IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received October 9, 2018, accepted November 1, 2018, date of publication November 13, 2018,
date of current version December 7, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2880416

Evolving Image Classification Architectures With Enhanced Particle Swarm Optimisation

BEN FIELDING AND LI ZHANG, (Member, IEEE)
Department of Computer and Information Sciences, Faculty of Engineering and Environment, Northumbria University, Newcastle upon Tyne NE1 8ST, U.K.
Corresponding author: Li Zhang (li.zhang@northumbria.ac.uk)

This work was supported in part by RPPtv Ltd. and in part by Northumbria University for jointly funding an industrial collaborative Ph.D. studentship.

ABSTRACT Convolutional Neural Networks (CNNs) have become the de facto technique for image feature extraction in recent years. However, their design and construction remains a complicated task. As more developments are made in progressing the internal components of CNNs, the task of assembling them effectively from core components becomes even more arduous. To overcome these barriers, we propose the Swarm Optimized Block Architecture, combined with an enhanced adaptive particle swarm optimization (PSO) algorithm for deep CNN model evolution. The enhanced PSO model employs adaptive acceleration coefficients generated using several cosine annealing mechanisms to overcome stagnation. Specifically, we propose a combined training and structure optimization process for deep CNN model generation, where the proposed PSO model is utilized to explore a bespoke search space defined by a simplified block-based structure. The proposed PSO model not only devises deep networks specifically for image classification, but also builds and pre-trains models for transfer learning tasks. To significantly reduce the hardware and computational cost of the search, the devised CNN model is optimized and trained simultaneously, using a weight sharing mechanism and a final fine-tuning process. Our system compares favorably with related research for optimized deep network generation. It achieves an error rate of 4.78% on the CIFAR-10 image classification task, with 34 hours of combined optimization and training, and an error rate of 25.42% on the CIFAR-100 image data set in 36 hours. All experiments were performed on a single NVIDIA GTX 1080Ti consumer GPU.

INDEX TERMS Computer vision, convolutional neural networks, deep learning, evolutionary computation, image classification, particle swarm optimization.

I. INTRODUCTION

Despite being a relatively mature concept, Convolutional Neural Networks (CNNs) have proven to be incredibly effective feature extractors in recent computer vision research. They were originally proposed and proven effective by LeCun *et al.* [1] in 1989 for classifying handwritten numerical digits from a dataset now commonly known as MNIST, but subsequently fell out of favour for a number of years. Since then, CNNs have experienced a large resurgence in popularity, particularly for challenging computer vision tasks requiring effective feature extraction. This rise in popularity has been motivated largely by the increases in performance demonstrated by CNNs on challenging tasks such as the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) [2]. The ILSVRC has been running annual competitions since 2010, backed by the ImageNet dataset which contains over 14 million images with various associated metadata, organised to follow the paradigms of the popular WordNet dataset [3]. Since 2015, all of the winners, and a majority of the entrants, have each been some variation of a deep CNN, with a newly proposed architecture being the improving factor for a number of the entrants. Often these new architectures are then used by subsequent works in relevant sub-fields by exploiting the pre-trained networks as essentially pre-created feature extractors and using transfer-learning techniques to apply them to the new tasks. Since this widespread adoption of deep convolutional networks as the go-to technique for image feature extraction, it has become necessary for many researchers to integrate these pre-trained networks into their own models, where previously they would have used a hand-designed feature extractor such as SIFT [4] or HOG [5]. Currently, the choice of pre-trained network comes from the few existing, well-known architectures that have been proposed and validated in deep learning literature, but modifying these architectures to better suit a particular task requires deep domain knowledge that is often unavailable to researchers or end-users in many sub-fields.

In this research, we refer to the specific combination of layers and parameters that make up the structure of a CNN as the 'architecture' of the network. This combination of design choices rapidly becomes larger as the intended depth of a network increases. The architectures of the networks described above have largely been designed by hand, using expert knowledge in order to tweak parameters and hyperparameters to achieve the best possible results. This has unfortunately resulted in a high barrier-to-entry for the field, as this expert knowledge must be acquired before one can design effective networks. The process can be seen as a sort-of human-led stochastic optimisation, whereby human agents iteratively design and test new architectures, using knowledge gained through previous experiments or from others' experimental results. When applying to sub-field tasks, transfer learning implementations rely on existing pre-trained models due to the time/hardware constraints, difficulty in designing and training a new model, and the potential pitfalls in this process. This does, however, mean that the pool of available networks is very small, as one must rely solely on the published architectures. Therefore an automatic architecture optimisation process is required to allow these researchers to design architectures specific to the task at hand without needing a deep background knowledge in the area. The process must also be fast and usable on commodity hardware, in order to be available to those who need it the most.

In this work, we propose Swarm Optimised Block Architecture (SOBA), a system to perform evolutionary deep CNN model generation using an enhanced Particle Swarm Optimisation (PSO) model. The proposed model is able to conduct concurrent architecture optimisation and end-to-end training for the task of image classification. In other words, the proposed method both optimises and trains model architectures at the same time, allowing a 'one-click' approach to creating an effective model from a block-based skeleton architecture. Using this approach, we provide an effective way to optimise the architecture of CNNs for image classification through the formation of the optimisation problem within a constrained search space. We employ a skeleton architecture that is used as a minimal starting point by our optimisation system, which loosely follows the VGG architecture [6]. A modified PSO model with adaptive acceleration coefficients is used to perform the evolving deep CNN generation, with parameter sharing used to alleviate the enormous computational cost of fully training and evaluating each new architecture. The newly proposed cosine annealing mechanisms for adaptive search weight generation in the enhanced PSO model enable the search to balance between local exploitation and global exploration to carefully direct the architecture generation around the additional constraints introduced by the parameter sharing strategy. The nature of the search process in combination with the training of the network blocks themselves ensures that each block becomes much more robust, as it is trained and optimised to its best performance level, regardless of the structure of the rest of the network. Specifically, each block will be trained using backpropagation independent of the specific configurations of the other layers, with the weights shared between different model configurations. This strategy is contrary to hand-designed models which are wholly trained with the same model configuration. Fig. 1 represents the proposed Swarm Optimised Block Architecture (SOBA) model.

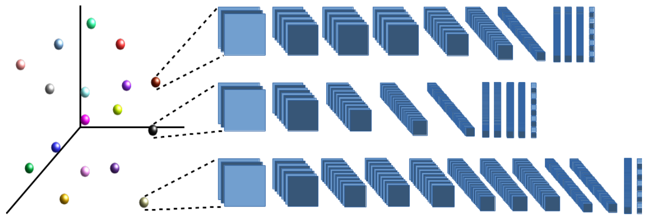


Fig. 1. The proposed Swarm Optimised Block Architecture where each particle represents a full, discrete convolutional architecture for image classification.

The research contributions are as follows:

1) We formulate the optimisation problem within a constrained search space through the creation of a bespoke objective function.
2) We employ a continual training method using a weight lookup table to alleviate the enormous computational cost of fully training and evaluating each new architecture.
3) An enhanced PSO model is proposed to perform the minimisation of our objective function with acceleration coefficients determined by a shifted cosine function to address the added constraints from our continual training method.
4) We apply a combined optimisation and training strategy to provide single-run, end-to-end optimisation with a final fine-tuning step.
5) Evaluated on the CIFAR-10 and CIFAR-100 image datasets, the proposed model shows superior classification performance on consumer hardware in reasonable time, over other related methods reported in the literature.

The proposed SOBA model allows for fast, effective *design & training* of simple VGG-style image classification models up to a high level of accuracy through stochastic exploration of a constrained euclidean search space, inspired by the simple and popular VGG architecture model. SOBA is designed and tested using consumer-level hardware with reasonable runtimes, providing an accessible method of architecture optimisation with huge potential for other complex computer vision tasks.

The rest of this paper is organised as follows. We discuss the background theory and related work in Section II, and present the proposed SOBA optimisation model for evolving deep CNN generation with adaptive PSO and weight sharing in Section III. We then provide system evaluation and experimental results in Section IV. Finally, we provide concluding remarks and discuss further directions in Section V.

II. BACKGROUND & RELATED WORK

A. ARCHITECTURE OPTIMISATION

Evidenced by recent research [6]–[9], CNNs have become exceedingly popular for solving computer vision problems owing to their ability to learn task-specific filters to extract the key information in an image. CNNs are usually comprised of multiple layers of computation that function together to form a network. When designing a CNN for classification, the typical goal is to embed the information found in the image into a fixed length vector, which can then be passed through fully-connected (linear) layers, or even another classifier, to finally output class probabilities. This is performed by cascading layers of convolutional operations with learned filters. There are a number of design and parameter choices that can be made for each layer, as well as for the connections between the layers. The traditional approach is a purely hierarchical model, where each layer feeds its output feature maps into the next layer sequentially until the final layer is reached. Recent works have, however, explored the possibility of considering the connections between the layers as a Directed Acyclic Graph (DAG), whereby each layer can connect to any number of subsequent layers. This approach is motivated by the success of residual or skip-connections [9], which proved effective at maintaining global features throughout the network by directly passing earlier feature maps to later layers in a hierarchical network. Besides the connections between the layers in the network, each layer itself requires a number of choices to be made pertaining to its parameters and functionality. As previously mentioned, these choices have traditionally been made based on prior knowledge and intuition, with many combinations being thoroughly tested before an optimal architecture is found. There has been a large amount of recent interest in the task of designing architecture search strategies to replace this human-led trial and error process and provide an effective method to automatically design optimal architectures and associated hyper-parameters. As an example, Stanley and Miikulainen [10]–[12] explored various methods for automating the process of designing neural network architectures. Their method, called NeuroEvolution of Augmenting Topologies (NEAT), took the form of a Genetic Algorithm (GA) which could 'grow' architectures from a simple starting point. Stanley *et al.* [13] also explored training neural networks via similar evolutionary methods, although this has thus far not proved to outperform back-propagation. There has been some recent work to extend the NEAT methodology to build deep learning architectures, as the size and complexity of CNN models present further challenges [14]–[17]. More recently there have been a number of experiments using reinforcement learning (RL) for automated architecture design [18]–[22], although these techniques have very recently been surpassed by relatively more simple evolutionary methods [23]–[26].

