Evolving Efficient Deep Neural Networks for Real-time Object Recognition

Gongjin Lan
Department of Computer Science,
Vrije Universiteit Amsterdam
The Netherlands
g.lan@vu.nl

Lucas de Vries
Faculty of Science, University of Amsterdam
The Netherlands
lucas.devries2@student.uva.nl

Shuai Wang
Department of Computer Science,
Vrije Universiteit Amsterdam
The Netherlands
s2.wang@vu.nl

ABSTRACT

While Deep Neural Networks (DNNs) achieve state-of-the-art accuracy in object recognition, they rely on the deep networks with millions or even billions of parameters. Current DNNs often take expensive computation. Accelerating DNNs by reducing the parameters of DNNs is crucial for real-time object recognition on low-performance computing hardware. This paper explores an evolutionary approach to evolve efficient DNNs with desired accuracy for real-time object recognition. This approach achieves the goal by two design choices. First, NeuroEvolution of Augmenting Topologies (NEAT) is applied to evolve both weights and topology of DNNs. NEAT evolves neural networks from simple initial topologies, which reduces the number of parameters of DNNs from millions to thousands. Second, we propose fitness functions to further select the evolved DNNs with fewer parameters and high accuracy. The experimental results show that the best evolved DNN recognizes the objects (modular robots in an arena) on a microcomputer, Raspberry Pi 3, with an accuracy of 95.6% and a speed of 5 fps. Last, we test the approach on well-known MNIST dataset for recognizing multi-class objects. It also achieved the reasonable accuracy and small size DNNs. This work can be extended to other real-time tasks. We published the source code[1] and video[2] of results that our approach recognizes two modular robots simultaneously in the real world.

CCS CONCEPTS

• **Computer systems organization** → **Embedded systems**; *Redundancy*; Robotics; • **Networks** → Network reliability;

KEYWORDS

Object recognition, Real-time, Neural networks, Neuroevolution, NEAT, Evolving neural networks, Robot vision, Low-performance computing hardware

ACM Reference Format:
Gongjin Lan, Lucas de Vries, and Shuai Wang. 2019. Evolving Efficient Deep Neural Networks for Real-time Object Recognition. In *Proceedings of the Genetic and Evolutionary Computation Conference 2019 (GECCO '19).* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

[1]https://github.com/langongjin/Evolving-DNNs
[2]https://youtu.be/RVjSXMunY1c

GECCO '19, July 13–17, 2019, Prague, Czech Republic

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

In recent years, DNNs have become the state-of-the-art models in many areas of machine learning, particularly object recognition such as face recognition [37, 38], pedestrian detection [27], vehicle detection [24], etc. However, most current methods of optimizing DNNs like back-propagation, are limited to optimize the weights of DNNs [34]. The topology of DNNs is often designed by researchers empirically. It is a tedious trial-and-error process and difficult to design the efficient DNNs as few parameters as possible for real-time object recognition. To achieve high accuracy, the current methods generally design the neural networks with deep layers topology. However, these high-performing neural networks rely on deep networks containing millions or even billions of parameters.

Although current DNNs have achieved remarkable accuracy, they are usually applied on powerful hardware systems such as GPU platforms and hardly implemented on Low-Performance Computing Hardware (LPCH) because of the expensive computation. Optimization algorithms like CMA-ES can be used to optimize the topology of DNNs for the appropriate networks [32], instead of empirical parameter tuning. Nevertheless, the structure of DNNs still grows deeper and deeper, and often contains significant redundant parameters that take expensive computation. Therefore, it is crucial to generate the appropriate DNNs with fewer parameters, while maintaining expected accuracy. In particular, this issue is extremely important for real-time object recognition on LPCH.

This paper therefore proposes a method that evolves DNNs by neuroevolution for real-time object recognition on LPCH like the microcomputer, Raspberry Pi 3. Although accelerating DNNs by reducing the parameter redundancy has attracted many research attention, most of them are just relatively accelerated that compare to the DNNs with millions of parameters. In this work, we use NEAT algorithm to evolve both weights and the topology of DNNs for real-time object recognition. It reduces the number of parameters of DNNs from millions to thousands. Furthermore, we propose fitness functions to evaluate the performance of evolved DNNs in terms of accuracy and computation time. It further selects evolved DNNs for lower computation time and desired accuracy. We apply the best evolved DNN to recognize modular robots on LPCH. The experimental results show that our method successfully evolves DNNs for real-time object recognition. The best evolved DNN was loaded and run on a Raspberry Pi 3 and recognizes modular robots at 5 fps and 95.7% accuracy. Furthermore, we test the approach

on the well-known MNIST dataset for recognizing multi-class objects. It achieves the small size DNNs and resonavle accuracy for multi-class object recognition on MNIST dataset. The experimental results demonstrate that our approach evolves efficient DNNs with thousands of parameters for both two-class and multi-class real-time object recognition.

