. We used as an example the CNN that classifies true and false microcalcifications detected on digitized mammograms by a prescreening algorithm. Four parameters of the CNN architecture were considered for optimization, the numbers of node groups and the filter kernel sizes in the first and second hidden layers, resulting in a search space of 432 possible architectures. The area A_z under the receiver operating characteristic (ROC) curve was used to design a cost function. The SA experiments were conducted with four different annealing schedules. Three different parent selection methods were compared for the GA experiments. An available data set was split into two groups with approximately equal number of samples. By using the two groups alternately for training and testing, two different cost surfaces were evaluated. For the first cost surface, the SD method was trapped in a local minimum 91% (392/432) of the time. The SA using the Boltzman schedule selected the best architecture after evaluating, on average, 167 architectures. The GA achieved its best performance with linearly scaled roulette-wheel parent selection; however, it evaluated 391 different architectures, on average, to find the best one. The second cost surface contained no local minimum. For this surface, a simple SD algorithm could quickly find the global minimum, but the SA with the very fast reannealing schedule was still the most efficient. The same SA scheme, however, was trapped in a local minimum on the first cost surface. Our CNN study demonstrated that, if optimization is to be performed on a cost surface whose characteristics are not known *a priori*, it is advisable that a moderately fast algorithm such as a SA using a Boltzman cooling schedule be used to conduct an efficient and thorough search, which may offer a better chance of reaching the global minimum. © *2001 American Association of Physicists in Medicine.* [DOI: 10.1118/1.1395036]

Key words: computer-aided diagnosis, mammography, microcalcification, optimization algorithm, convolution neural network

I. INTRODUCTION

Many computer-aided diagnosis (CAD) systems use neural networks (NNs) for either detection or classification of abnormalities on medical images.[1–3] Different CAD systems have different NN implementations with different architectures. An NN architecture is basically determined by the number of input and output nodes, the number of hidden layers, and the number of nodes in the hidden layers. There are no well-established rules to determine the best architecture. Therefore, selecting a network architecture to achieve the best detection or classification results is an open problem. A commonly used approach is to try different combinations of parameters in an *ad hoc* manner and empirically select the ‘‘best’’ architecture based on the test results. However, this manual ‘‘optimization’’ process usually only searches very limited regions of the large-dimensional parameter space.

In order to overcome the difficulties associated with manual optimization, automated methods have been developed. In the *step-by-step* evolution method, there are two major approaches to the automatic selection of NN architectures.[4] One approach, *constructive*, starts with a minimal architecture and keeps on enlarging it until no significant improvement can be observed in the performance of the NN.[5] The other approach, *destructive*, starts with a large initial architecture and prunes it until there is no significant change in the performance.[6] Both of these approaches require the decision of how small (or how large) the initial architecture must be, and how much change in the performance should be considered as the stopping criteria. Additionally, the solution offered by either approach could be a local optimum of the overall cost function, a problem that also manifests itself in the manual search method.

Moody *et al.* proposed the use of a three-stage heuristic method for architecture selection in a two-layer back-propagation NN.[7,8] The first stage, *sequential network construction* (SNC), determines the number of hidden layer nodes. While the number of nodes is increased from a minimum number to a maximum number, the NN weights obtained at each increase are utilized in further steps in a nested manner. As this is a constructive process, the sequential process is terminated when there is no significant performance change. Next, *sensitivity-based pruning* (SBP) reduces the number of input nodes. To measure the sensitivity of the NN to an input node, the NN is fed with the sample average of that input node over time and the effect of this replacement on the training error is measured. Inputs with minimal influence on the error are pruned. In the final step, *optimal brain damage* (OBD), the connections of the NN are pruned if the influence of their weights on the NN training error is not large. In our problem, we keep the number of nodes fixed and assume a fully connected network structure. Therefore, the second and the third stages of this heuristic method are not applicable to our problem. The SNC differs from the manual search method in the respect that NN weights are calculated in a nested manner, i.e., NN weights calculated in one iteration are utilized in other iterations. However, it is basically a constructive method and is still prone to being trapped in a local optimum while increasing the number of nodes of the architecture in the process of optimization.

Another approach to automated architecture selection utilizes genetic evolution and evolutionary algorithms. Maniezzo considered the selection of the architecture and the weights *by genetic evolution*.[9] In genetic evolution, genetic algorithms determine both the architecture and the weight distribution of NNs. Angeline *et al.* proposed the use of *evolutionary programming* for the same problem.[10] Evolutionary programming is similar in principle to the genetic algorithm but mainly uses mutation schemes (see Sec. II F). In this approach, a *network temperature* is defined in terms of the ratio of individual fitness values to the maximum fitness value in the population. This temperature determines the way and the severity of the mutation applied to a generation. In our problem, the number of the NN connection weights is large due to the process of convolution. Therefore, we mainly considered the optimization of the architecture and left the task of optimization of the weights to the NN training by error back-propagation.[3]

In this article, we considered the optimization of a feed-forward convolution neural network (CNN) architecture. Four parameters of the NN architecture were considered for optimization: the number of nodes in the first and second hidden layers, and the kernel sizes of the filters in these hidden layers. These parameters were limited to a finite set of values. In this application, the CNN performed the classification of true-positive (TP) and false-positive (FP) microcalcifications detected on digitized mammograms. We compared three automated methods: steepest descent (SD), simulated annealing (SA), and a genetic algorithm (GA) for selecting an optimal CNN architecture.

The goal of our study is to investigate the use of automated algorithms for optimization of a neural network architecture. We used the problem of optimizing a CNN for classification of true and false microcalcifications as an example to compare the different automated methods. Although the best automated algorithm may depend on the optimization problem, our study demonstrated the feasibility of this approach and the variations of the different techniques. This approach may be adapted to other optimization problems in CAD.

II. MATERIALS AND METHODS

A. Data set

Our data set consisted of region-of-interest (ROI) images extracted from 108 mammograms, which were randomly selected from the files of patients who had undergone biopsies at the University of Michigan. The images included microcalcifications of visibility ranging from subtle to obvious that are typically encountered in mammography practice. The mammograms were digitized with a LUMISCAN 85 scanner at a pixel resolution of 0.05×0.05 mm^2 with 4096 gray levels and then converted to 0.1×0.1 mm^2 resolution by averaging adjacent 2×2 pixels and subsampling. The optical density (OD) range of this digitizer was 0 to 4.0. The digitizer was calibrated so that the gray values were linearly and inversely proportional to the OD with a slope of -0.001 OD/pixel value.

The locations of individual microcalcifications in these images were manually identified and saved in a truth file. After the prescreening stage of the microcalcification detection program,[1] the detected signals were labeled as TP or FP automatically by comparing with the truth file. A 16×16 pixel ROI was then extracted for each of the detected signals and these ROI images were used for training and testing the CNN. Either a true or a false microcalcification was located at the center of the ROI. The microcalcification detection program detected more FP ROIs than TP ROI images at the prescreening stage. In order to have approximately equal numbers of TP and FP ROIs, only a randomly selected subset of FP ROI images was used.