Real *et al.* [25] demonstrate evolution of image classifiers using relatively few constraints. They inherit weights between evolutionary generations for more effective training with less wasted processing time for re-training layers of the same shape/depth in the architecture. Their model uses a form of *tournament selection* whereby an initial population of models is trained, then individual pairs are compared and the weaker of the pair is terminated. On the contrary, the fitter solution of the pair is selected for reproduction to yield an offspring solution via a mutation operation. This offspring model is then trained and evaluated and becomes a member of the overall population and the above process is repeated. Mutations are chosen from a population of eleven hand-picked operations (such as Insert-convolution, Alter-learning-rate, and Alter-filter-size) intended to mimic the steps a human would take when attempting to improve the architecture. Their work could also be extended to yield "hybrid evolutionary–hand-design methods" in the future.

Brock *et al.* [27] train a 'hypernetwork' model to generate weights for a given architecture, in order to effectively evaluate an architecture's validation performance. They can then sample architectures to obtain validation performance estimates using the weights generated by the hypernetwork. Once a number of architectures are generated and sampled, they take the architecture with the best estimated validation performance and fully train it in the usual way, in order to test its performance on the test set.

Zoph and Le [19] use a recurrent neural network (RNN) as a controller network to generate potential network architectures. These architectures are trained for a large number of epochs and then evaluated on a validation set to determine a score for the child architecture. The controller architecture is trained using policy gradients, particularly the REINFORCE algorithm [28]. Owing to the enormous cost of training for evaluating each child architecture, an asynchronous, distributed training process is used whereby many child architectures are trained concurrently using multiple workers. This approach is common for current architecture optimisation works, although it does require huge resources not commonly available to research teams.

Negrino and Gordon [29] propose a method that allows researchers to describe search spaces, which can then be effectively explored using a tree-search strategy. They demonstrate the effectiveness of Monte Carlo tree search (MCTS) and sequential model-based optimisation (SMBO) over random search when traversing their defined search spaces.

Baker *et al.* [18] use Q-Learning to maximise the overall expected reward by modelling architecture optimisation as a Markov Decision Process (MDP). Their agent iteratively selects layers to add to the architecture, until it reaches a stop-state. Initially the agent is allowed to sample architectures with a random walk, giving the opportunity to explore the search space before targeted optimisation. They allow the agent to continue this random behaviour to a certain extent, until the end of the process, producing a number of different models which can then be evaluated as an ensemble in order to improve final performance.

Besides the above, Real *et al.* [30] perform the first controlled comparison of reinforcement learning and evolutionary methods for architecture optimisation and found that "regularized evolution consistently produces models with similar or higher accuracy across a variety of contexts without need for re-tuning parameters".

Wang *et al.* [31] proposed a method using the PSO model to optimise architecture, with an embedding scheme derived from IP address allocation. Unfortunately, their proposed system was unable to effectively deal with the issue of function-evaluation cost, whereby each particle evaluation is prohibitively time and resource expensive when performed as a full training-evaluation process. They implemented early evaluation, where each individual was partially trained on the training dataset and then evaluated on the validation set for each fitness function evaluation. This is effective at reducing the time required for each function evaluation, but results in unreliable fitness scores due to the fact that each architecture has not been trained to its full extent when evaluating and it is impossible to know when its performance will plateau. Their work thus showed limited capabilities to be tested on any datasets other than the decidedly small MNIST dataset. Their system performed favourably in comparison with the other methods, although it is notable that they did not compare against any key deep learning architectures, or indeed any systems proposed in recent years.

The drawbacks of the vast majority of these works are that they focus on the optimisation of the architecture as a distinct search problem, and separate it from the actual training of the model for the specific task. This results in these methods having two distinct steps, i.e. resource-heavy optimisation, followed by lengthy training in order to generate the model with the most promising performance. This motivates the proposed work in this research to jointly optimise and train an identified deep learning model to enhance both system performance and computational efficiency. The proposed SOBA optimisation model also employs a weight sharing mechanism to overcome the function-evaluation cost issue encountered by [31] and discussed above.

B. PARTICLE SWARM OPTIMISATION

Particle Swarm Optimisation (PSO) [32] is a stochastic optimisation technique that relies on a population X of m individuals, each with a specific position in the search space defined by a fixed-length vector $\mathbb{R}^n$. Each position in the search space represents a distinct set of parameters to an objective function f. The fitness of an individual particle represents the result of evaluating the objective function f with the position of the particle as parameters. The goal of PSO is to minimise or maximise the objective evaluation by finding the best overall particle position $\min_{\boldsymbol{x}} f(\boldsymbol{x})$ or $\max_{\boldsymbol{x}} f(\boldsymbol{x})$. The individual particles in the population are initialised with random positions in the search space, usually by drawing their values from a uniform distribution U, bounded by defined upper (b_u) and lower (b_l) bounds. The particles are then iteratively evaluated and conduct the search process by following personal and global best solutions in order to attain global optimality. Specifically, as the particles are moved around the search space, the best positions found so far, along with their fitness scores, are stored for each individual particle. These are referred to as the 'local best' solutions. The best solution of the overall swarm is referred to as the 'global best' solution and indicates the best set of parameters that the algorithm has as-yet found for the objective function. (1) & (2) denote the velocity and position updating operations for each particle respectively.

$$V_i^t = wV_i^{t-1} + c_1r_1(P_i - X_i^{t-1}) + c_2r_2(P_g - X_i^{t-1}) \tag{1}$$
$$X_i^t = X_i^{t-1} + V_i^t \tag{2}$$

where $c1$ and $c2$ denote acceleration coefficients, and r_1 and r_2 are random vectors drawn from $U(0, 1)$ to introduce stochasticity. P_i and P_g represent the personal and global best solutions respectively, with w as the inertia weight. X_i^{t-1} and V_i^{t-1} represent the position and velocity of the particle i from the previous $(t-1)$ iteration, respectively. The process is repeated over a defined number of iterations, or until a stop criterion is met. The pseudo-code of the PSO model is provided in Fig. 2.

```
 1: $X \leftarrow U(b_l, b_u)$
 2: $P \leftarrow X$
 3: $G \leftarrow X_{argmin(f(X))}$
 4: $V_{max} \leftarrow \lambda(b_u - b_l)$
 5: $V_{min} \leftarrow -V_{max}$
 6: $V \leftarrow U(V_{min}, V_{max})$
 7: for $t \leftarrow 0, \ldots, T$ do
 8:     for $i \leftarrow 0, \ldots, m$ do
 9:         $V_i \leftarrow wV_i + c_1r_1(P_i - X_i) + c_2r_2(G - X_i)$
10:         $X_i \leftarrow X_i + V_i$
11:         if $f(X_i) < f(P_i)$ then
12:             $P_i \leftarrow X_i$
13:         if $f(X_i) < f(G)$ then
14:             $G \leftarrow X_i$
```

Fig. 2. The original PSO algorithm.

The velocities of the particles are updated by using three components, i.e. the existing velocity, the distance between the current position and the best position of this particle so far (local best), and the distance between the current position and the swarm leader (global best). Each of the three main components thus described is weighted to control the effect it has on the resulting velocity and position updates. These search weights take the form of w, c_1, and c_2, where w controls the impact of the previous velocity, c_1 controls the effect of the local best, and c_2 controls the effect of the global best. The standard PSO model employs pre-determined, fixed search weights, thereby defining the magnitude of the effect of the previous velocity, and the local and global bests on the resultant velocity.

The 'No Free Lunch' theorems [33] suggest that "if an algorithm performs well on a certain class of problems then

Table 1. The structure of the convolutional architecture to be optimised when applied to CIFAR-10 and CIFAR-100.

layer type	input	convolutional	convolutional	convolutional	convolutional	fully-connected	output
depth (no. f-maps)	3	64	128	256	512	4096	10 | 100
spatial-size (of f-maps)	32×32	16×16	8×8	4×4	2×2	1	1
filter-size	n/a	3×3	3×3	3×3	3×3	n/a	n/a
block-size (no. layers)	n/a	$1 \dots 10$	$1 \dots 10$	$1 \dots 10$	$1 \dots 10$	$1 \dots 10$	n/a

it necessarily pays for that with degraded performance on the set of all remaining problems''. This is widely considered to suggest that optimisation algorithms can be necessarily tuned towards a specific problem, without the burden of being required to prove their general (or average) performance over the set of all problems. Following this paradigm, many works [34], [35] successfully focus on improving the performance of PSO for their own specific optimisation tasks by addressing their unique constraints through modification of the PSO algorithm. For example, in order to overcome premature convergence of the original PSO model, Mirjalili *et al.* [35] proposed and tested a number of 'adaptive' acceleration coefficients, whereby the acceleration coefficients become a function of the current iteration and the overall number of iterations, resulting in search weights that change throughout the course of the optimisation process. Intuitively, they found that allowing the individual and social behaviours of the particles to change, as the optimisation process approaches global optimality, results in improved overall performance and convergence speed. Using adaptive search weight techniques therefore allows for tailoring of the search behaviour throughout the optimisation process, in comparison with using fixed parameters.

III. METHODOLOGY

A huge variety of strategies could be employed for swarm-based architecture optimisation. One intuitive strategy is to traverse through a large number of distinctive key architecture decisions, where each individual could represent a discrete architecture by determining distinct filter sizes, dilation factors, strides, pooling kernel size, etc. for every layer in the network. Our initial experiments with this approach used an $\mathbb{R}^{5j+2k}$ dimensional vector to represent each architecture where the dimensionality of the search space was $(2 \times 8^4 \times 16^2 \times 32^2 \times 126)^j \times (2 \times 4096)^k$. Here, j and k represent the number of convolutional layers and the number of fully-connected layers respectively. Using our standard values for j & k of $j = 7$ and $k = 2$, this represents an approximate total number of discrete, potential models of $\sim 7.1 \times 10^{87}$. Such a search space allowed for many configurations of architecture but proved incredibly difficult to efficiently navigate owing to its enormous size, especially compounded by the constraints introduced by the interactions and conflictions between the individual scalar control values in a position vector. Many combinations of the scalar control values could result in physically impractical architectures, which proved impossible to create given realistic hardware constraints. This approach thus proved unworkable, even when heavily penalising these failures inside the optimisation process, as the model would often resort to creating the simplest possible (unconstrained) architecture.