2 RELATED WORK

State-of-the-art Object Recognition In recent years, DNNs became the state-of-the-art models in object recognition [16]. For example, the outstanding work [16] achieved breakthrough results on the ImageNet dataset using a DNN containing 0.6 million nodes and 61 million parameters (cost 240 MB storage) with five convolutional layers and three fully-connected layers. Other models include R-CNN [10], Faster R-CNN [29], YOLO [28], SSD [22]. Another example is that [31] proposed a very deep network that contains 1.5 million nodes and 144 million parameters (takes 528MB storage) for large-scale image recognition. The top face recognition results [37, 38] were obtained with DNNs containing hundreds of millions of parameters. Such huge DNNs can hardly reach real-time performance on LPCH. Furthermore, a harsh reality is that recent neural networks are deeper and deeper.

Real-time Object Recognition Although current DNNs achieve remarkable accuracy, they are difficult to work on LPCH due to expensive computation and significant memory requirements. Some recent works proposed improved approaches for efficient DNNs with fewer parameters. Generally, optimization algorithms like CMA-ES can be used to optimize the topology of DNNs for the appropriate networks [32], instead of empirical parameter tuning. [25] presented an optimized Fast R-CNN on a CPU and GPU platform at 1.85fps. A YOLO-based low-complexity neural network was proposed that works on a GPU platform for object recognition [39]. Although there are many related works that generate DNNs for real-time object recognition, these DNNs only can be run on GPU platforms rather than LPCH like Raspberry Pi. For instance, the accelerated DNN in our previous work [17] achieved a speed of 1.76 fps on Raspberry Pi 3. However, it still can not meet the requirement in many applications. Nevertheless, the structure of DNNs would still grow deeper and deeper, and often contains significant redundant parameters that take expensive computation. Therefore, it is crucial to generate appropriate DNNs with fewer parameters, while maintaining expected accuracy, especially for real-time object recognition on LPCH.

Accelerating Neural Networks Current DNNs rely on the deep networks with millions or even billions of parameters to achieve impressive accuracy, but often take expensive computation. Thus, accelerating DNNs by reducing the parameters of DNNs is crucial for real-time object recognition. [2] presented a circulant projection approach to replace the conventional linear projection in fully-connected layers for reducing the parameter redundancy of DNNs. [20] proposed a method ESPACE to accelerate the DNNs by eliminating the spatial and channel redundancy. A sparse decomposition was used to reduce the parameter redundancy of DNNs in [21], where the maximum sparsity is obtained by exploiting both inter-channel and intra-channel redundancy. [12] described a method that reduces parameters of well-known AlexNet from 61 million to 6.7 million by pruning the redundant connections. Similarly, [43] proposed a structured sparsity learning method to regularize the topology of AlexNet, which achieves a speed at average 5.1× speedup of computation. Furthermore, to compare with [43], [23] adopted a Bayesian point of view through sparsity inducing priors to prune the size of the network and furthermore reduce compute time. It reached a speed up factor of around 8× on a Titan X (GPU) platform. Although these methods accelerated the computation of DNNs, these DNNs are only limited accelerated compare to the conventional DNNs with millions of parameters on CPUs or/and GPUs platform. In contrast, our work focus on evolving DNNs for fewer parameters that makes it possible to work on LPCH for real-time object recognition.

Evolving Neural Networks Evolutionary algorithms have been a great interest in obtaining neural networks [7]. This approach is called NeuroEvolution. Over the last decades, NeuroEvolution has been successfully applied to many fields, e.g., general game playing [13, 30, 33], evolutionary robotics [3, 45]. Among them, NEAT [36] and HyperNEAT [35] are two successful algorithms that were proposed to evolve neural networks. They made evolving DNNs more practical for the evolution of both weights and topology in many fields. For example, they have been successfully applied to the tasks of evolutionary robotics [11, 18]. In the application of object recognition, [6] proposed an evolutionary approach for real-time object recognition by genetic programming, which works on a GPU platform rather than LPCH. [8] proposed a genetic approach to images classification that only tested on benchmark datasets rather than real-time object recognition in real-world. Furthermore, it aims to overcome the limitations of manually crafted architectures and low interpretability for convolutional neural networks. Similarly, [42] proposed variable-length particle swarm optimization to evolving deep convolutional neural networks for image classification on GPUs platform. A recent work [26] proposed a new algorithm, CoDeepNEAT, for optimizing deep architectures by extending existing methods of evolving neural networks to topology, components, and hyper-parameters. In this work, we explore evolving efficient deep neural networks for real-time object recognition on LPCH.