The selected ROIs were divided into two separate groups. For the first part of the experiments, the first group, G1, was used for training the CNN and the second group, G2, was used for testing the trained CNN. For the second part of the experiment, the roles of G1 and G2 were switched. The first group, G1, consisted of 533 TP and 553 FP ROIs. Of the 533 TP ROIs, 293 were extracted from benign clusters and 120 from malignant clusters. Mirror images of the malignant ROIs were also included so that the CNN would be less dependent on the potential biases on the directions of the microcalcification or the tissue texture in the training ROIs. Furthermore, this would make the numbers of malignant and benign ROIs almost balanced. The second group G2 had 547 microcalcification ROIs, 295 of which were benign. The remaining ROIs consisted of 126 malignant microcalcifications and their mirror images. There were 570 FP ROIs in G2. Therefore, G1 contained a total of 1086 ROIs and G2 contained 1117 ROIs.

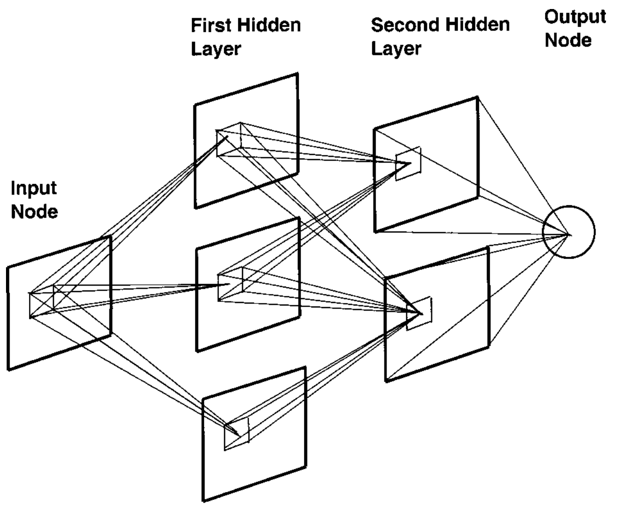


FIG. 1. Convolution neural network architecture. The node values in one layer are convolved with the weights in the "filter" kernels to obtain the node values in the next layer. Input node is an image and only one output node is used.

B. Convolution neural network

The CNN, which is based on the neocognitron structure of Fukushima,[11] was previously used for the detection of lung nodules on chest radiographs, detection of microcalcifications on mammograms, and classification of mass and normal breast tissue on mammograms.[1–3] The CNN structure, shown diagrammatically in Fig. 1, is explained in detail in the literature.[1] The task of the CNN was to classify the input ROI as containing a TP or a FP. During training, the desired output of the CNN was set to 1 for microcalcification ROIs and to 0 for FP ROIs. In this work, the CNN structure had one input image, one output node, and two hidden layers. All node groups in the two hidden layers were fully connected. The node values in one layer were convolved with the weights in the filter kernels to obtain the node values in the next layer. A sigmoidal activation function was used. The initial weights of the CNN were chosen to be uniformly distributed random numbers between -0.5 and 0.5. The CNN was trained using the error back-propagation rule.[3]

For a given CNN architecture, after completion of each training epoch, the classification performance was evaluated on the test set. For evaluation purposes, receiver operating characteristic (ROC) methodology[12,13] was applied to the output values of the CNN. A ROC curve is the relationship between the true-positive fraction (TPF) and the false-positive fraction (FPF) as the decision threshold varies. A commonly used figure of merit for classification performance is the area, A_z, under the ROC curve. The A_z value for classifying the test samples was calculated using the LABROC1 program.[14] At an epoch, whenever the current A_z value became higher than all the previous A_z values, the corresponding kernel weights were recorded to be used in the selection of the best architecture, as described later in this work. The CNN training was terminated when the total squared error value per training sample fell below a preset threshold value (0.03) or the number of epochs exceeded 1000.

TABLE I. Search space for the current optimization problem. Each CNN architecture was a combination of four parameters. The other CNN parameters were fixed.

Optimization parameter	Search space
Node groups in hidden layer 1	1, 2, 4, 6, 8, 10, 12, 14
Node groups in hidden layer 2	1, 2, 4, 6, 8, 10
Kernel size in hidden layer 1	5, 7, 9
Kernel size in hidden layer 2	3, 5, 7

C. Optimization procedure

Because of the computational requirements, we limited ourselves to the optimal selection of four parameters of the CNN architecture in this optimization study: the numbers of node groups in the first and second hidden layers, and the kernel sizes of the filters in these hidden layers. However, these are not all the parameters that can be included in the optimization process. Other possibilities may include the numbers of hidden layers and output nodes, or the form and parameters of the activation functions. These latter parameters were fixed in the current study.

Each optimization parameter can theoretically have a large number of different values. However, it would not be practical, again in terms of computational requirements, to search the entire parameter space for an optimal solution. Therefore, we limited our parameter choices to finite sets of values. The ranges of these parameters were chosen based on our previous experience with the CNN. Table I shows these parameters and their range of values. The complete set of parameters that define a NN architecture is called a *state*. Therefore, there were 432 $(=8\times6\times3\times3)$ possible states in our experiments and four parameters defined in each state.

At each iteration, the state of the network changed if at least one of the parameters (e.g., the number of node groups in the first layer) changed. Two states are called *neighbor states* if the parameters of the states differ only by consecutive numbers (e.g., 6 and 8 for the first parameter). For instance, 2-4-5-7 and 2-6-5-5 are neighbor states. There can be more than one neighboring state. Note that a change from the minimum value of a parameter to the maximum (and vice versa) is not considered a consecutive change (e.g., 1 and 14 for the first parameter or 10 and 1 for the second parameter) and the resultant states with such changes are not considered as neighbor states. A change in the state means a change in *at least one* of the parameters in an architecture.

Definition of a cost function plays an important role in the selection of the architectures. A good cost function should reflect the overall performance of the selected architecture. One such choice is suggested by the ROC methodology.[14] In our experiments, the cost function $f(A)$ for an architecture, A, is defined as $1-A_z$. An alternative for the cost function could be designed by replacing A_z with the partial A_z above TPF$=0.9$ for the ROC curve.

To facilitate the comparison of the performance of the automated methods, we first trained and obtained the test A_z values of all 432 possible architectures in the search space. The ranking of each solution architecture was determined from the test A_z values. The test A_z values were stored in a look-up table. During the optimization process using any algorithm, the performance of any CNN in the search space could be obtained from the look-up table. Since no CNN training was actually performed, we could evaluate different algorithms and initialization conditions very efficiently.

While evaluating automated optimization methods, two figures of merit were considered: the number of different architectures that were evaluated in the selection process, and the ranking of the selected architecture. Evaluation of each architecture requires the training and testing of the NN and this process is orders of magnitude slower than all the other computational requirements. Therefore, the first figure of merit is related to the *computational cost* of the architecture selection procedure. The second figure of merit indicates its ability to select a successful architecture for the classification problem among all possible architectures.

D. Steepest descent method

The SD is one of the most commonly used forms of iterative optimization. In our SD implementation, the search for the optimal architecture starts with a randomly selected architecture. At each iteration, the cost values of the neighbor states are calculated. The neighbor state with the highest A_z value (therefore the lowest cost value) becomes the current state and the next iteration of SD starts from that state. The iterations continue until no neighbor state with a lower cost value can be found.

The SD method may suffer from a number of drawbacks depending on the shape of the cost function.[15] The usual one is the local minima problem. If the cost of one of the architectures is lower than the costs of all its neighbors but still higher than that of a global minimum, then this architecture represents a local minimum on the cost surface. Once the method selects this architecture, it will be trapped in this local minimum.

In order to overcome the inherent problems of the SD method, some stochastic optimization methods have been developed. The SA algorithm and the GA are two commonly used methods. We compared the performances of these methods along with the SD method for the automated NN architecture selection problem, as described later.