A. SEARCH SPACE/OBJECTIVE FUNCTION DESIGN

We started with the idea of a simplified version of the search space described above, by looking at existing effective convolutional network designs and distilling them down into their core components. VGG-16 [6] is a popular CNN architecture to start with when approaching a new computer vision task owing to its simplicity, combined with its proven effectiveness. We construct a restricted search space around the core concept of the VGG family of networks by taking the concept of downsampling the width and height of the network, whilst simultaneously increasing the number of feature maps created, and build a 'skeleton' of blocks where each block ends with the proposed down/up-sampling operation. We then have a model for the gradual decrease in spatial size and increase in number of feature maps as the network progresses, which we hypothesise to be the key to the success of the network. The decrease in spatial size from the downsampling operations can also be seen as a gradual increase in receptive field size of the filters, beyond the usual increases, as each subsampling operation increases the receptive field of the next convolutional layer by condensing the feature map. For simplicity, for all convolutional operations we use a filter size of (3×3) with stride (1×1) and padding (1×1) to maintain the feature map sizes between convolutional layers. For each convolutional block, we produce a number of feature maps equal to 2^{ϵ} where ϵ takes the values $[6, 7, 8, 9]$ for blocks $[0, 1, 2, 3]$ respectively. For the downsampling layers, we use max pooling with kernel size (2×2) and stride (2×2). If we were to scale up the experiments to larger images, we would look to add additional subsampling layers to decrease the spatial size of the feature maps, increase receptive field sizes, and create more tuneable blocks in our architecture. The skeleton architecture which is used as the starting point for the deep CNN evolution can be seen in Fig. 3, with further detail in Table 1.

Next, we define a method for mapping a vector of integers to a full convolutional architecture by 'stacking' layers in each block, according to the value in the specific index of the architecture vector. In this way, we can explore an n dimensional space, where n represents the number of tuneable blocks in our architecture. We then frame the task of generating the architecture of a model as a minimisation of an objective function $f(\boldsymbol{x})$ (defined in Fig. 4), where $\boldsymbol{x}$ represents an abstraction of network architecture into a single

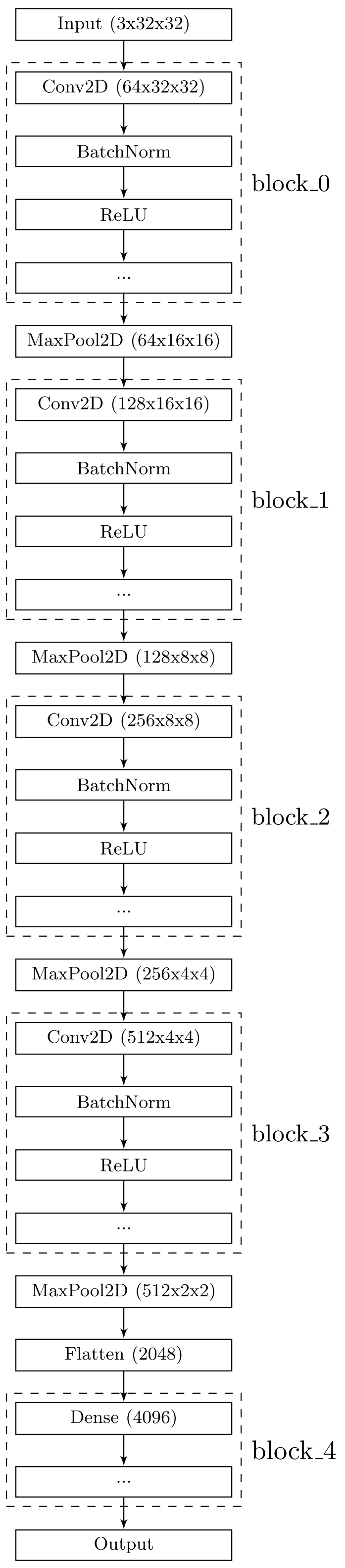


Fig. 3. The skeleton architecture used in the proposed system.

point in the navigable multi-dimensional search space and $f(\boldsymbol{x})$ represents the error rate of the model when evaluated on the validation set. This involves discovering the optimal value of $\boldsymbol{x}$ which produces the minimal error rate when evaluated using the fitness function, as shown by (3).

$$\operatorname*{argmin}_{\boldsymbol{x}} f(\boldsymbol{x}) = \{\boldsymbol{x} \mid \boldsymbol{x} \in \boldsymbol{S} \wedge \forall \boldsymbol{y} \in \boldsymbol{S} : f(\boldsymbol{y}) \geq f(\boldsymbol{x})\} \tag{3}$$

where,

$$\boldsymbol{S} \subset \mathbb{N}^n \mid \forall \boldsymbol{x} \in \boldsymbol{S} : 0 \leq \{x_0, \ldots, x_{n-1}\} < 10 \tag{4}$$

in actual fact, we explore the search space using,

$$\boldsymbol{S} \subset \mathbb{R}^n \mid \forall \boldsymbol{x} \in \boldsymbol{S} : 0 \leq \{x_0, \ldots, x_{n-1}\} < 10 \tag{5}$$

and rely on the implicit integer-cast to function as a form of regularisation by only allowing large, or multiple small, movements to modify the structure, similar to the approach taken by [25]. We do this by min-max scaling the position values into our desired range and then converting into integers representing the number of layers to add to each block. This is simply performed by multiplying each position value by the upper bound of the range, as we use the values 0 and 1 as the lower and upper bounds for optimisation respectively. In order to optimise the objective function, we use the previously described evolutionary optimisation technique, i.e. the proposed adaptive PSO model, to efficiently explore the search space defined by our objective function.

B. ENHANCED PARTICLE SWARM OPTIMISATION

We propose an enhanced PSO model for optimal deep CNN model generation. It considers each individual architecture in the search space as a position in an n-dimensional space where n represents the number of distinct blocks in our skeleton architecture. Instead of using fixed acceleration coefficients as in the original PSO model, adaptive search parameters based on linear and non-linear functions are proposed. Four new strategies have been applied for the coefficient generation: (1) linear functions with an equal crossover in the centre, (2) cosine functions with an equal crossover in the centre, (3) cosine functions with a later crossover, and (4) cosine functions with no crossover. These strategies are described in further detail in Section III-D.

We start by initialising a population of m individual particles as random positions in the search space, where each dimension in each particle is drawn from a uniform distribution:

$$X = \mathcal{U}(b_l, b_u) \tag{6}$$

where b_l and b_u are the lower and upper boundaries of the search space, respectively. We then initialise the velocities of each of the particles:

$$V_i = \mathcal{U}(v_{min}, v_{max}) \tag{7}$$

where,

$$\begin{aligned} v_{max} &= \lambda(b_u - b_l) \\ v_{min} &= -v_{max} \end{aligned} \tag{8}$$

and we use a value of 0.2 for λ in all experiments based on best practice.

```
1: function OBJECTIVEFUNCTION($position$):
2:     $model \leftarrow InitModel(position)$    ▷ Initialise the model using the particle position vector
3:     if $weight\_sharing$ then
4:         $model.weights \leftarrow BlockLookup(position)$    ▷ Construct the weight lookup keys from the model structure
5:                                     ▷ Determine the weight lookup strategy
6:                                     ▷ Iteratively check the weight lookup table for existing weights
7:                                     ▷ Load the existing weights into the model
8:     $Train(model)$    ▷ Train the model for a single epoch (or a defined number of batches)
9:     $error\_rate \leftarrow Validate(model)$    ▷ Evaluate the model on the validation set to determine the error rate
10:    if $weight\_sharing$ then
11:        $BlockStore(position, model.weights, error\_rate)$    ▷ Iteratively update the weights in the weight lookup table
12:    return $(error\_rate/100)$    ▷ Return the error rate as the fitness for this function evaluation
```

Fig. 4. The bespoke objective function to be optimised.

Once the swarm has been initialised, the optimisation process can begin. It starts with updating the inertia weight and both acceleration coefficients according to the specific strategies chosen. Next, each particle X_i is processed with the following steps. First, the velocity of the particle is updated using the search weights and the distances between the current position and the local and global best positions as defined in (1). Using the velocity, the new position of the particle is calculated based on the previous position as illustrated in (2). The fitness of the particle is evaluated using the objective function provided in Fig. 4. The fitness score of X_i is compared against those of the previous personal best position P_i and the global best solution, respectively. The local best position is updated according to (9).

$$P_i = \begin{cases} X_i, & \text{if } f(X_i) < f(P_i) \\ P_i, & \text{otherwise} \end{cases} \tag{9}$$

Whilst the global best is similarly updated according to (10).

$$G = \begin{cases} \underset{P_i}{\operatorname{argmin}}\, f(P_i), & \text{if } \underset{f(P_i)}{\min} < f(G) \\ G, & \text{otherwise} \end{cases} \tag{10}$$

In practice, in order to avoid unnecessary overhead we store the fitness values for each evaluation for the score comparisons, rather than re-calculating the fitness for each updated particle position. This process then repeats over all particles, and all iterations, until a certain stop criterion has been met, i.e. the maximum number of iterations. Once the iterations have completed, the final output of the system is the global best position G and its fitness value $f(G)$, representing the best arguments to minimise the objective function ($\operatorname{argmin}_{\boldsymbol{x}} f(\boldsymbol{x})$) and the fitness value respectively. As mentioned earlier, Section III-D outlines several linear and cosine annealing mechanisms that are proposed for adaptive acceleration coefficient generation in the proposed PSO model to enable the search to balance well between local exploitation and global exploration. Fig. 5 demonstrates the proposed, customised PSO, with Fig. 4 comprising our bespoke objective function. In comparison with [31], we also employ weight sharing strategies to overcome the function-evaluation cost constraints, which are introduced below.

```
1: function PSO($x$):
2:     $f \leftarrow ObjectiveFunction()$
3:     $m \leftarrow population$
4:     $n \leftarrow dimensions$
5:     $b_l \leftarrow 0$
6:     $b_u \leftarrow 1$
7:     $X_{m,n} \leftarrow U(b_l, b_u)$
8:     $P \leftarrow X$
9:     $G \leftarrow X_{argmin(f(X))}$
10:    $V_{max} \leftarrow \lambda(b_u - b_l)$
11:    $V_{min} \leftarrow -V_{max}$
12:    $V \leftarrow U(V_{min}, V_{max})$
13:    $mode \leftarrow ['fixed'|'linear'|'cosine'|'AGPSO1']$
14:    for $t \leftarrow 0, \ldots, T$ do
15:        $lr \leftarrow SetLearningRate(lr, t)$
16:        $w, c_1, c_2 \leftarrow SetSearchWeights(T, t, mode)$
17:        for $i \leftarrow 0,\ldots,m$ do
18:            $V_i \leftarrow wV_i + c_1r_1(P_i - X_i) + c_2r_2(G - X_i)$
19:            $X_i \leftarrow X_i + V_i$
20:            if $f(X_i) < f(P_i)$ then
21:                $P_i \leftarrow X_i$
22:            if $f(P_i) < f(G)$ then
23:                $G \leftarrow P_i$
```

Fig. 5. The proposed PSO model with adaptive acceleration coefficients.