3 METHODOLOGY

NEAT and HyperNEAT are the most successful NeuroEvolutions that have been applied to evolve neural networks in many fields. HyperNEAT is an extension of NEAT method. It uses NEAT to evolve the connection weights and network topology of Compositional Pattern Producing Networks (CPPNs). And then, CPPNs generate connection weights of objective neural networks with fixed topology. NEAT algorithm evolves both weights and topology of neural networks directly. By contrast, HyperNEAT is more suitable for evolving very large neural networks in weight space, while NEAT is more suitable for evolving neural networks from simple initial topology in both weight and topological space. In this work, we therefore employ NEAT algorithm to evolve DNNs for real-time object recognition. First, NEAT generates new DNNs by evolving weights and topology for searching better DNNs. Second, the features of samples in datasets are extracted and fed into new DNNs. Subsequently, the evolved DNNs output the results of object recognition. The performance of each evolved DNN is evaluated

by a fitness function whose output value depends on both accuracy and computation time. The process iterates until termination, which is shown in Figure 1. In this section, we describe the details of main steps including NEAT algorithm, input features, and fitness functions.

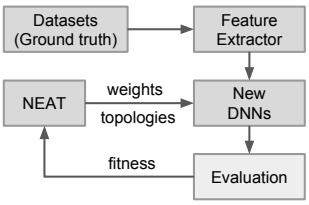

Figure 1: The work-flow of evolving DNNs includes three main steps: evolving DNNs by NEAT, feature extracting, evaluation by the fitness function. Note that, to observe the evolving of DNNs independently, this work designs the fixed feature extractor that extracts informative feature instead of evolving an feature extractor.

3.1 The NEAT algorithm

NeuroEvolution of Augmenting Topologies is a genetic algorithm that evolves DNNs in weight and topological space, attempting to search an optimal DNNs on both aspects of weights and topology [36]. We summarize the properties of NEAT to address how it evolves DNNs.

- NEAT evolves DNNs with a flexible topology, starting from an elementary topology where all input nodes are connected to all output nodes. In this way, the DNNs can be evolved that are as fewer parameters as possible. The correct matching of the genomes through marking genes with historical markings.
- The addition of nodes and connections in neural networks leads to an augmented topology. It allow to generate more flexible network structures. NEAT searches optimal DNNs through the weight space and the topological space simultaneously. There is no need for an initial or pre-defined fixed-topology that relies on the experience of researchers. Recombination and mutation induce an optimal topology of DNN to real-time object recognition. For instance, the crossover of genomes and the removal or addition of connections and nodes explore the different topology of DNNs.
- The solutions in the evolving population are grouped by similarity into species. Each of solutions can compete only with individuals in the same species.

In this work, we implement NEAT by employing an adaptable library, *MultiNEAT* [3], which is friendly to develop with Python bindings and has enhanced efficiency with its core components written in C++.

[3]https://github.com/peter-ch/MultiNEAT

3.2 Input Features

For many practical problems, the input feature to a neural network is a crucial factor for the final performance. Current feature extractors generally generate informative features by extracting high dimensional features. For instance, convolutional feature extractors in convolutional neural networks often include many convolutional layers and many convolutional kernels each layer. They extract informative but high dimensional features. However, High dimensional input often increases the size of DNNs and thus take more expensive computation for running DNNs. We therefore expect that the input feature is low dimensional and informative. To observe the evolving of DNNs independently, this work uses the fixed feature extractor instead of evolving an feature extractor. To find an appropriate feature extractor, we compare the performance of the two types of well-known feature extractors that output informative features with moderate dimensions, include *Sobel* feature extractor [41] and Histogram of Oriented Gradients (HOG) [4] feature extractor for evolving DNNs.

The Sobel feature extractor performs a 2D spatial gradient measurement on images. It converts a pixel array P into the corresponding gradient magnitude that represents meaningful feature with less redundant [41]. The Sobel operator consists of a pair of 3×3 convolution kernels G_x and G_y, as shown below.

$$P = \begin{bmatrix} P_{11} & P_{12} & P_{13} \\ P_{21} & P_{22} & P_{23} \\ P_{31} & P_{32} & P_{33} \end{bmatrix}, G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

where G_x and G_y are horizontal and vertical kernels respectively that calculate the gradient horizontally and vertically with a stride of 3. Such gradients can then be combined to the absolute or approximate gradient magnitude [41]. In this work, we calculate the absolute gradient magnitude on three colour channels as follows.

$$\begin{bmatrix} G_r & G_g & G_b \end{bmatrix}^T = \sqrt{(P \cdot G_x)^2 + (P \cdot G_y)^2} \tag{1}$$

where G_r, G_g, G_b are the absolute gradient magnitude of the red, green, blue colour channels respectively. The edge feature on three channels can be extracted when the process is applied to an image.