E. Simulated annealing algorithm

The SA algorithm emulates the process of determining the lowest energy ground state of a physical system with many interacting atoms.[16] An efficient path of searching for a global minimum is guided by a scalar cost function. The annealing process brings in iterative improvement. Occasionally, solutions with higher cost values are accepted. This reduces the chances that the optimization will be trapped in a local minimum. For this reason, the SA algorithm is used in many applications for optimization of multi-parameter problems.[17]

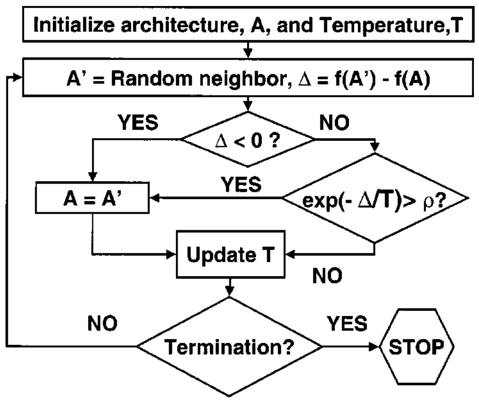

Initialize architecture, A, and Temperature,T
A' = Random neighbor, Δ = f(A') - f(A)
YES
Δ < 0 ?
NO
A = A'
YES
exp(- Δ/T)> ρ?
Update T
NO
NO
Termination?
YES
STOP

FIG. 2. Flow diagram of the SA. ρ is a uniformly distributed random number between 0 and 1. The algorithm terminates when the temperature reaches a predetermined final temperature value.

The SA algorithm is summarized in Fig. 2. The solution, A, is initially set to a random architecture and the algorithm starts at an initial temperature, T_0, which is updated at each iteration according to an update rule. The cost, $f(A)$, for each architecture, A, is defined in the same way as in the SD method. At each iteration, one of the neighbor states is randomly chosen as a candidate for the next solution according to the transition diagram shown in Fig. 3. For a particular parameter other than the first and the final parameter values, the transition to a larger, a smaller, or staying in the same value is random with equal probabilities ($\frac{1}{3}$). For the first and the final parameter values, the probabilities of transition to the neighboring value or staying in the same value are both equal to $\frac{1}{2}$.

After each transition, the difference, Δ, between the cost of the neighbor state and that of the current state is calculated (Fig. 2). If the difference is negative, the neighbor state is always accepted as the current state and the iterations proceed from this state. If the difference is zero or positive, the neighbor state is accepted with a probability of acceptance, κ, defined as $\kappa = \exp(-\Delta/T)$, where T is the current temperature. A uniformly distributed random number, ρ, between 0 and 1, is drawn. If ρ is less than κ, then the neighbor state becomes the current state. Otherwise, the original state is used to start the next iteration. If Δ is very small, solutions that increase the cost will be accepted with relatively high probability until the system reaches a very low (cool) temperature. If Δ is large, the probability of accepting the new solution decreases rapidly with decreasing temperature.

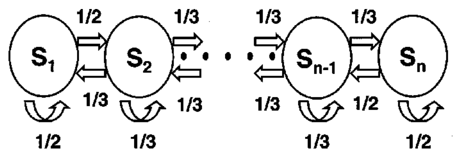


FIG. 3. Transition diagram used with the simulated annealing program. Each parameter value is one of the possible values for a given parameter. The arrows indicate the transition probabilities, which are equal to $\frac{1}{3}$ except for the first (S_1) and the last (S_n) values. The transition probabilities for the first and the last values are equal to $\frac{1}{2}$.

The SA algorithm requires setting an initial temperature, T_0, a termination condition, and a method of updating the temperature. These parameters and conditions are known as the *annealing schedule*. The success and efficiency of the simulated annealing algorithm depend on the annealing schedule. Theoretically, it has been shown that the SA algorithm converges to the global minimum with probability 1 if an appropriate annealing schedule is used.[17] However, the annealing schedule depends on the underlying distribution of the data samples, making it difficult to identify *a priori*.

Proper selection of the initial temperature in the SA algorithm influences the quality of solution. There are several approaches in the literature. In a method proposed by Kirkpatrick *et al.*, a very high initial temperature value, T_0, is chosen at the beginning.[16] The SA is then iterated several times. After each iteration, the acceptance ratio, χ, is calculated as the ratio of number of accepted solutions to all solutions up to that point. If χ is less than a predetermined acceptance ratio, χ_0, then the initial temperature is doubled, otherwise it is halved.

In our experiments we used a method developed by Laarhoven *et al.*, which is a refined version of Kirkpatrick's method.[18] A preliminary SA is run to estimate an appropriate initial temperature. In the preliminary run, the SA iteration starts with a very high initial temperature. This guarantees that all the initial moves are accepted regardless of their cost values. Each time an architecture with a cost value higher than the current cost value is selected, it will be recorded. Finally, after N iterations, the initial temperature is calculated as

$$T_0=\frac{\overline{\Delta C^+}}{\ln[N_+/(\chi_0 N_+-(1-\chi_0)N_-)]}, \tag{1}$$

where N_+ is the number of times a solution with a higher cost value is accepted, $\overline{\Delta C^+}$ is the average of all the cost value increases, $N_-=N-N_+$ is the number of times a solution with a lower cost value is accepted, and χ_0 is the initial acceptance ratio. The calculated initial temperature is then used in the further iterations of the SA method.

The initial temperature is updated according to a *cooling schedule*. This step also plays an important role in the success of the SA. Many methods have been suggested in the literature.[16,19,20] In this study, we investigated the use of four of these methods, namely, the Kirkpatrick, Boltzman annealing, fast annealing, and very fast reannealing. In the cooling schedule of Kirkpatrick *et al.*,[16] the temperature is reduced to a certain percentage of its current value at each iteration:

$$T_n=T_{n-1}\times\alpha, \tag{2}$$

where T_n is the temperature value at the nth iteration and α is a constant between 0 and 1. This temperature update is also equivalent to

$$T_n=T_0\times\alpha^n, \tag{3}$$

where T_0 is the initial temperature.

Another annealing schedule is known as *Boltzman annealing*.[19] In this schedule, the temperature is updated according to the following relation:

$$T_n=\frac{T_0}{\ln(n+1)}, \tag{4}$$

where T_0 is again the initial temperature and n is the number of iterations.

The annealing schedule known as the *fast annealing* has the following relationship:[20]

$$T_n=\frac{T_0}{n^{1/k}}, \tag{5}$$

where $k=1,2,3,...$ determines the speed of cooling. As the value of k increases, the cooling becomes slower.

The final cooling schedule we evaluated was *very fast reannealing*.[21] In this cooling schedule the temperature is updated according to the relationship

$$T_n=T_0\exp(-cn^{1/k}), \tag{6}$$

where c determines the speed of convergence. The value of c is calculated using the relationship

$$c=N_{\max}^{-1/k}\ln\frac{T_0}{T_f}, \tag{7}$$

where T_f is the final temperature, $N_{\max}$ is the number of iterations allowed to go from the initial temperature to the final temperature, and $k=1,2,3,...$ is the speed factor.

F. Genetic algorithm optimization

The GA optimization is inspired by the concepts of evolution.[22,23] The optimization parameters are coded as the chromosomes of the individuals of a population. Each population generates a new population through evolutionary concepts such *as parent selection, cross over, and mutation*. It is assumed that each generation will produce some better offspring and that the strength of these offspring will be transferred to new generations through evolutionary mechanisms.