C. CONTINUAL TRAINING/PARAMETER SHARING

Due to the nature of our fitness function, evaluating fully for each particle requires inhibitive amounts of time and resources as each architecture must be trained and validated to obtain a fitness score. We reconcile this constraint through the use of early evaluation of the function on the validation set, similar to the approach used by [31]. However, using early evaluation solely does not provide a realistic view of the performance of the particle, as an architecture may train very successfully initially but later plateau before reaching an acceptable level of accuracy. In contrast to [31], we additionally use a form of parameter sharing, allowing new architectures to inherit the weights from previous particles when performing their fitness function evaluations. This allows the architectures to train alongside the optimisation process, with each block keeping track of its own weights. In this way, the fitness function evaluations are representative of the performance of the individual particles at all stages of the optimisation process, as their training progresses and performance increases throughout. Real *et al.* [25] implemented weight inheritance as a feature of the inheritance process of their GA, allowing weights for layers with matching shapes to be inherited, or not depending on the specific mutation. We take this a step further by considering weight sharing as a means of continually training all of the models jointly. This continual training allows us to consider the optimisation process as the bulk of the training of the final model, with the fitness function evaluations becoming more accurate as training progresses and the shared weights are trained. This allows us to quickly navigate the search space, following the path that leads to the greatest performance increases as we go. We combine this approach with a previously mentioned annealing strategy applied to the PSO search weights. These adaptive search parameters in PSO enable the search to favour local exploitation in early iterations and global exploration in final iterations. In this way we ensure that we avoid prematurely optimising to a local minimum before the possible architectures have been trained for a reasonable number of iterations. Without this strategy, it is likely that the large improvements in error rate that can be seen with the first few training iterations would result in a rapid clustering of all of the particles into one area after following the global best solution.

We jointly train the population of continually evolving network architectures by maintaining a lookup table of convolutional filter parameters and fully-connected layer weights. The lookup table consists of a simple key-value store, where the key takes the form of a string concatenation of the integer block number in the architecture, with the integer size of the block (i.e. the number of layers in the block minus 1; zero indicates a single layer) separated by a period. A key thus takes the form '$\psi.(\omega - 1)$' where ψ is the specific number of the block in the skeleton architecture, and ω is the number of layers in the block. Each value in our key-value store is itself a smaller key-value store, consisting of two key-value pairs, i.e. the best performing parameters, and the last used parameters, for each distinct block & size. This allows us to check if a specific block has been constructed to a certain size before, and inherit the weights of that block, thereby gradually training the individual blocks as we explore the architecture search space. Initially we experimented with storing the best performing weights for each particle position & block size (with the error-rate of the model as performance indicator). This approach has the downside of limiting the exploration of the model since it ensures that any training run that does not increase performance by the end of the run will be discarded, in favour of the original parameters. The model is then limited in its exploration capability. To alleviate this issue, we also store the last known weights of each block number & size. We can then choose whether to continue with the last known weights, or to select the best seen so far. We control the selection of existing parameters through a weight value β which controls how likely we are to select the best weights over the last known weights by comparing with a random value between 0 and 1. This process can be seen in Fig. 6.

```
1: function LOOKUP($\beta$)
2:     if $\beta > U(0,1)$ then
3:         return $best$
4:     else
5:         return $last$
```

Fig. 6. The weighted parameter lookup function.

We experimented with a number of strategies for β. Initially we chose to always use the best weights that have been seen before for each block when performing the inheritance process. However, as previously described, this led to a cycle of limited exploration, whereby a model would become stuck repeatedly retraining parameters but never achieving a better validation score, therefore discarding its progress. Next we experimented with a fixed 50:50 chance of inheritance from best or last, allowing for exploration but also promoting superior results with an even chance. We found this approach to perform better than just using the best results but we still hypothesised that there would be a superior approach. We then implemented an annealing schedule for β, in order to allow the chance of inheriting from best, rather than last, to gradually increase from 0% to 100%. This schedule can be seen in (11).

$$\beta = 1 + \frac{cos(\pi(1 - \frac{\phi}{\Phi}))}{2} \tag{11}$$

where ϕ and Φ represent the current and total number of fitness function evaluations respectively. Φ can be calculated by $m + m \times T$, where m represents the swarm population and T represents the number of iterations for the optimisation algorithm. Finally, we experimented with only using the last values stored for each block, regardless of their fitness value. Surprisingly, we found this approach to be the most performant, which led to its subsequent use in all following experiments. The four main techniques that were tested can be seen in Fig. 7 and the experimental results can be seen in Table 2. When storing the weights for a specific block in the architecture, we transfer the weight tensors into RAM in order to save the on-board GPU memory for larger batch sizes and larger potential network architectures.

Once our optimisation process has concluded, we need to finalise the model for testing. In related work, this is usually performed by discarding any parameters learned during the optimisation process, taking the best architecture discovered, and training it completely from scratch. We use a different

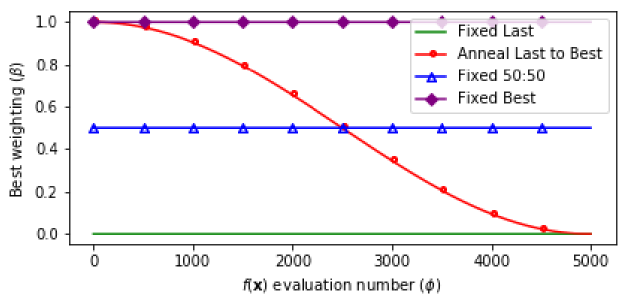


Fig. 7. The different weighting strategies for parameter lookup table access (β for the lookup function) against the number of fitness function evaluations (ϕ).

Table 2. Error Rates (%) for Different Weight Lookup β Strategies.

Method	CIFAR-10
Fixed Best	9.09
Fixed 50:50	7.90
Cosine Annealing	5.28
Fixed Last	**4.78**

approach in order to combine the optimisation process with the final model training, so as not to waste the training progress thus far. We construct a fine-tuning dataset from the previous training and validation sets by combining them into one large training dataset. We then fine-tune the best performing model from the optimisation process on this dataset for a small number of epochs in order to alleviate the effect of any overfitting introduced by the optimisation process. Once we have completed our fine-tuning on the larger training set, we are able to test the model on the test set and report the final results of the model without re-creating or re-training the model after the optimisation process.

D. ADAPTIVE PSO SEARCH WEIGHTS

Using the process outlined above, we are able to effectively explore our constructed search space in reasonable time using the proposed PSO model. In the original PSO model, the search weights for cognitive and social components are fixed when updating the velocity of each particle, which is then used to update the position. Specifically, the coefficients are set to give fixed weightings to both the local and global best solutions for each particle when performing the update. Our continual training method means that initially, we expect to see large gains no matter where a particle moves, owing to the initial training of the networks up to a reasonable level of performance. Because of this, it is desirable that each particle can be allowed to explore its own space initially, rather than move towards the global best. This will ensure that the particles perform useful exploration in these initial stages, by moving towards the area with the greatest improvements around themselves. We can ensure that this happens by setting the ratio between the local search weight and the global search weight to a high value. This could be achieved by setting the local search weight to a fixed, high value and the global search weight to a fixed, low value. However, later in the training process, we expect that the performance gains from each iteration will slow down significantly, as the networks come closer to achieving their optimal performance. At this point, it is desirable that we obtain the best performing, single network from the population of individuals. It follows then, that rather than allowing the particles to explore around their own space, it would be beneficial to exploit one of the key functions of PSO, by allowing the particles' positions to trend towards the position of the best performing particle, in the hopes that they can explore together around the position and achieve even better performance. At this later iteration stage, we would like the previously described ratio between local and global search weights to reverse, promoting a high ratio of global to local. This means that a fixed ratio for the whole optimisation process is less than ideal, as this reversal process cannot happen. In order to achieve the above, desired outcome for the search weight strategy and overcome the premature convergence of the original PSO model, we explored the concept of adaptive acceleration coefficients, where the search weights can change depending on the current iteration number.