The Histogram of Oriented Gradients (HOG) feature extractor was proposed in [4] which calculates the gradients of ROIs. In this paper, we test our approach on HOG descriptor with two parameter settings that have been demonstrated for moderate dimensions and high informativeness, noted HOGa and HOGb [17]. The detailed parameter settings are presented in Table 1.

	ROIsize	blockSize	blockStride	cellSize	bins
HOGa	(112,32)	(8,8)	(8,8)	(4,4)	9
HOGb	(112,32)	(16,16)	(8,8)	(8,8)	9

Table 1: The parameter settings of HOGa and HOGb feature extractors.

We test the three feature extractors above on the dataset in Table 3 to evolve DNNs. The Sobel convolution operator outputs a 1110 dimensional feature vector on 112×32 samples with a stride of 3. The feature extractors HOGa and HOGb output 2016 and 1404

dimensions vectors respectively. The three feature extractors output the features with different dimensions and informativeness as the input of evolved DNNs. Therefore, the evolution of DNNs has different initial neural networks and search space. The accuracy during the evolution process is shown in Figure 2. The best evolved DNN achieves 95.7% accuracy with Sobel feature extractor. Interestingly, evolving DNNs with Sobel feature in 1110 dimensions performs better evolution process than both HOGa in 2016 dimensions and HOGb in 1404 dimensions. In the traditional approaches, high dimensional features are more likely to contain informative features that contributes to the accuracy than the low dimensional features. However, the higher dimension features implies that the corresponding evolved DNNs have more nodes in input layer and a larger search space of the evolution. NEAT often achieves the better performance in a small search space. Therefore, this work choose the Sobel feature extractor due to the lower dimension of input feature and better performance.

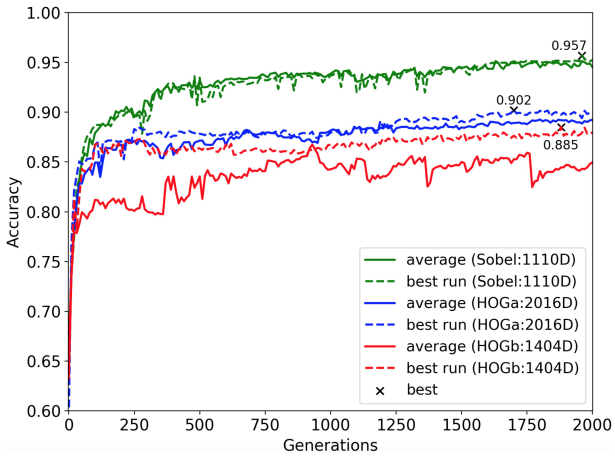

Figure 2: The performance of Sobel (green), HOGa (blue) and HOGb (red) feature extractor for evolving DNNs on the modular robots dataset. The solid lines are the average of accuracy. The dashed lines are the accuracy of best runs.

3.3 Fitness Function

Real-time object recognition on LPCH is usually not as complex as generic object recognition on super computing hardware (e.g., GPUs platform). It often aims to recognize two-class objects (object and non-object). But still, it is possible to be required to achieve multi-class object recognition. To achieve efficient fitness functions that can guide the evolution of DNNs for two-class and multi-class objects, we propose two fitness functions for two-class and multi-class objects respectively. This section provides the step-by-step derivations that lead to the fitness functions.

Two-class fitness function. To be simple in description, we first derive the fitness function by considering only accuracy without computation time. For the two-class object recognition, the DNNs output a two dimensional vector $\vec{p}_i\{(p_{i0}, p_{i1}) \mid p_{i0} + p_{i1} = 1\}$, where i is the index of samples in the dataset, and p_{i0} represents the probability that the sample is recognized as the object. The label of a sample is a two dimensional binary vector $\vec{l}_i \in \{(1,0),(0,1)\}$, where $(1,0)$ represents *target* and $(0,1)$ represents *non-target*. Considering a situation that a sample is recognized as object by two evolved DNNs with probabilities of 60% and 90% respectively, we expect that the evolved DNN (with result of 90%) obtains a higher fitness than another evolved DNN (with result of 60%) since this guides the evolution of DNNs towards the optimal DNNs better. Therefore we propose the fitness function[4] that guides the evolution to maximize the accuracy ($\vec{p}_i \cdot \vec{l}_i$) and the absolute value of the difference $\mid p_{i0} - p_{i1} \mid$. The evaluation on a sample can be expressed as follows.