Figure 4 summarizes the steps in the GA. Solutions in the GA terminology are called *chromosomes* and they are expressed as binary strings. The first step in the GA process is encoding of solutions as strings. For instance, in our search space, the first and the second parameters, i.e., the number of node groups in the first and second hidden layers, can be expressed by 3 bits whereas only 2 bits are enough to encode the third and the fourth parameters. Hence, the chromosome length is 10. Each architecture in our search space is expressed as a 10 bit binary chromosome. For example, the architecture 12-8-5-5 is expressed as 1101000001. Note that this encoding is applied to the array indices for 12, 8, 5, 5 which are the seventh (110), fifth (100), first (00), and second (01) values, respectively, for the four parameters in the parameter space.

In the GA terminology, each chromosome has a fitness value and the GA method tries to maximize the fitness values in each generation. In the SA method, the optimization targets to minimize the cost value, which was chosen as (1

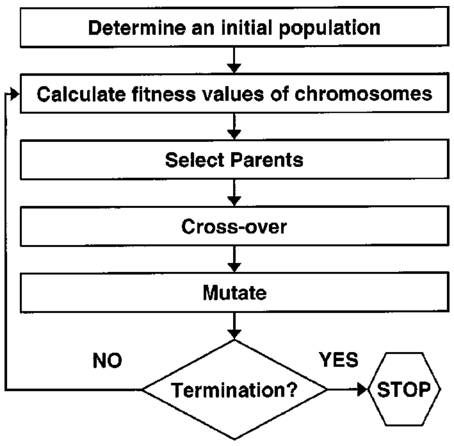

Determine an initial population
Calculate fitness values of chromosomes
Select Parents
Cross-over
Mutate
NO
YES
Termination?
STOP

FIG. 4. Flow diagram of the GA. The algorithm terminates when the maximum number of generations is reached or a stable solution is obtained.

$-A_z$). Accordingly, we chose the fitness value of an architecture as its test A_z value. The fitness values of architectures that are not among the search space of 432 architectures but still generated during the GA process (due to 10 bit representation) are assigned to be zero.

The search for an optimal architecture usually starts with a randomly created population of architectures. The choice of the population size is a critical factor for the quality of the solution. The larger the population, the higher the chances of achieving the optimal solution. On the other hand, a large population increases the computational cost of the GA.

The reproduction process starts after an initial population has been chosen. Reproduction directs the search to areas of the search space with high fitness values. The first step in reproduction is *parent selection*. In this step, chromosomes in the current population are chosen and matched to produce the next generation. There are several heuristic methods in the literature for parent selection.[22] In this work, we experimented with three different variations: roulette-wheel selection, roulette-wheel selection with linear scaling, and the stochastic remainder method.

In the *roulette-wheel parent selection* method, the chances of reproducing from a chromosome is directly proportional to its fitness value. First, a running total fitness value for chromosome i, $F(i)$, is calculated:

$$F(i)=\sum_{k=1}^{i} f(k), \tag{8}$$

where $f(k)$ are individual fitness values, $k=1,...,N$, for a population of size N. Then, a random number, γ, is generated between 0 and $F(N)$. The roulette-wheel selection strategy selects the ith chromosome for reproduction if

$$F(i-1)<\gamma<F(i). \tag{9}$$

In the *linearly scaled version of the roulette-wheel* selection, the minimum fitness value, $f_{\min}$, and the maximum fitness value, $f_{\max}$, are determined in the population of N chromosomes. The fitness value of each chromosome, $f(k)$, is normalized as $f'(k)=(f(k)-f_{\min})/(f_{\max}-f_{\min})$ for $k=1,...,N$. The random selection procedure previously described is then applied to these normalized fitness values.

In the *stochastic remainder* parent selection method, the average fitness value, f_{avg}, is first calculated.[24] Then, each fitness value within the population is divided by this average value. The integer parts of this division for each chromosome determine how many copies of the chromosome will be used in the reproduction process. The remaining candidates for reproduction are determined by a roulette-wheel selection on the fractional parts of the division.

After parent selection, the chromosomes evolve using two basic genetic operations *cross-over* and *mutation*. In the cross-over process, chromosome information from the selected parents is exchanged. The cross-over probability determines the chance that genetic materials from two parents will be mixed to produce offspring. A random number in (0,1] is generated and it is compared to the preselected cross-over probability. If the generated random number is smaller than the cross-over probability, a cross-over occurs. The parent chromosomes are cut at a random location into left and right parts. Then, the left part of the first parent is combined with the right part of the second parent, and vice versa. The mutation process creates new offspring by randomly changing some bits in the chromosome according to a mutation probability. Mutation brings diversity to the population. The mutation probability determines the chance that a mutation will occur. A random number in (0,1] is generated for each bit in the chromosome. If the generated random number is smaller than the preselected mutation probability, then that bit is replaced with its binary complement value.

III. RESULTS

We performed two sets of experiments by using the two ROI groups alternately as training and test sets. These two combinations of training and test sets resulted in two different cost surfaces; one surface had several local minima and the other surface did not contain local minimum. The test A_z values for the evaluated architectures of the first cost surface (train G1–test G2) varied between 0.793 for the architecture 2-1-9-7 and 0.913 for the architecture 14-4-5-5, which was the optimal architecture on this cost surface. For the second cost surface (train G2–test G1) these values varied between 0.787 for the architecture 4-1-5-3 and 0.930 for the architecture 14-10-5-7, which was the optimal architecture on this cost surface. Note that if the data set is infinitely large, the cost surface of the two combinations should be essentially identical, other than small statistical fluctuations. However, the small sample size available for this study caused the large differences in the two cost surfaces. For the purpose of this study, this offered us the opportunity to demonstrate the dependence of the performance of the optimization algorithms on the characteristics of cost surfaces. In the following two sections, the results of optimization for these two types of surfaces will be discussed.

TABLE II. Parameters of simulated annealing algorithm.

Cooling schedule	Parameter	Range	
Kirkpatrick	α	[0.7,0.998]	
Boltzman	$\frac{T_f}{T_0}$	[0.1,0.3]	
Fast annealing	$\frac{T_f}{T_0}$	$k=1$	[0.0001,0.1]
		$k=2$	[0.001,0.5]
		$k=3$	[0.01,0.5]
		$k=4$	[0.1,0.5]
Very fast reannealing	$\frac{T_f}{T_0}$	$k=1$	[0.01,0.5]
		$k=2$	[0.001,0.3]
		$k=3$	[0.001,0.2]
		$k=4$	[0.0001,0.1]

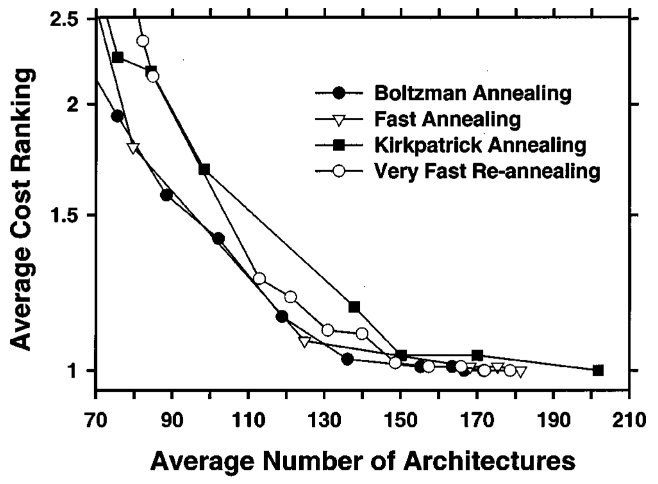


FIG. 5. Average cost ranking versus the average number of selected neural network architectures for different cooling schedules of the SA. The best results are obtained using the Boltzman cooling schedule.