Motivated by [35], several adaptive acceleration coefficient generation schemes are subsequently explored in this research. Fig. 8 shows the explored mechanisms for coefficient generation, i.e. (1) fixed acceleration coefficients in Fig. 8(a), as the baseline where $c1 = c2 = 0.9$ and $w = 0.6$; (2) a linear AGPSO1 schedule borrowed from [35] (in Fig. 8(b)), as another baseline; (3) a proposed linear crossover function in Fig. 8(c); and (4)-(6) proposed nonlinear cosine annealing schedules with equal, late, and no crossover in Fig. 8(d), Fig. 8(e), and Fig. 8(f) respectively. The above adaptive search strategies are employed with the goal of allowing the particles to explore in their local search space initially, so as not to prematurely head towards the global best during the initial training stages. This could happen due to the large increases in accuracy that are present in the initial training steps of a convolutional architecture, combined with our continual training method. We found that maintaining a reasonably high level of local exploration during the optimisation process was effective to allow the particles to efficiently cover the search space, as can be seen in Fig. 9 as well as Fig. 10 (with detailed explanation provided later in this section). The proposed linear and nonlinear adaptive search weight functions are provided in (12) and (14), respectively. Fig. 11 shows the pseudo-code for the overall adaptive acceleration coefficient generation function, with the accompanying graphs in Fig. 8. First of all, the proposed linear coefficient generation function is defined in (12):

$$\begin{aligned} c_1 &= Q - \frac{t}{T}(Q - q) \\ c_2 &= q + \frac{t}{T}(Q - q) \end{aligned} \tag{12}$$

where t refers to the current iteration number, T refers to the total number of iterations to be performed for the optimisation

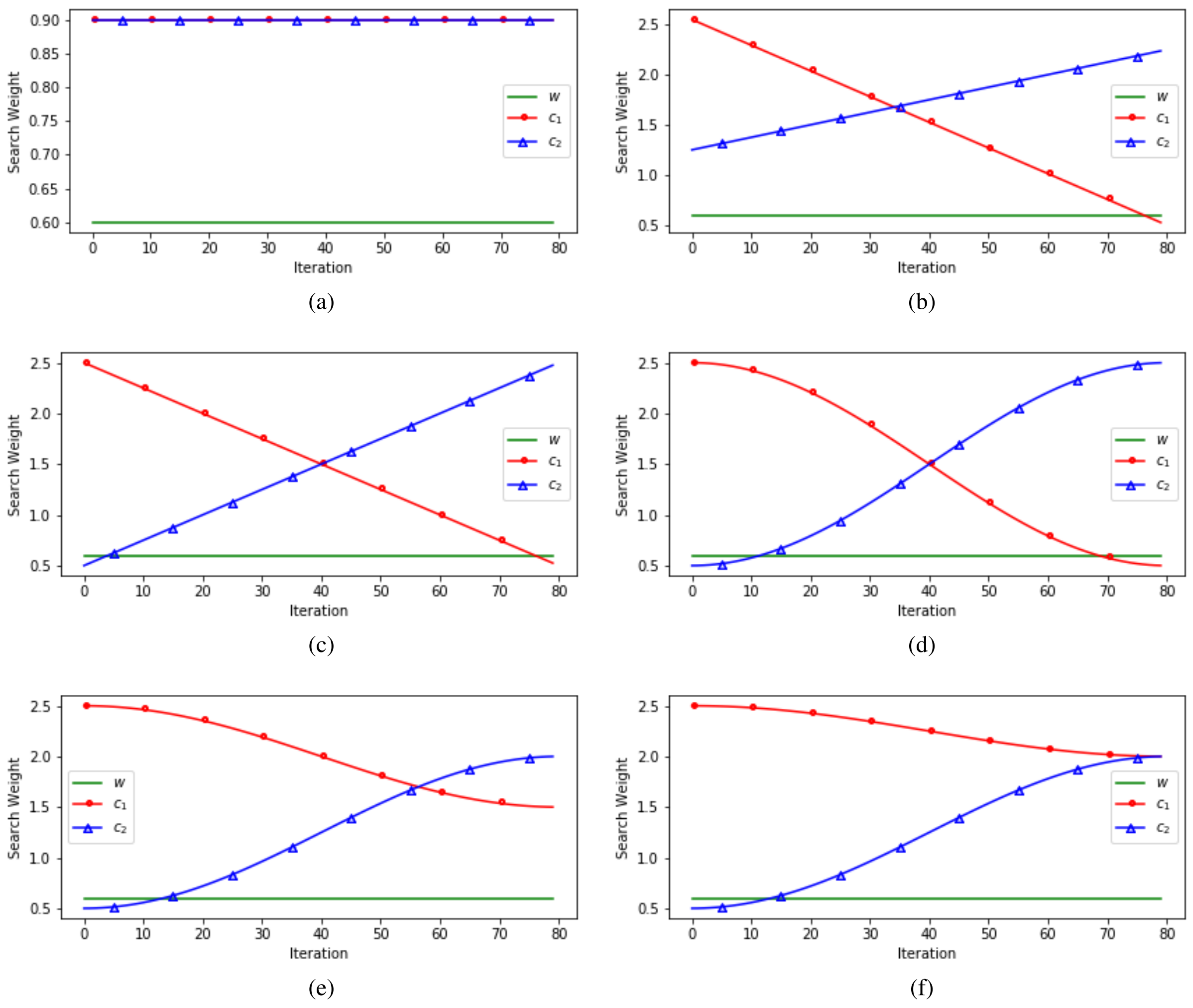


Fig. 8. PSO search weights by iteration. (a) Original PSO (fixed). (b) AGPSO1. (c) Linear. (d) Cosine Equal Crossover. (e) Cosine Late Crossover. (f) Cosine No Crossover.

run, q refers to the lower bound for the search weight, and Q refers to the upper bound for the search weight. A baseline adaptive weight generation strategy of AGPSO1 from [35] is defined below:

$$\begin{aligned} c_1 &= t\frac{1}{T} + 1.25 \\ c_2 &= t\frac{-2.05}{T} + 2.55 \end{aligned} \tag{13}$$

Motivated by (13), in order to adapt the weights more gently towards the extreme values, a smoother annealing schedule is proposed in this research, based on the cosine function. (14) denotes the proposed cosine search weights and variants:

$$\begin{aligned} c_1 &= q + \frac{Q-q}{2}cos(\pi(1-\frac{t}{T})) + 1 \\ c_2 &= q + \frac{Q-q}{2}cos(\pi\frac{t}{T}) + 1 \end{aligned} \tag{14}$$

where the variants are created by choosing different values for q and Q for c_1 and c_2. The Cosine Equal Crossover variant (Fig. 8(d)) is created using $q = 0.5$, $Q = 2.5$ for both c_1 and c_2. The Cosine Late Crossover variant (Fig. 8(e)) is created using $q = 1.5$, $Q = 2.5$ for c_1 and $q = 0.5$, $Q = 2.5$ for c_2. The Cosine No Crossover variant (Fig. 8(f)) is created using $q = 2.0$, $Q = 2.5$ for c_1 and $q = 0.5$, $Q = 2.5$ for c_2.

As mentioned above, Fig. 11 shows the pseudo-code of the adaptive weight generation over iterations, which are then used to update the velocities of the particles using (1).

With fixed search weights, the behaviour of particles with respect to local or global exploration remains the same throughout the optimisation process, favouring whichever is given a higher weighting, or neither if they are equally weighted. Using an annealing schedule allows us to introduce different search behaviour trends on local and global exploration that evolve over the course of the optimisation. Specifically we utilise this for the linear, AGPSO1, and cosine schedules to initially favour local exploration around each particle's own best position, and gradually trend towards focusing on global exploration, towards the best particle position found so far. We do this to prevent our particles from quickly abandoning their local search space in favour

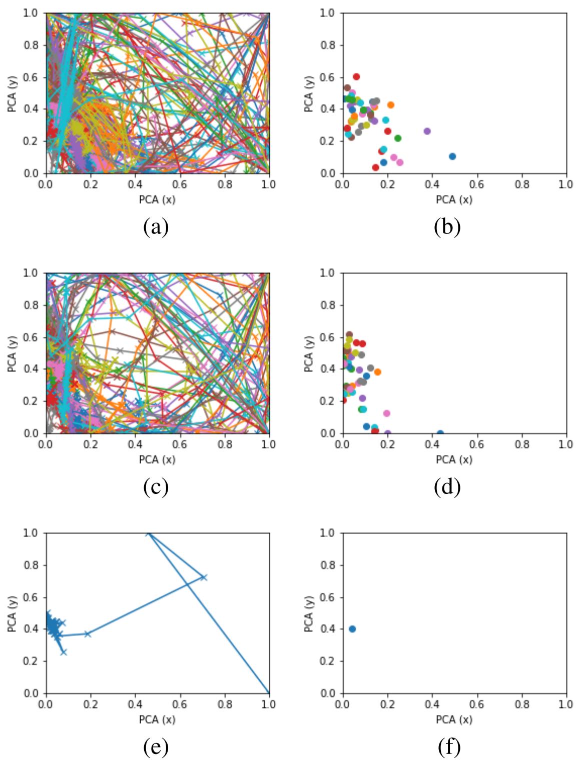


Fig. 9. **Example particle movements over time (PCA for dimensionality reduction to display particle positions on two axes). (a) All Particle Positions. (b) Last Particle Positions. (c) Local Best Positions. (d) Last Local Best Positions. (e) Global Best Positions. (f) Last Global Best Positions.**

Table 3. **Error rates (%) for different search weight strategies.**

Method	CIFAR-10
Original PSO (fixed)	5.38
AGPSO1 [35]	5.84
Linear Crossover	5.78
Cosine Equal Crossover	5.14
Cosine Late Crossover	**4.78**
Cosine No Crossover	10

of pursuing the best position, and thereby all becoming stuck in the same local minima. Fig. 10 demonstrates the effects of the late crossover cosine search weight strategy on the local bests of each particle and the overall global best. The x and y axes represent the particle positions projected into two-dimensional space using Principal Component Analysis (PCA). The z axis represents the fitness value for the particle after being evaluated on the validation set. Fig. 10(a) shows how initially the particles explore their own space, improving their fitness scores but not converging on a single location, as well as how they begin to converge on the x and y axes later as the search weights begin to favour following the global best. This can also be seen in the positions of the local and global bests in Fig. 10(b) and Fig. 10(c), which eventually converge around a single point after gradually narrowing focus and improving fitness scores.

The test results for the different strategies for adaptive coefficient generation can be seen in Table 3.