$$f_i = \begin{cases} \vec{p}_i \cdot \vec{l}_i + \mid p_{i0} - p_{i1} \mid & \text{if } \vec{p}_i \cdot \vec{l}_i > 0.5 \\ \vec{p}_i \cdot \vec{l}_i - \mid p_{i0} - p_{i1} \mid -1 & \text{if } \vec{p}_i \cdot \vec{l}_i \leq 0.5 \end{cases} \tag{2}$$

where $\vec{p}_i \cdot \vec{l}_i$ is the accuracy, $\mid p_{i0} - p_{i1} \mid$ aims to add a value that scale up the fitness. For instance, a sample $\vec{l}_i = (1,0)$ is recognized as $\vec{p}_i = (0.8, 0.2)$, f_i therefore equals 1.4 (0.8 + (0.8 - 0.2)) rather than 0.8 (the accuracy). The value of 0.5 is the threshold for the correct classification that a sample is recognized as an object when $\vec{p}_i \cdot \vec{l}_i > 0.5$ (namely, $p_{i0} > p_{i1}$).

An evolved DNNs can be evaluated in a training dataset of size m by the fitness f_a as follows.

$$f_a = \frac{\sum_{i=0}^{m} f_i}{2m} \tag{3}$$

where the coefficient 2 aims to normalize f_a because the maximum of f_i is 2 when the $\vec{p}_i$ is (1,0) and $\vec{l}_i$ is (1,0).

However, we cannot only consider the accuracy for real-time object recognition in this work. Obtaining efficient DNNs with fewer parameters for low computation time is one of the goals that we expected. Therefore, it is crucial that the fitness function can encourage the evolution of DNNs towards both high accuracy and low computation time. The final fitness is proportional to f_a and inversely proportional to computation time. Hence, the fitness function can be expressed as follows.

$$f_{(f_a, C_t)} = f_a + \frac{1}{\beta \cdot C_t} \tag{4}$$

where C_t (in milliseconds) is the average evaluation time on the training dataset. β is the weight of C_t, can be modified depending on the requirement at accuracy and computation time of specific tasks. It aims to achieve a trade-off between accuracy and computation time.

Multi-class fitness function. While real-time object recognition on LPCH generally faces to simple tasks like recognizing two-class objects (object and non-object), multi-class object recognition is still likely to be required. We therefore propose a fitness function to evaluate the performance of the evolved DNNs for real-time multi-class object recognition. We provide a step-by-step derivation for the fitness function stated in Equation 6.

First, we directly use the accuracy instead of the probability $\vec{p}_i$ in Equation 2. Second, we introduce standard deviation σ of the number of correct recognition for each class objects to evaluate the performance of the evolved DNNs. In our preliminary experiments, we notice that the evolved DNNs often recognize the objects of some classes with high accuracy but hardly recognize the objects of

[4]The well-known F1-score is not suitable in this work.

other classes correctly. For instance, assuming the 5 classes objects from MNIST and 100 samples in each class are taken as the training dataset, two evolved DNNs perform the same average number of correct recognition but different distributions as (80, 79, 81, 78, 12) and (70, 68, 71, 61, 60) respectively. We expect that the latter has higher fitness because the objects of last class are hardly classified correctly in the former. Furthermore, in the preliminary experiments we also notice that, giving more evolutionary iterations would hardly improve the accuracy of the former DNNs on the fifth class object (0.12 accuracy) but the latter DNN is easier to be evolved for higher accuracy. We therefore design the fitness function to reward or punish the evolved DNNs for low or high σ respectively as shown in Figure 3. We consider that the number of correct recognition for each class object has equally good distribution when σ is less than a constant σ'. In such cases, we give a constant reward η to the evolved DNNs. Then, the reward is gradually decreased over σ when $\sigma > \sigma'$. Furthermore, the reward equals zero and becomes a gradually increasing penalty (minus value) to an evolved DNNs when $\sigma > \zeta$.

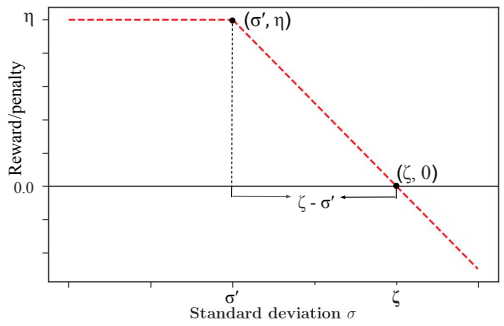

Figure 3: Illustration on the reward/penalty over the standard deviation σ of the number of correct recognition in each class for the multi-class object recognition.