A. Results of optimization—Training with G1 and testing with G2

In our experiments with the SD method, we experienced the local minimum problem for this cost surface. We initiated the SD optimization starting at each of the possible architectures, resulting in 432 different experiments. On the average, the SD method evaluated 108 architectures before reaching a solution and the average ranking of the solution architectures was 2.0. The architecture of the first rank was selected as the solution architecture in only 9% (40/432) of the experiments. The other 91% of the solutions were architectures ranked as the 2nd, 5th, and 11th place. These architectures represented the local minima of the cost function.

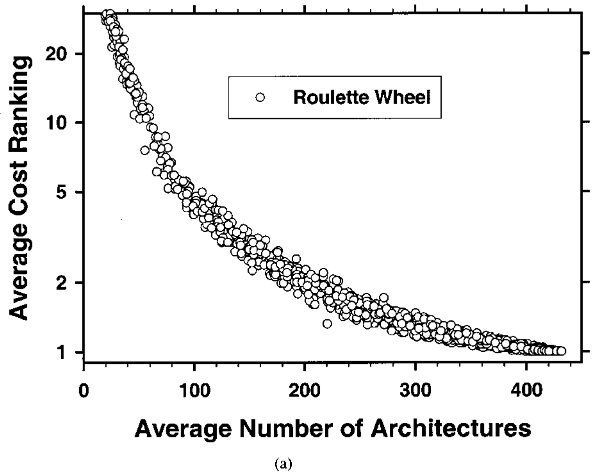
In our SA experiments, the initial temperature in the preliminary SA run was chosen as 10^6 and the initial number of iterations, N, was chosen as 10. These values were not very critical because they were used only for estimation of the initial temperature to be used for the actual SA run (see Sec. II E). The initial architecture was arbitrarily selected as 1-1-5-3 (the minimum numbers in the parameter space). When the Kirkpatrick annealing schedule was used, the cost value increased in four of the ten iterations and the average increase in the cost value, $\overline{\Delta C^+}$, was found to be 0.014. We selected the initial acceptance ratio, χ_0, as 0.8, a value typically chosen in the SA experiments. Thus, the initial temperature was calculated as 0.0205. The architecture with the lowest cost value in the first ten iterations was found to be 6-2-7-7 (its cost value is the 139th lowest cost among all 432 cost values). Further SA experiments started with this initial temperature value and random initial architectures.

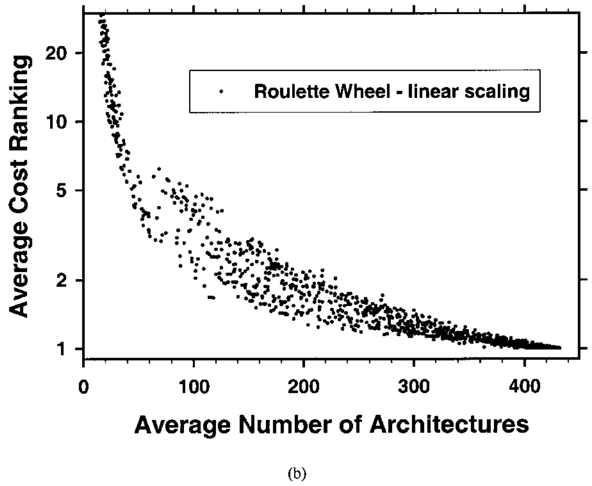
Each cooling schedule of the SA algorithm has a different number of parameters. Parameters for each cooling schedule were varied over a wide range in order to provide a fair assessment for each algorithm. Table II shows the range of these values. In the very fast reannealing algorithm $N_{\max}$ was chosen to be 1000. Additionally, each experiment was repeated 100 times with different random number seeds and the results were averaged to reduce the variability due to the stochastic nature of the SA algorithm. Figure 5 shows the average cost ranking plotted against the average number of architectures evaluated for some combinations of parameters used in this study. Note that a curve on this graph does not represent a functional relationship. The individual data points for the same annealing schedule were linked to facilitate reading. On a given curve each point represents the average performance of the SA with a given set of parameters. The best results with fast annealing were obtained when $k=3$ and the best results with very fast reannealing were obtained when $k=4$. These k values were used in the plots of Fig. 5. Our results indicate that, for this application, the best results are achieved using the Boltzman annealing schedule. The SA with the Boltzman schedule and $T_f/T_0=0.15$ selected the best architecture (14-4-5-5, test $A_z=0.912$) after evaluating, on average, 167 architectures. The performance of fast annealing is very similar to that of the Boltzman annealing schedule. The fast annealing schedule with $T_f/T_0=0.08$ evaluated an average of 181 architectures to reach the best solution.

In our experiments with the GA, we varied the population size, the maximum number of generations, cross-over probability, and the mutation probability. Table III gives the ranges of values for these parameters. Similar to the SA experiments, we varied each parameter within its parameter space and initiated the experiments 100 times with different random number seeds to account for the variability due to the stochastic nature of the GA. Figures 6(a)–(c) show the average cost ranking for these architectures versus the aver-

TABLE III. Parameters of the genetic algorithm.

GA parameter	Parameter values
Population size	10, 40, 50, 80, 100, 150, 200
Max. no. of generations	5, 10, 15, 20, 30, 50, 75, 100
Cross-over rate	0.5, 0.6, 0.7, 0.8, 0.9
Mutation rate	0.005, 0.01, 0.02, 0.03





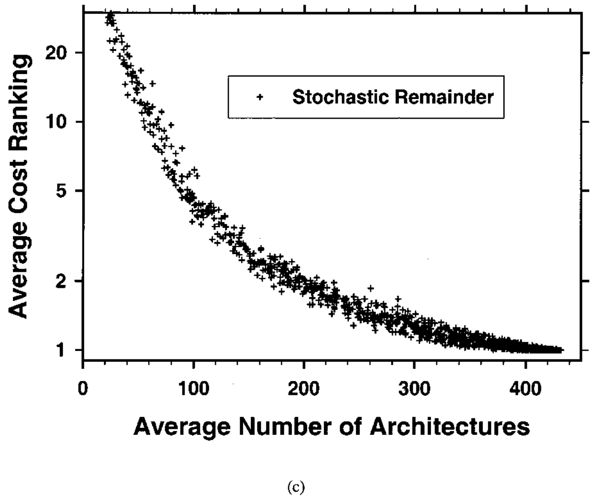


FIG. 6. Average cost ranking versus the average number of selected neural network architectures for GA. (a) Roulette-wheel parent selection method. (b) Roulette wheel with linear scaling parent selection method. (c) Stochastic remainder parent selection method.

age number of evaluated architectures for the three different parent selection methods, respectively. It may be observed from these figures that the GA achieved its best performance with linearly scaled roulette-wheel parent selection, which needed to evaluate 391 different architectures, on average, to find the best architecture. The GA that reached this performance had a population size of 150, a maximum number of generations of 75, a cross-over probability of 0.7, and a mutation probability of 0.005.

B. Results of optimization—Training with G2 and testing with G1

We repeated the experiments in Sec. III A by swapping the training and test sets. The A_z values for training with G2 and testing with G1 were larger than those values obtained for training with G1 and testing with G2. This can be attributed to better training, an easier test set, or both. The selected best architecture was 14-10-5-7 with a test A_z value of 0.930, which was the fourth architecture in our previous set of experiments. We also observed that the SD algorithm could always reach the best architecture regardless of the initial architecture. We checked the neighboring gradients of all 432 points on the cost surface and confirmed that this cost surface did not contain any local optima.