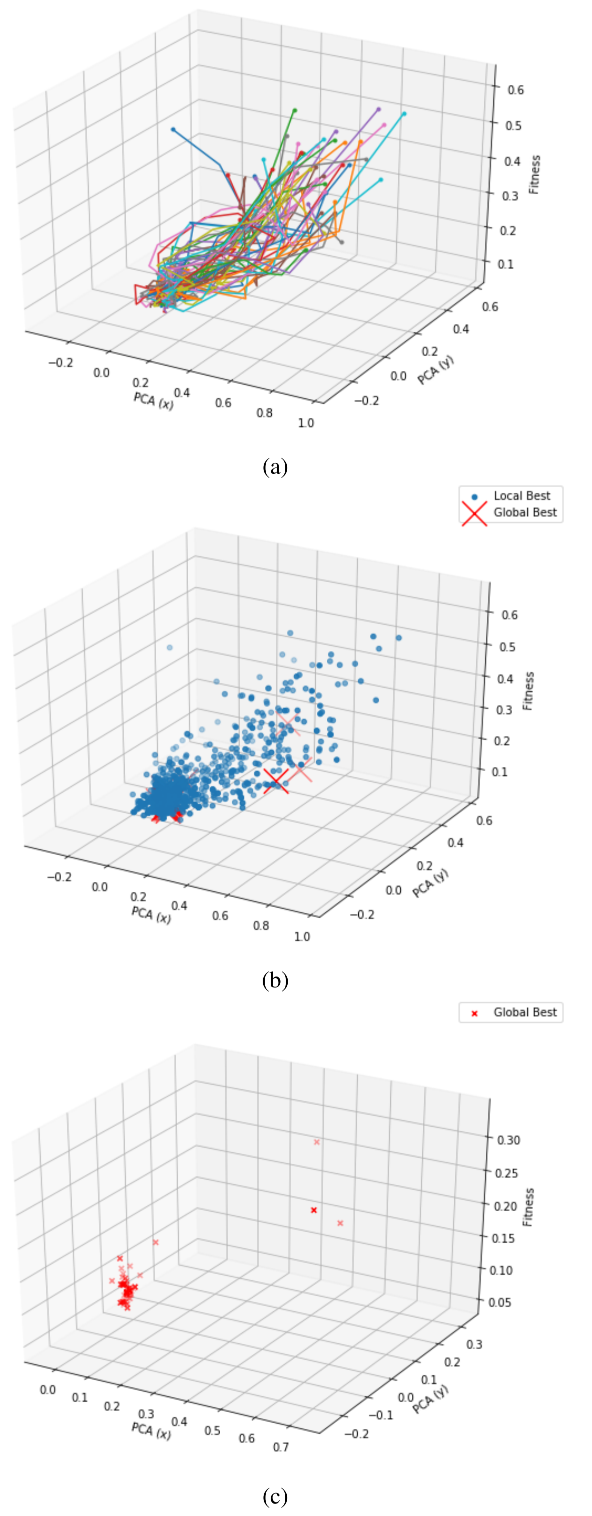


Fig. 10. **Example local and global best positions over time (PCA for dimensionality reduction to display particle positions on two axes). (a) Local best positions & their fitness scores by iteration for all 50 particles (point indicates start position). (b) All local & global best positions & their fitness scores. (c) Global best positions & their fitness scores.**

E. HYPERPARAMETERS

We use the excellent PyTorch library [36] for all deep learning implementation and training. Each particle fitness function

```
function SETSEARCHWEIGHTS($T, t, mode$)
    if mode is *fixed* then
        $w \leftarrow 0.6$
        $c_1 \leftarrow 0.9$
        $c_2 \leftarrow 0.9$
    else if mode is *linear* then
        $q \leftarrow 0.5$
        $Q \leftarrow 2.5$
        $w \leftarrow 0.6$
        $c_1 \leftarrow Q - \frac{t}{T}(Q - q)$
        $c_2 \leftarrow q + \frac{t}{T}(Q - q)$
    else if mode is *cosine_equal_cross* then
        $q \leftarrow 0.5$
        $Q \leftarrow 2.5$
        $w \leftarrow 0.6$
        $c_1 \leftarrow q + \frac{Q-q}{2} cos(\pi(1 - \frac{t}{T})) + 1$
        $c_2 \leftarrow q + \frac{Q-q}{2} cos(\pi \frac{t}{T}) + 1$
    else if mode is *cosine_late_cross* then
        $q \leftarrow 1.5$
        $Q \leftarrow 2.5$
        $w \leftarrow 0.6$
        $c_1 \leftarrow q + \frac{Q-q}{2} cos(\pi(1 - \frac{t}{T})) + 1$
        $m \leftarrow 0.5$
        $n \leftarrow 2.5$
        $c_2 \leftarrow q + \frac{Q-q}{2} cos(\pi \frac{t}{T}) + 1$
    else if mode is *cosine_no_cross* then
        $q \leftarrow 2.0$
        $Q \leftarrow 2.5$
        $w \leftarrow 0.6$
        $c_1 \leftarrow q + \frac{Q-q}{2} cos(\pi(1 - \frac{t}{T})) + 1$
        $m \leftarrow 0.5$
        $n \leftarrow 2.5$
        $c_2 \leftarrow q + \frac{Q-q}{2} cos(\pi \frac{t}{T}) + 1$
    else if mode is *AGPSO1* then
        $w \leftarrow 0.6$
        $c_1 \leftarrow t\frac{1}{T} + 1.25$
        $c_2 \leftarrow t\frac{-2.05}{T} + 2.55$
    return $w, c_1, c_2$
```

Fig. 11. The function for the generation of adaptive PSO search weights.

evaluation involves creating a full CNN architecture from the particle representation, training the CNN using stochastic gradient descent with backpropagation, and evaluating the CNN on the validation set to determine its fitness score. Treating the CNN training as a 'black box' like this means that our method could be generalised to other deep learning tasks, where the fitness function could be modified to train a different type of network without having to modify the PSO system. We train each individual CNN model using stochastic gradient descent (SGD) with a nesterov momentum factor [37], [38] of 0.9, and weight decay (L2 regularisation) of 1e-4, on minibatches of size 256 for one epoch per fitness function evaluation. The proposed PSO swarm optimisation is performed using a population size of 50 individual particles with a maximum number of iterations of 100.

1) LEARNING RATE

In this research, we also experimented with a number of different approaches for the learning rate schedule in order to find the best mechanism for our combined optimisation and training process. We started with (1) simple scheduled learning rate decay at certain points in the optimisation/training process using (15),

$$\alpha_\phi = \begin{cases} \gamma\alpha_{\phi-1}, & \text{if } \phi = 50 \textbf{ or } \phi = 75 \\ \alpha_{\phi-1}, & \text{otherwise} \end{cases} \tag{15}$$

where γ was set to 10^{-1} and the initial learning rate was also set to 10^{-1}. This approach was reasonably effective.

Motivated by [39] and [40], we experimented with (2) cyclic learning rates, which proved effective owing to the fast initial convergence, allowing the shared parameters to quickly become effective. Huang *et al.* [39] use a cyclical learning rate to repeatedly train a network to an acceptable level of performance, then store a snapshot of the model in that state. These snapshots are then combined into an ensemble model in a manner similar to gradient boosting, to achieve enhanced performance. They construct an ensemble by 'snapshotting' the parameter weights after each cycle of the learning rate, although we found it effective to simply train our models with shared parameters using the cyclic learning rate without snapshotting. The cyclic learning rate that we used in our experiments can be seen in (16).

$$\alpha_\phi = \frac{\alpha_0}{2}(cos(\frac{\pi mod(\phi - 1, \lceil \Phi/M \rceil)}{\lceil \Phi/M \rceil}) + 1) \tag{16}$$

where M represents the number of cycles, which was set to 5 in our experiments.

Our final approach for the learning rate was to simply use (3) cosine annealing to decrease the learning rate over a defined interval, throughout the optimisation process. This was performed using (17),

$$\alpha_\phi = b + \frac{a - b}{2} cos(\pi \frac{\phi}{\Phi}) + 1 \tag{17}$$

where a and b represent the initial and final learning rate values respectively, for the interval. We found this approach works best for our combined optimisation and training process, with values of 10^{-1} and 10^{-4} for a and b respectively. Therefore, all of the following reported results use this strategy. We also used this decay strategy for the fine-tuning portion of the model training, using values of 10^{-4} and 10^{-7} for a and b respectively. A visualisation of the three different learning rate schedules can be seen in Fig. 12.

2) FINE-TUNING

As previously mentioned, the usual method for evaluating architecture optimisation systems is to separate out the optimisation process from the final model training process. Contrary to this approach, we evaluate our optimisation process by integrating it into the training process for the final model to be tested. Once the optimisation process has completed, we then fine-tune the best performing model on a combined

Table 4. Aggregate performance measures on the CIFAR-10 test set.

Stage	Accuracy	Error Rate	Macro-Average Precision	Macro-Average Recall	Macro-Average F1 Score
Pre-fine-tuning	10.56%	89.44%	0.0544	0.1056	0.0718
Post-fine-tuning	95.22%	4.78%	0.9523	0.9522	0.9522

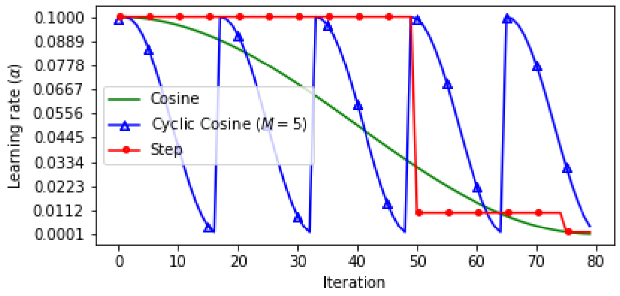


Fig. 12. Learning rate by iteration.

training set consisting of the training and validation sets together for a small number of epochs. In this way the training of the final network is embedded in the optimisation process, meaning the architecture design and training are performed as one task. By using this approach we couple the architecture design and training processes together and remove some of the high barrier-to-entry for building CNNs for new problems. The finetuning process is performed for 10 epochs and uses SGD with a nesterov momentum factor of 0.9, and weight decay (L2 regularisation) of 1e-4, on minibatches of size 256 with a cosine annealing learning rate schedule over an interval of $a = 10^{-4}$ and $b = 10^{-7}$.

IV. EVALUATION

The well known CIFAR-10 and CIFAR-100 datasets [7] are used to evaluate the proposed SOBA model.