Thus, the fitness function (denote f_{Ma}) is defined with the accuracy and the standard deviation σ as follows.

$$f_{Ma} = \begin{cases} \frac{\mathcal{N}_c}{m} + \eta & \text{if } \sigma < \sigma' \\ \frac{\mathcal{N}_c}{m} + \eta + \frac{-\eta(\sigma-\sigma')}{\zeta-\sigma'} & \text{if } \sigma \geq \sigma' \end{cases} \tag{5}$$

where $\frac{\mathcal{N}_c}{m}$ is the accuracy, $\mathcal{N}_c$ is the sum of the number of correct recognition for each class, and m is the size of the training dataset. The parameters η, σ', ζ can be adjusted depending on the size of the training datasets and tasks. In this work, we test the setting $\eta = 0.25$, $\sigma' = 20$, $\zeta = 40$ on MNIST dataset to recognize the handwritten digits for appropriate reward and penalty.

Similar to Equation 4, we provide the fitness function for multi-class object recognition to encourage both high accuracy and low computation time as follows.

$$f_{M(f_a, C_{tM})} = f_{Ma} + \frac{1}{\alpha \cdot C_{tM}} \tag{6}$$

where C_{tM} (in milliseconds) is the average evaluation time on the training dataset. α is a parameter for adjusting the weight of computation time C_{tM}, which can be modified depending on the requirement of tasks.

4 EXPERIMENTS

4.1 Experimental Set-up

The experiments include two parts: evolving DNNs on the desktop computer with a 2.6GHz Intel i5 CPU and deploying the evolved DNNs for real-time object recognition on Raspberry Pi 3 with a 1.2GHz ARM CPU in the real world. The evolution of DNNs is tested on two datasets, including a self-defined modular robots dataset, and MNIST dataset. In the preliminary experiments, we tested different parameter settings in NEAT algorithm for expected performance. The final main parameters of NEAT that we used in this work are listed in Table 2. The parameters *MutateAddNeuronProb* and *MutateAddLinkProb* aim to add a node and a connection respectively for searching the appropriate topology. *MutateAddLinkProb* attempts to add a new connection between two chosen nodes. If the connection already exists, it is replaced by a new connection. These operations improve the evolution of DNNs towards the suitable topology with fewer parameters. Adding nodes or links guides the evolution to more diversified DNNs. An evolved DNN generally needs many iterations to search for superior weights. *MutateWeightsProb* and *WeightMutationRate* are two mutation probability for the achievement that the weights are evolved sufficiently towards the satisfied DNNs.

Parameter	two-class	Multi-class
PopulationSize	100	100
MinSpecies	5	5
MaxSpecies	15	15
OverallMutationRate	0.75	0.50
MutateAddNeuronProb	0.10	0.05
MutateAddLinkProb	0.10	0.03
MutateRemLinkProb	0.01	0
MutateWeightsProb	0.70	0.90
WeightMutationRate	0.40	1.0
WeightMutationMaxPower	0.50	1.0
WeightReplacementRate	0.20	0.20
WeightReplacementMaxPower	4.00	1.0
Elitism	0.01	0.01

Table 2: The main parameters setting of NEAT.

The preliminary results in this work demonstrated that Sigmoid activation functions yield better DNNs than other activation functions. Thus, we use the unsigned Sigmoid functions as the activation functions for hidden and output node. The softmax function is applied to normalize the output of evolved DNNs as a categorical probability distribution. As a result, the output of the final recognition is a vector $\vec{p}_{out}(p_1, p_2, ..., p_i, ..., p_n \mid \sum_{i=1}^{n} p_i = 1)$. In addition, we tested many values of β in Equation 4 and α in Equation 6. $\beta = 10$ and $\alpha = 10$ are finally set for the evolution on robots dataset and MNIST dataset respectively.

4.2 Experiments on Robots Dataset

In many practical tasks, there are often some specific constraints, such as limited computational resources, a small memory capacity, and limited physical size. For instance, the modular robots proposed in [15], have only space inside to fit a small battery and a small computer with low computational power. The same is true for other types of robots like drones [40] and swarm robots [14]. These constraints make real-time object recognition on LPCH particularly challenging. In this work, the evolving DNNs is applied to achieve real-time recognizing the modular robots.

4.2.1 Evolving on Robots Dataset. In object recognition, the comprehensiveness of the dataset is directly related to the performance. We took a large number of images for the modular robots from different views and distances to create a comprehensive dataset. All of the images were captured when the modular robots were moving. We divided the samples into the training and testing dataset, as shown in Table 3. All of the samples have a resized resolution of 112 by 32 pixels.

	Robots	Noise	Total
Training	2250	2164	4414
Testing	750	721	1471
Total	3000	2885	5885

Table 3: The statistics of the modular robots dataset.