For this cost surface we observed that the SA also achieved the best performance. As in the previous experiment, all SA schedules included in this study gave better performance than the best GA parent selection method. However, the best annealing schedule was the very fast reannealing (VFRA) schedule. We experimented with the parameters of the different schedules to change their cooling rates. It was observed that the faster the cooling rate is, the faster the global minimum can be found. From the temperature update relationships of the four SA cooling schedules (Sec. II E), it can be shown that the parameter of a VFRA schedule can be chosen to have the fastest cooling rate among the four for a given number of iterations (Kirkpatrick can overtake VFRA after some initial number of iterations). We applied the fastest VFRA schedule to the first example, i.e., a cost surface with local minima, and it was trapped in the local minima most of the time, as was the SD algorithm. These experiments therefore indicate that, for a cost surface without local optima, a very fast cooling schedule such as VFRA or SD will be the most efficient method to search for the global optimum. However, a very fast cooling schedule is almost guaranteed to be trapped in a local optimum on a cost surface with local optima.

IV. DISCUSSION

The SD, SA, and GA optimization methods are some of the most commonly used optimization techniques. In this work, we compared their performance and demonstrated their usage as an alternative to a manual search method. Manual search can only examine a very limited parameter space because it is time consuming, requires human intervention, and can easily be trapped in a local optimum because of ''satisfaction of search.''

In this study, we evaluated the automated optimization algorithms using two different cost surfaces, one of which did not contain any local minimum. For a general CAD optimization problem, such a surface is rarely found. Therefore, we believe that the results obtained from the first cost sur-

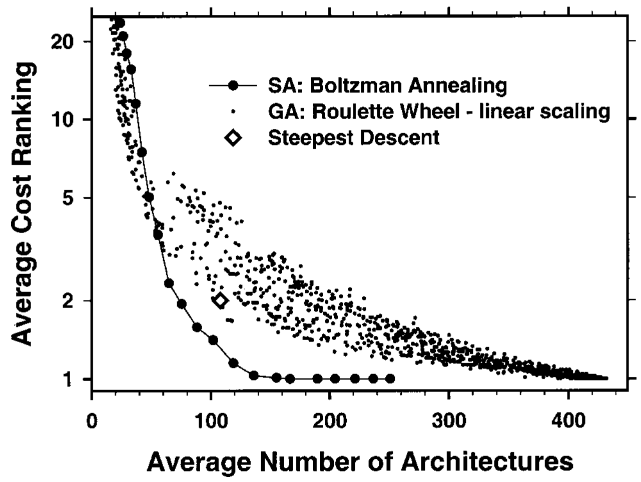


FIG. 7. Comparison of the performance of three optimization procedures, the simulated annealing, the genetic algorithm, and the steepest descent. The simulated annealing algorithm used the Boltzman cooling schedule and the genetic algorithm used the roulette-wheel parent selection method with the linear scaling of the fitness values.

face, which are presented in Sec. III A, are more general than those obtained with the second cost surface. In the following paragraphs results for the first cost surface are discussed.

In order to compare the SA and GA methods, we chose the best performing SA cooling schedule (the Boltzman schedule) and the best performing GA parent selection method (roulette-wheel selection with linear scaling). Figure 7 compares the performance of the two optimization schemes. The performance of the SD on this cost surface was also plotted. If the performances of the algorithms are compared in terms of the average number of architectures evaluated to reach the best architecture, the SA method was able to find the optimal solution with the least computational cost on average. The SA using Boltzman schedule selected the best architecture after evaluating, on average, 167 architectures. The GA using linearly scaled roulette-wheel parent selection, however, had to evaluate 391 different architectures, on average, to find the best one. These values were obtained by finding the minimum of the average number of architectures evaluated to obtain a cost ranking of 1 in Fig. 7. Alternatively, if we compare the average cost ranking that can be achieved for a fixed average number of architectures evaluated, the SA with Boltzman schedule could reach a lower cost architecture than the GA when the average number of architectures was greater than about 50. Furthermore, if we choose a fixed average cost ranking, the SA required a lower average number of architectures evaluated than the GA when the average cost ranking was lower than about 5. Above this range the performance of GA was somewhat better. This relative performance is expected to depend on the shape of the cost surface but not on the absolute differences in the costs of the architectures of consecutive ranks. Whether the differences are statistically significant will not change the relative performance comparison, except that the user may want to select an appropriate stopping criterion that is suited for their problem.

TABLE IV. Variations in optimization solutions.

	Mean	Standard Deviation	Minimum	Maximum	Median
		(a) SA and GA comparison			
SA (Boltzman schedule) No. of Architectures	167	23.3	115	224	164
SA (Kirkpatrick schedule) No. of Architectures	202	22.6	140	251	203
SA (Fast schedule) No. of Architectures	181	22.8	142	235	180
SA (VFRA schedule) No. of Architectures	172	22.9	123	219	170
GA (Linear scaling) No. of Architectures	391	22.4	325	430	396
		(b) SD experiments			
SD—432 Expt. No. of Architectures	108	37.7	24	203	108
SD—432 Expt. Solution rank	2	1.1	1	11	2
SD—100 Expt. No. of Architectures	108	38.9	24	195	110
SD—100 Expt. Solution rank	2	1.3	1	11	2

Table IV(a) shows the mean and variation of the solutions during repeated experiments with different random number seeds. We repeated each GA and SA experiment 100 times. It is observed that the SA and GA have similar variances. The SD results were obtained from all 432 experiments corresponding to all possible initial configurations. We also selected 100 architectures randomly among the 432 architectures and compared the results of the SD experiment with those of the GA and SA experiments as shown in Table IV(b). It should be noted that, in the SD experiments, the final A_z rankings that the optimization procedure could achieve vary so that the average is also shown in the table. It can be observed that, for the SD method, the results obtained from selecting 100 random architectures are similar to those obtained from all 432 architectures.

For this application, the optimal CNN architecture on both cost surfaces contained a relatively large number of node groups. However, since the cost surface was determined by test performance rather than training performance, it is expected that the CNNs were not overdesigned. Overtrained neural networks usually have poor generalization. For the CNN architectures included in this study, the A_z values steadily increased (with statistical fluctuation) towards the maximum. Figure 8 shows a surface plot demonstrating the dependence of the average test A_z values on the node