The CIFAR-10 dataset consists of 60,000 images equally split over 10 classes (6,000 per class). The dataset divides into 50,000 training images and 10,000 test images. For our experiments we further divided the 50,000 training images into 45,000 training and 5,000 validation, whereby the validation set was used to generate the fitness scores for each function evaluation in the optimisation process. Fig. 13(a) shows a confusion plot generated from the **pre-fine-tuning** test on the CIFAR-10 dataset, whilst (18) shows the raw confusion matrix.

$$\begin{bmatrix} 933 & 0 & 119 & 24 & 4 & 819 & 6 & 20 & 12 & 2 \\ 18 & 15 & 2 & 4 & 0 & 1 & 0 & 0 & 965 & 947 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 4 \\ 1 & 963 & 30 & 94 & 22 & 122 & 990 & 4 & 8 & 17 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 12 & 21 & 834 & 15 & 14 & 6 & 1 & 20 & 9 & 12 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 32 & 0 & 5 & 846 & 7 & 39 & 1 & 19 & 6 & 16 \\ 4 & 0 & 10 & 17 & 953 & 13 & 2 & 937 & 0 & 2 \end{bmatrix} \tag{18}$$

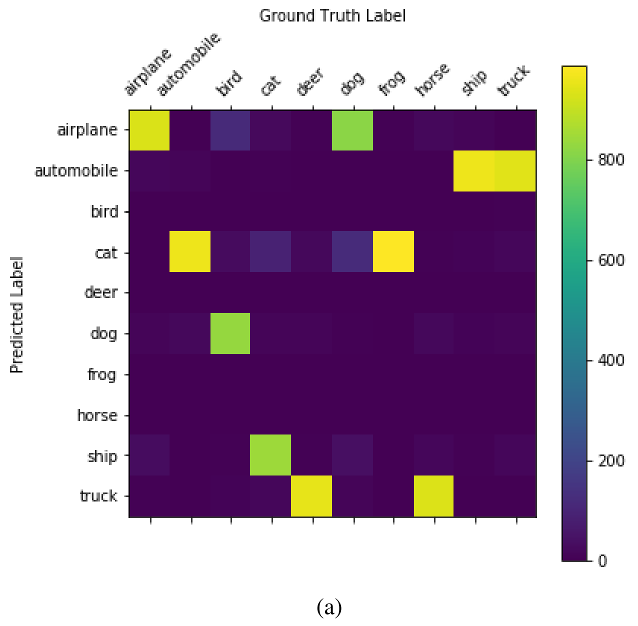
This test was not used to influence the fine-tuning in any way. It was simply performed in order to demonstrate the effect of fine-tuning on the classification performance. Fig. 13(b) shows the confusion plot generated **after fine-tuning**, with (19) showing the accompanying raw confusion matrix.

$$\begin{bmatrix} 965 & 1 & 7 & 6 & 3 & 2 & 3 & 4 & 16 & 4 \\ 1 & 979 & 0 & 1 & 0 & 0 & 0 & 0 & 5 & 22 \\ 13 & 0 & 934 & 18 & 8 & 9 & 8 & 4 & 5 & 1 \\ 2 & 0 & 14 & 878 & 9 & 41 & 8 & 8 & 2 & 1 \\ 1 & 1 & 15 & 16 & 966 & 12 & 3 & 12 & 0 & 0 \\ 0 & 0 & 13 & 64 & 8 & 930 & 1 & 9 & 0 & 1 \\ 1 & 0 & 10 & 7 & 2 & 3 & 975 & 1 & 0 & 0 \\ 0 & 0 & 3 & 5 & 3 & 2 & 2 & 961 & 0 & 0 \\ 16 & 3 & 3 & 4 & 1 & 1 & 0 & 0 & 968 & 5 \\ 1 & 16 & 1 & 1 & 0 & 0 & 0 & 1 & 4 & 966 \end{bmatrix} \tag{19}$$

Table 4 shows aggregate statistics for the CIFAR-10 test runs before the fine-tuning process and after. It is clear that the fine-tuning process is effective in reversing the overfitting on the training set and allows the model to generalise to greatly improved performance on the test set. This fast, effective generalisation is owing to the robustness of the weights learned through our combined optimisation and training process, as each block in the architecture is trained to be globally optimal for all potential surrounding configurations, rather than one single, fragile configuration. It is also interesting to note that after fine-tuning, the areas that the model seems to struggle, namely the intersections of cat-dog, ship-airplane, and truck-automobile, are areas that intuitively would also prove difficult to a human classifying the images by hand.

The CIFAR-100 dataset consists of the same number of images but split over 100 classes, each of which belongs to one of twenty 'superclasses'. Each class has 500 training images and 100 testing images, resulting in the same training-testing split as that of CIFAR-10 (50,000 vs 10,000 respectively). As with CIFAR-10, we further split the training images into 45,000 training and 5,000 validation for CIFAR-100, and the validation images are used to generate the model fitness after each optimisation function evaluation.

Fig. 14 shows some analysis of the lookup table following the final access during the optimisation/training process. Each line represents a different block in the architecture, with the number of layers in the block displayed along the x axis. The y axis shows the lowest error rate achieved by any particle with that configuration of block in its architecture (although the other blocks could be in any configuration). It is clear from the CIFAR-10 experiment (Fig. 14(a)) that the system tends to prefer more depth in the initial blocks, whilst the



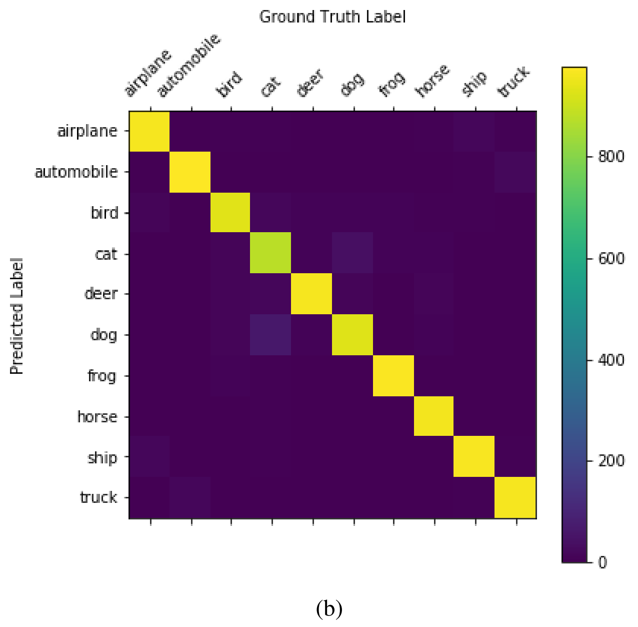
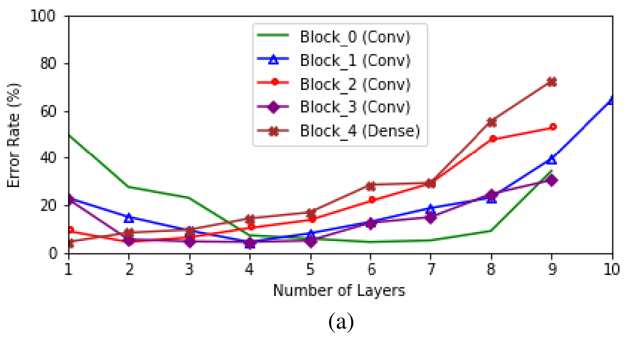


Fig. 13. Confusion plots for CIFAR-10 test results. (a) Pre-Finetuning. (b) Post-Finetuning.



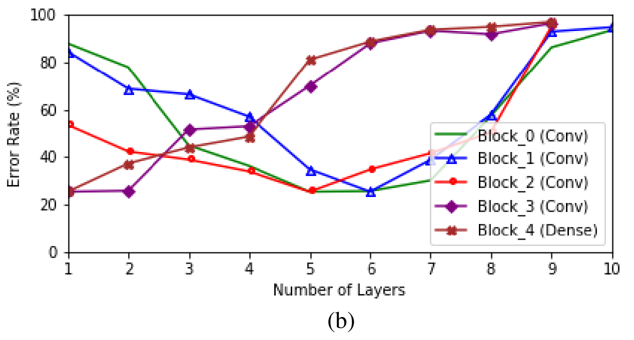


Fig. 14. The best validation error rates achieved for each individual block during the optimisation process, from the final lookup table. (a) CIFAR-10. (b) CIFAR-100.

feature maps are larger and the receptive field is smaller, and more shallow blocks later in the network, especially when it comes to the fully connected layers. This becomes drastically more pronounced in the CIFAR-100 experiment (Fig. 14(b)) when the number of output classes is increased by an order of magnitude and the same pattern can still be seen.

We have also compared our model with related research as illustrated in Table 5. The first section of the table contains traditional, hand-crafted architectures, the second section contains reinforcement learning (RL) techniques, the third section contains evolutionary techniques, and the final section contains our proposed model. In comparison with most of the existing studies, our proposed model achieves the best trade-off between performance and computational efficiency. As an example, [25], [26], and [19] used 200, 250, and 800 GPUs with 36, 264, and 672 processing hours respectively, but in some cases the performance enhancement was marginal. Whilst these works did allow for more variation in the proposed architectures, they are also inaccessible to the average user due to the enormous cost. They also require retraining from scratch for the eventually discovered architectures, whilst our method provides optimisation and training in one process, quickly delivering an effective, trained network. In addition, we also compare against hand-designed methods [6], [9], [41], and [42], where our method creates architectures that are most similar to [6]. However, we achieve much improved results using our combined optimisation and training strategy and the enhanced PSO model to generate and train the models. This improvement comes not only from the optimised network topology itself, but also from the robustness of the parameters in our network, as each block is trained to its optimal performance regardless of the configuration of the blocks surrounding it, ensuring that we do not train to fragile local minima. Networks such as VGG-16 [6] are likely to train to local minima as the structure of the network never changes during the training process. The integration of an overall optimisation process in SOBA ensures that even as the structure of the network changes, we still strive for the global minimum, thereby avoiding the local minima as they disappear and reappear. This robustness of weights is ideal for transfer learning as the network is already able to

Table 5. Classification error rates (%) for the proposed SOBA model and other related studies.

Method	CIFAR-10	CIFAR-100	GPUs	Time (hours)
Handcrafted architectures				
VGG [6]	7.25	-	n/a	n/a
ResNet-110 [9]	6.61	-	n/a	n/a
WideResNet [41]	3.80	18.30	n/a	n/a
DenseNet [42]	3.46	17.18	n/a	n/a
Reinforcement learning techniques				
MetaQNN [18]	6.92	27.14	10	10
Neural Architecture Search [19]	3.65	-	800	672
Evolutionary optimisation techniques				
Large-Scale Evolution [25]	5.40	23.00	250	264
Block-QNN [21]	3.54	18.06	32	72
SMASHv1 [27]	5.53	22.07	-	-
SMASHv2 [27]	4.03	20.60	-	-
Genetic CNN [43]	7.10	29.03	~ 20	~ 24
Hierarchical Representations [26]	3.60	-	200	36
Our system				
SOBA (inc training model)	4.78	25.42	**1**	**34** (**36** for C-100)

deal very effectively with changes in surrounding structure, so adding or removing layers will not disrupt the performance greatly.