We run the experiments with the main NEAT parameters as specified in Table 2 on the modular robots dataset. Generally, NEAT evolves neural networks from simple initial topologies which reduce the number of parameters from millions to thousands. Furthermore, we use the two-class fitness function in Equation 4 to evaluate the performance of evolved DNNs. Which encourages the evolution of DNNs towards both fewer parameters for low computation time and high accuracy. To explore how the part of computation time in fitness function further encourages the evolution of DNNs towards lower computation time. We observe the f_a and $f(f_a, C_t)$ of top 10 evolved neural networks in a generation, are shown in Table 4. we note that the evolved DNN $\mathcal{N}_2$ obtains higher fitness $f(f_a, C_t)$ than $\mathcal{N}_1$, but they have the same value of f_a. Similarly, the evolved DNN $\mathcal{N}_7$ ranks seventh by f_a but ranks ninth by $f(f_a, C_t)$. Therefore, the evolved DNNs are further selected for lower computation time by $f(f_a, C_t)$. As such in each generation, the evolved DNNs are guided to low computation time. It can be adjusted to higher or lower by tuning the parameter β in Equation 4. Empirically, the parameter β should be moderate in case the evolution cannot converge to the satisfied accuracy and low computation time.

We run the evolution with the fitness function $f(f_a, C_t)$ on the modular robots dataset for 5 runs. Each run takes more than a day. The fitness of the evolution process is shown in Figure 4.

During the evolution process, the size of evolved DNNs generally increases to search the optimal structure in topological space. Although NEAT often removes the connections for fewer parameters by the parameter *MutateRemLinkProb*, it is essential to increase

DNNs	f_a	$f_{(f_a,C_t)}$	Ranking by f_a	Ranking by $f_{(f_a,C_t)}$
$\mathcal{N}_1$	0.513	0.605	**1**	**2**
$\mathcal{N}_2$	0.513	0.612	**2**	**1**
$\mathcal{N}_3$	0.491	0.591	3	3
$\mathcal{N}_4$	0.489	0.587	4	4
$\mathcal{N}_5$	0.481	0.579	5	5
$\mathcal{N}_6$	0.480	0.578	6	6
$\mathcal{N}_7$	0.473	0.571	**7**	**9**
$\mathcal{N}_8$	0.472	0.573	**8**	**7**
$\mathcal{N}_9$	0.470	0.571	**9**	**8**
$\mathcal{N}_{10}$	0.468	0.571	10	10

Table 4: The values of f_a, $f(f_a, C_t)$ of top 10 evolved DNNs in a generation, and their ranking. The *Ranking* by f_a and $f_{(f_a,C_t)}$ show the ranking lists respectively.

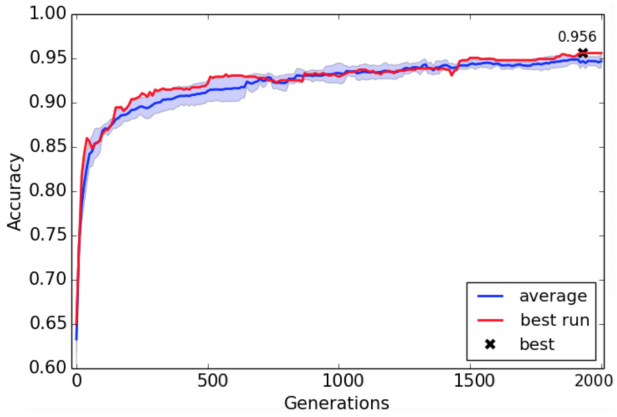

Figure 4: The average accuracy of evolving DNNs over 5 runs with Sobel feature extractor on the modular robots dataset. The lightblue shadow is the 95.45% confidence area (two standard deviations). The blue line is the average accuracy. Red line is the best run with the best accuracy of 95.6%

nodes and connections for searching appropriate structures in topological space by the parameters *MutateAddNeuronProb* and *MutateAddLinkProb*. The best topology of evolved DNNs with $f_(f_a, C_t)$ and f_a in generation 400, 1200, 1960/2000 are shown in the top and bottom of Figure 5 respectively. Furthermore, the corresponding specification of the topologies is shown in Table 5. It shows that the topology of DNNs is evolved to more and more complex networks over generations for optimal structure. Although the topology of DNNs is evolved towards deeper and deeper, the total number of parameters still remains in thousands. The evolved DNNs with $f_(f_a, C_t)$ have the smaller network than the evolved DNNs with f_a. The evolution with fitness function $f(f_a, C_t)$ obtains the evolved DNNs with subequal accuracy but fewer parameters than the evolution with fitness function f_a. Comparing to the traditional DNNs with millions even billions of parameters, the evolved DNNs significantly reduce the number of parameters for low computation time. Therefore, our fitness function successfully encourages the evolution of DNNs towards fewer parameters and high accuracy.

As shown in Table 5, evolution with f_a tends to achieve high accuracy by adding more connections searching topological space. By comparison, evolution with $f(f_a, C_t)$ achieves the DNNs with fewer parameters and layers.