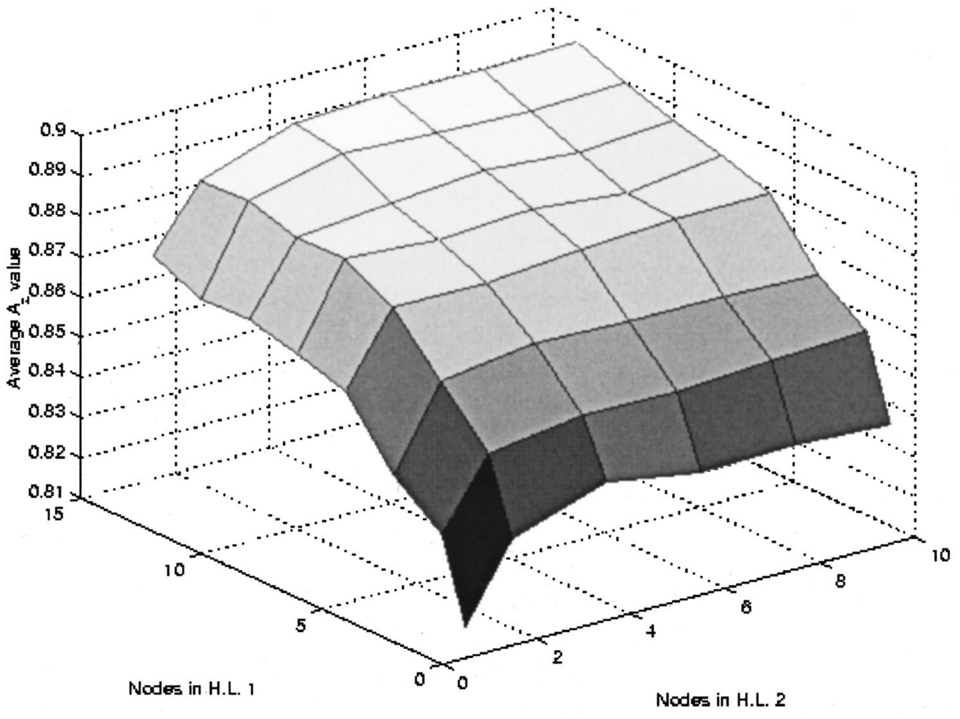


FIG. 8. Dependence of the average test A_z values on the node group numbers in the first hidden layer (H.L. 1) and the second hidden layer (H.L. 2). The averaging was performed over the different filter kernel sizes. As the number of node groups increased, the average A_z value also increased.

group numbers in the first hidden layer and the second hidden layer when G1 was used for training and G2 for testing. Note that this was not the four-dimensional cost surface used in the optimization process. To facilitate plotting, the test A_z values for a given pair of node group numbers in the two hidden layers were averaged over the different filter kernel sizes. This graph indicated that the average test A_z value increased as the number of node groups increased in both layers within the range studied.

As demonstrated in this study, the optimization functions (fitness function, annealing schedule, etc.) and parameters of the algorithms have a strong influence on the results. The SA and GA methods can have different variations of optimization functions. We examined four SA methods using different cooling schedules and three GA methods using different parent selection methods. Once the general form of an optimization function is determined, there are still more parameters that need to be set such as the population size in the GA and the final temperature in the SA as shown Tables II and III. In summary, the optimization process involves not only finding the best solution with one of the methods but also finding a good optimization function and parameters. The overall efficiency for a given method will have to take into account how to find the best function and parameters.

Depending on the nature of the problem and a particular implementation, one optimization method can be found more efficient than the others. Ingber and Rosen[25] compared genetic algorithms and the very fast simulated reannealing algorithm and concluded that the SA was not only an efficient search strategy, but was also statistically guaranteed to find the optimum of the function. Mitchell *et al.*[26] compared the GA with the SD (referred to as ''Hill Climbing'' in their work) using the ''Royal Road'' function. While the SD outperformed one version of GA, it was inferior to another version for the same problem.

Microcalcifications and the surrounding tissue texture in a mammogram image, on average, should not have a directional or spatial preference so that the CNN should be spatially invariant and rotationally symmetric. If computation time is not a consideration, ideally one should mirror and rotate each ROI and its mirror image at four orientations to create a set of training samples that are balanced in all directions. Mirroring and rotating an ROI image that contains a microcalcification simulate the ROI images with microcalcifications at different orientations. Although a given weight in a kernel (node group) will be connected to each of the pixels in the process of convolution without mirroring or rotation, the kernel as a group sees different neighboring pixels in a mirrored or rotated ROI. This means that the weights in a node group will be trained by different neighborhoods of pixels by using a mirrored or rotated ROI image. The weights in the CNN kernels will therefore be trained with more different pixel neighborhoods. This will provide an averaging effect to improve the training of the spatially invariant and rotationally symmetric properties of the CNN. It is possible to achieve similar training effect by increasing the number of independent training ROI samples, but it requires at least an eightfold increase in the sample size. The mirroring and rotating approach is much more efficient.

In this study, because of the very long processing time for training the 432 CNNs, we only included the mirrored images of the malignant microcalcification ROIs without rotation. The trained CNNs may not be as robust as those would have been if all ROI patterns were included. However, the relative performance of the CNNs may be reasonably ranked. Since a limited sample size generally causes poorer generalization (i.e., poorer test result) for classifiers with a larger number of weights, the fact that the CNNs with large architectures were ranked higher indicate that they were better than the smaller ones despite the possible poorer training. Furthermore, for the purpose of demonstrating the application of the automated algorithms to neural network optimization, the cost surfaces formed with these test A_z values were adequate even though they might not be the best possible.

When we performed some experiments without mirroring the ROIs in this study and without mirroring and rotation in our previous studies, the test results were consistently poorer than those obtained from training with mirroring and rotation. Since overtraining generally leads to poor generalization rather than improved test results, the observed improvement in generalization supports our assumption that inclusion of the mirrored ROI images provides additional training samples that improve the training of a spatially invariant and rotationally symmetric CNNs.

In this study, we observed that linear scaling of the fitness values improves the performance of the GA. This may be partly due to the fact that scaling changes the relative area distribution on the roulette wheel and gives a bias towards chromosomes with higher fitness values. At the same time, chromosomes with the minimum fitness value are eliminated from the solution space (they can reappear through crossover or mutation). Since solutions with higher fitness values are closer to the optimal value, the scaling process improves the chance of evolving toward the optimal result.

In an optimization task, ranking is the most commonly used criterion by the optimization procedure. In each step, an optimization algorithm compares the relative performance of the CNNs, such as the test A_z (or some other figures of merit), and that is a ranking process. We therefore did not emphasize the specific test A_z values of the 432 CNNs. It is possible to impose other criteria such as selecting the simplest architecture among the already-evaluated architectures that have test A_z values within one standard deviation of one another, or stopping the iteration automatically if the test A_z value does not improve more than a standard deviation within a preset number of iterations, etc. However, the goals of this study are to demonstrate that an automated optimization procedure can be used to find the true global minimum on NN cost surfaces for a given CAD application, and to compare the efficiency of different automated optimization procedures. The most effective optimization procedure is defined in this study as the one that finds the global minimum with the least effort, regardless of the depth of the global minimum relative to the others. The user may impose other criteria to select the ''optimal'' NN architecture based on its application, which is out of the scope of this study. We do not intend to compare the automated optimization procedures with different optimization criteria or for different optimization applications. However, interested readers may follow a similar approach as described in this article and investigate the different variations of this problem.

If a CAD system utilizes a NN, an architecture needs to be selected for the NN regardless of its type. The optimization algorithms discussed here are not specific to the CNN, the prescreening program, which detects suspicious regions, or the classification application. The algorithms require only the definition of a cost function and the values of this cost function for a parameter space to select the architecture of the NN in a CAD system. Therefore, the automated methods evaluated in this study can be easily adapted to other neural network architecture selection problems in other CAD applications, or in any other applications.

The training and testing of the neural network for 432 architectures took about 1 month on a Compaq Alpha Workstation (512 MB RAM, 600 MHz CPU speed). In many NN architecture selection problems, the number of architectures needed to be evaluated using an exhaustive search approach can be so large that it is practically impossible. Many researchers will instead use manual search for such an architecture selection problem for CAD system optimization. The problems associated with manual search have been discussed earlier. Automated algorithms such as those investigated here will be much better alternatives because they are faster, more systematic, and can be carried out with minimal operator intervention This type of application is more general than the specific application of optimizing CNN for our CAD system. We intend to introduce the different automated optimization algorithms to other CAD applications. Our approach can be adapted by other researchers for optimization of NNs or the entire CAD system in their applications.