V. CONCLUSION

In this research we have presented an enhanced PSO model with adaptive search weights and weight sharing for optimising the architecture of deep image classification networks. Our model starts with a hand-crafted skeleton architecture and quickly explores our constructed search space by concurrently training multiple architectures, using a weighted lookup table of trained parameters. This allows us to optimise the architecture of CNNs quickly and effectively, without having to fully train and validate each architecture from scratch, which is prohibitively expensive. We demonstrate results on CIFAR-10 and CIFAR-100, with comparison to other recent architecture optimisation studies. As illustrated in Table 5, our proposed model is among the top performers and has significantly lower computational and hardware requirements than those of other methods, even with our method of training and optimising all-in-one, with no separation between the optimisation and training of the final test architecture.

For future work, we intend to explore adding residual connections [9] to the architecture search space, in order to investigate the differences on the performance of the search methods. We intend to expand our formulation of the search space to treat the network structure as a Directed Acyclic Graph, rather than a simple hierarchical structure. We also intend to compare the performance of our enhanced PSO method with other, similar optimisation techniques inside our SOBA model on a wider variety of datasets, including transfer learning tasks such as facial emotion recognition, image description generation, and visual question generation.

ACKNOWLEDGMENT

The authors would like to thank RPPtv Ltd and NorthumbriaUniversity for jointly funding an industrial collaborative PhD studentship which encompassed the development and production of this work.

REFERENCES

[1] Y. LeCun *et al.*, "Backpropagation applied to handwritten zip code recognition," *Neural Comput.*, vol. 1, no. 4, pp. 541–551, 1989.

[2] O. Russakovsky *et al.*, "ImageNet large scale visual recognition challenge," *Int. J. Comput. Vis.*, vol. 115, no. 3, pp. 211–252, Dec. 2015.

[3] G. A. Miller, "WordNet: A lexical database for English," *Commun. ACM*, vol. 38, no. 11, pp. 39–41, 1995.

[4] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *Int. J. Comput. Vis.*, vol. 60, no. 2, pp. 91–110, 2004.

[5] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, Jun. 2005, vol. 1, pp. 886–893.

[6] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, pp. 1–14, Sep. 2014.

[7] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Univ. Toronto, Toronto, ON, Canada, Tech. Rep., 2009, p. 7, vol. 1, no. 4.

[8] C. Szegedy *et al.*, "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2015, pp. 1–9.

[9] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2016, pp. 770–778.

[10] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evol. Comput.*, vol. 10, no. 2, pp. 99–127, 2002.

[11] K. O. Stanley and R. Miikkulainen, "Efficient evolution of neural network topologies," in *Proc. Congr. Evol. Comput. (CEC)*, vol. 2, May 2002, pp. 1757–1762.

[12] K. O. Stanley and R. Miikkulainen, "Efficient reinforcement learning through evolving neural network topologies," in *Proc. 4th Annu. Conf. Genetic Evol. Comput.* Burlington, MA, USA: Morgan Kaufmann Publishers, 2002, pp. 569–577.

[13] K. O. Stanley, B. D. Bryant, and R. Miikkulainen, "Evolving adaptive neural networks with and without adaptive synapses," in *Proc. Congr. Evol. Comput. (CEC)*, vol. 4, Dec. 2003, pp. 2557–2564.

[14] D. Floreano, P. Dürr, and C. Mattiussi, "Neuroevolution: From architectures to learning," *Evol. Intell.*, vol. 1, no. 1, pp. 47–62, 2008.

[15] K. O. Stanley, D. B. D'Ambrosio, and J. Gauci, "A hypercube-based encoding for evolving large-scale neural networks," *Artif. Life*, vol. 15, no. 2, pp. 185–212, Apr. 2009.

[16] P. Verbancsics and J. Harguess, "Image classification using generative neuro evolution for deep learning," in *Proc. IEEE Winter Conf. Appl. Comput. Vis. (WACV)*, Jun. 2015, pp. 488–493.

[17] R. Miikkulainen *et al.* (2017). "Evolving deep neural networks." [Online]. Available: https://arxiv.org/abs/1703.00548

[18] B. Baker, O. Gupta, N. Naik, and R. Raskar. (2016). "Designing neural network architectures using reinforcement learning." [Online]. Available: https://arxiv.org/abs/1611.02167

[19] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," in *Proc. ICLR*, 2017, pp. 1–16.

[20] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Salt Lake City, UT, USA, Jun. 2018, pp. 8967–8710.

[21] Z. Zhong, J. Yan, W. Wu, J. Shao, and C.-L. Liu. (2017). "Practical block-wise neural network architecture generation." [Online]. Available: https://arxiv.org/abs/1708.05552

[22] H. Cai, T. Chen, W. Zhang, Y. Yu, and J. Wang. (2017). "Efficient architecture search by network transformation." [Online]. Available: https://arxiv.org/abs/1707.04873

[23] F. Assunção, N. Lourenço, P. Machado, and B. Ribeiro, "Evolving the topology of large scale deep neural networks," in *Proc. Eur. Conf. Genetic Program. (EuroGP)*, Parma, Italy, Springer, Apr. 2018, pp. 19–34.

[24] T. Elsken, J.-H. Metzen, and F. Hutter. (2017). "Simple and efficient architecture search for convolutional neural networks." [Online]. Available: https://arxiv.org/abs/1711.04528

[25] E. Real *et al.*, "Large-scale evolution of image classifiers," in *Proc. 34th Int. Conf. Mach. Learn.*, in Proceedings of Machine Learning Research, vol. 70. D. Precup and Y. W. Teh, Eds. Sydney, NSW, Australia: PMLR, Aug. 2017, pp. 2902–2911. [Online]. Available: http://proceedings.mlr.press/v70/real17a.html

[26] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu. (2017). "Hierarchical representations for efficient architecture search." [Online]. Available: https://arxiv.org/abs/1711.00436

[27] A. Brock, T. Lim, J. M. Ritchie, and N. Weston. (2017). "SMASH: One-shot model architecture search through hypernetworks." [Online]. Available: https://arxiv.org/abs/1708.05344

[28] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Mach. Learn.*, vol. 8, nos. 3–4, pp. 229–256, 1992.

[29] R. Negrinho and G. Gordon. (2017). "Deeparchitect: Automatically designing and training deep architectures." [Online]. Available: https://arxiv.org/abs/1704.08792

[30] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le. (2018). "Regularized evolution for image classifier architecture search." [Online]. Available: https://arxiv.org/abs/1802.01548

[31] B. Wang, Y. Sun, B. Xue, and M. Zhang. (2018). "Evolving deep convolutional neural networks by variable-length particle swarm optimization for image classification." [Online]. Available: https://arxiv.org/abs/1803.06492

[32] R. Eberhart and J. Kennedy, "A new optimizer using particle swarm theory," in *Proc. 6th Int. Symp. Micro Mach. Hum. Sci. (MHS)*, Oct. 1995, pp. 39–43.

[33] D. H. Wolper and W. G. Macready, "No free lunch theorems for optimization," *IEEE Trans. Evol. Comput.*, vol. 1, no. 1, pp. 67–82, Apr. 1997.

[34] K. Mistry, L. Zhang, S. C. Neoh, C. P. Lim, and B. Fielding, "A micro-GA embedded PSO feature selection approach to intelligent facial emotion recognition," *IEEE Trans. Cybern.*, vol. 47, no. 6, pp. 1496–1509, Jun. 2017.

[35] S. Mirjalili, A. Lewis, and A. S. Sadiq, "Autonomous particles groups for particle swarm optimization," *Arabian J. Sci. Eng.*, vol. 39, no. 6, pp. 4683–4697, 2014.

[36] A. Paszke *et al.*, "Automatic differentiation in pytorch," in *Proc. 31st Conf. Neural Inf. Process. Syst.*, Long Beach, CA, USA, 2017.

[37] Y. Bengio, N. Boulanger-Lewandowski, and R. Pascanu, "Advances in optimizing recurrent networks," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, May 2013, pp. 8624–8628.

[38] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," in *Proc. Int. Conf. Mach. Learn.*, 2013, pp. 1139–1147.

[39] G. Huang, Y. Li, G. Pleiss, Z. Liu, J. E. Hopcroft, and K. Q. Weinberger. (2017). "Snapshot ensembles: Train 1, get M for free." [Online]. Available: https://arxiv.org/abs/1704.00109

[40] L. N. Smith and N. Topin. (2017). "Super-convergence: Very fast training of neural networks using large learning rates." [Online]. Available: https://arxiv.org/abs/1708.07120

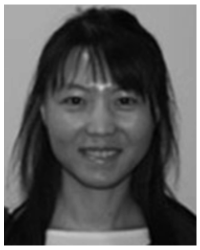
[41] S. Zagoruyko and N. Komodakis. (2016). "Wide residual networks." [Online]. Available: https://arxiv.org/abs/1605.07146

[42] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proc. CVPR*, vol. 1, no. 2, 2017, p. 3.

[43] L. Xie and A. Yuille, "Genetic CNN," in *Proc. ICCV*, Oct. 2017, pp. 1388–1397.

BEN FIELDING received the B.Sc. degree in computer science from Northumbria University, Newcastle upon Tyne, U.K., in 2015, where he is currently pursuing the Ph.D. degree.

His current research interests include deep learning, computer vision, and evolutionary computation.



LI ZHANG (M'14) received the Ph.D. degree from the University of Birmingham. She is currently an Associate Professor and a Reader in computer science with Northumbria University, U.K., and also serving as an Honorary Research Fellow at the University of Birmingham, U.K.

She holds expertise in artificial intelligence, machine learning, evolutionary computation, and deep learning. She has served as an Associate Editor for *Decision Support Systems*.