V. CONCLUSION

In this study, we investigated the utilization of automated optimization methods for the selection of CNN architecture. We compared the performance of the SD, SA, and GA methods for the automated architecture selection task and demonstrated the variability of the optimization algorithms to the input parameters. Our experiments indicate that, for the optimization of CNN architecture for microcalcification detection, the SA using a Boltzman cooling schedule was the most efficient approach for a general cost surface. While the SA algorithms using the fastest cooling schedule (VFRA) were the most efficient approach for a cost surface that did not contain any local minimum, they were inevitably trapped in local minima on a general cost surface. Since the characteristics of the cost surface are generally not known *a priori* in an optimization problem, it is advisable that a moderately fast algorithm such as an SA using a Boltzman cooling schedule be used to conduct an efficient and thorough search, which may offer a better chance of reaching the global minimum. Although the relative effectiveness of the optimization methods depends on the structure of the cost surface, the type, and the parameters of the chosen optimization algorithms, we demonstrated the approach of using optimization algorithms as an alternative to the commonly used manual ''optimization'' method. Furthermore, due to the variability, it is always advantageous to carry out optimization with different input parameters, within the constraint of computational costs.

The CNN-based algorithm that classifies true and false microcalcifications is part of a microcalcification detection program.[1] Since classification of true and false signals is only one of the steps in detection, the performance of the selected architecture should be further evaluated in terms of the detection results of the CAD system. Alternative performance measures for the optimal architecture selection can be developed based on the overall detection results. Our previous study of using a manually selected CNN in the microcalcification detection system indicated that the CNN, although it may not be optimal, improved the detection free response receiver operating characteristic curves substantially.[1] Further studies are underway to investigate the effects of optimal architecture selection on the detection performance of our current CAD system.

ACKNOWLEDGMENTS

This work is supported by a USPHS Grant No. CA 48129 and a Career Development Award (BS) from the USAMRMC (DAMD 17-96-1-6012). The content of this publication does not necessarily reflect the position of the government and no official endorsement of any equipment and product of any companies mentioned in the publication should be inferred. The authors are grateful to Charles E. Metz, Ph.D., for providing the LABROC program.

[a]Author to whom correspondence should be addressed: Department of Radiology, University of Michigan, 1500 E. Medical Center Drive, CGC B2103, Ann Arbor, MI 48109-0904. Electronic mail: gurcan@umich.edu

[1] H. P. Chan, S. C. B. Lo, B. Sahiner, K. L. Lam, and M. A. Helvie, ''Computer-aided detection of mammographic microcalcifications: Pattern recognition with an artificial neural network,'' Med. Phys. **22**, 1555–1567 (1995).

[2] S. C. B. Lo, H. P. Chan, J. S. Lin, H. Li, M. Freedman, and S. K. Mun, ''Artificial Convolution neural network for medical image pattern recognition,'' Neural Networks **8**, 1201–1214 (1995).

[3] B. Sahiner, H. P. Chan, N. Petrick, D. Wei, M. A. Helvie, D. D. Adler, and M. M. Goodsitt, ''Classification of mass and normal breast tissue: A convolution neural network classifier with spatial domain and texture images,'' IEEE Trans. Med. Imaging **15**, 598–610 (1996).

[4] J.-P. Vila, V. Wagner, P. Neveu, M. Voltz, and P. Lagacherie, ''Neural network architecture selection: new Bayesian perspectives in predictive modeling: Application to a soil hydrology problem,'' Ecol. Model **120**, 119–130 (1999).

[5] T. Y. Kwok and D. Y. Yeung, ''Constructive algorithms for structure learning in feedforward neural networks for regression problems,'' IEEE Trans. Neural Netw. **8**, 630–645 (1997).

[6] R. Reed, ''Pruning algorithms—a review,'' IEEE Trans. Neural Netw. **4**, 740–747 (1993).

[7] J. Moody and J. Utans, ''Architecture selection strategies for neural networks: Application to corporate bond rating prediction,'' in *Neural Networks in Capital Markets* (Wiley, New York, 1994).

[8] J. Moody, ''Prediction Risk and Architecture Selection for Neural Networks,'' *NATO ASI Series F, From Statistics to Neural Networks: Theory and Pattern Recognition Applications* (Springer-Verlag, New York, 1994).

[9] V. Maniezzo, ''Genetic evolution of the topology and weight distribution of neural networks,'' IEEE Trans. Neural Netw. **5**, 39–53 (1994).

[10] P. J. Angeline, G. M. Saunders, and J. B. Pollack, ''An evolutionary algorithm that constructs recurrent neural networks,'' IEEE Trans. Neural Netw. **5**, 54–64 (1994).

[11] K. Fukushima, S. Miyake, and T. Ito, ''Neocognitron: A neural network model for a mechanism of visual pattern recognition,'' IEEE Trans. Syst. Man Cybern. **SME-13**, 826–834 (1983).

[12] J. A. Swets, ''ROC analysis applied to the evaluation of medical imaging techniques,'' Invest. Radiol. **14**, 109–121 (1979).

[13] D. M. Green and J. A. Swets, *Signal Detection Theory and Psychophysics* (Wiley, New York, 1966).

[14] C. E. Metz, B. A. Herman, and J. H. Shen, ''Maximum-likelihood estimation of receiver operating characteristic (ROC) curves from continuously-distributed data,'' Stat. Med. **17**, 1033–1053 (1998).

[15] W. H. Press, B. P. Flanner, S. A. Teukolsky, and W. T. Vetterling, *Numerical Recipes in C* (Cambridge University Press, New York, 1988).

[16] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, ''Optimization by simulated annealing,'' Science, 671–680 (1983).

[17] D. F. Wong, H. W. Leong, and C. L. Liu, *Simulated Annealing for VLSI Design* (Kluwer Academic, Boston, 1988).

[18] P. J. M. van Laarhoven and E. H. L. Aarts, *Simulated Annealing: Theory and Applications* (Kluwer Academic, Norwell, MA, 1987).

[19] S. Geman and D. Geman, ''Stochastic relaxation, Gibbs distribution and the Bayesian restoration in images,'' IEEE Trans. Pattern Anal. Mach. Intell. **6**, 721–741 (1984).

[20] H. Szu and R. Hartley, ''Fast simulated annealing,'' Phys. Lett. A **122**, 157–162 (1987).

[21] L. Ingber, ''Very fast simulated re-annealing,'' Math. Comput. Model. **12**, 967–963 (1989).

[22] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning* (Addison-Wesley, New York, 1989).

[23] P. Sutton and S. Boyden, ''Genetic algorithms: A general search procedure,'' Am. J. Phys. **62**, 549–552 (1994).

[24] L. B. Booker, ''Intelligent behavior as an adaptation to the task environment,'' Ph.D. dissertation, University of Michigan, 1982.

[25] L. Ingber and B. Rosen, ''Genetic algorithms and very fast simulated reannealing: A comparison,'' Math. Comput. Model. **16**, 87–100 (1992).

[26] M. Mitchell, J. H. Holland, and S. Forrest, ''When will a genetic algorithm outperform hill climbing? in *Neural Information Processing Systems, 6*, edited by J. D. Cowan, G. Tesauro, and J. Alpspector (Morgan Kaufman, San Franscisco, CA, 1994).