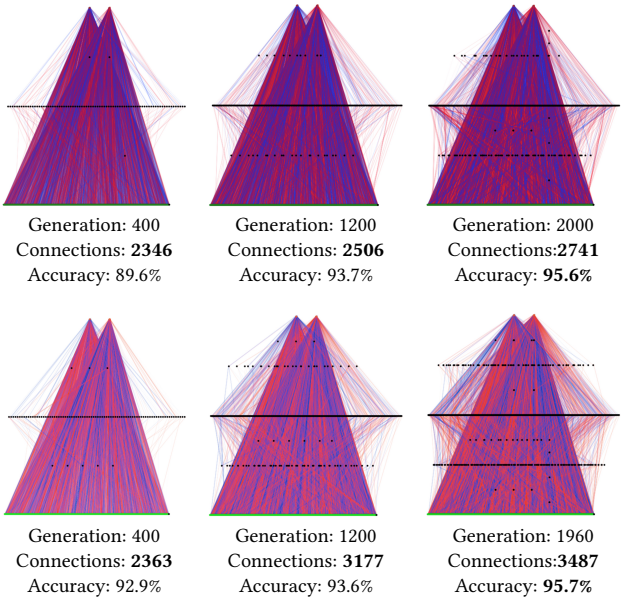

Figure 5: Six topologies of the best DNNs in 400, 1200 and 1960/2000 generations. The top three topologies are the best DNNs in 400, 1200 and 2000 generations of the evolution with Sobel feature extractor and the fitness function $f(f_a, C_t)$. The bottom three topologies are the best DNNs in 400, 1200 and 1960 generations of the evolution with Sobel feature extractor and the fitness function f_a. Red connections indicate positive weights, blue connections indicate negative weights. The input nodes are represented by green points. The black points are hidden nodes.

4.2.2 Real-time object recognition. In the real-world tasks, we aim to real-time recognize the modular robots. Therefore, we test the best evolved DNNs on Raspberry Pi 3 and Raspberry Pi camera V2. It mainly consists of two parts including searching ROIs and classifying by the best evolved DNNs. The work-flow on Raspberry Pi 3 is shown in Figure 6. We use the Fast ROIs Search (FROIS) algorithm to propose the ROIs on images from the camera. The FROIS algorithm has been demonstrated that it propose ROIs on the images with a low computation on Raspberry Pi 3 in our previous work [17]. The softmax function is applied to normalize the output of evolved DNNs into a probability distribution over the output classes. The results show that the ROIs on images are recognized as *Robot* or *Noise*.

We test the computation time of recognizing modular robots with the best evolved DNN and Sobel feature extractor as shown in Table 6. The results are averaged over 1000 images. The best evolved DNN with 2741 parameters takes an average of 6.0 ms to recognize the modular robots. Note that, it takes only 3.2% of the total computing time t_{total}. Most of the computation time was spent

Nodes / Best NN in / Layer	400 gen.	1200 gen.	2000/1960 gen.
Input layer	1110+1	1110+1	1110+1
Hidden layer 1	1 / 5	15 / 29	1 / 1
Hidden layer 2	80/82	140 / 5	30/3
Hidden layer 3	2 / 3	5 /348	1 / 1
Hidden layer 4	-	- / 17	3/60
Hidden layer 5	-	- / 3	1 / 1
Hidden layer 6	-	-	213/6
Hidden layer 7	-	-	11/452
Hidden layer 8	-	-	1 / 2
Hidden layer 9	-	-	1/32
Hidden layer 10	-	-	- / 2
Output layer	2	2	2
Total connections	2346/2363	2506/3177	**2741**/3487
Reduction in size	17 (0.7%)	671 (21.1%)	746(21.4%)

Table 5: Illustration of the topologies of six best DNNs in 400, 1200, 1960/2000 generation respectively. The left and right in *left/right* are the numbers of connections of best DNNs for the evolution with $f(f_a, C_t)$ and f_a respectively. The percentage is the ratio of reduction in size to the number of best DNNs that evolved by f_a.

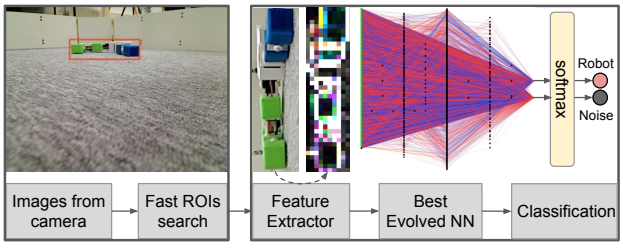

Figure 6: The work-flow of real-time recognizing modular robots on Raspberry Pi 3 in real-world. Notice that the ROIs (the region in the red bounding box) can be *Robot* or *Noise*. The features were extracted by Sobel convolutional operator which also can be HOGa or HOGb.

on reading images, searching the ROIs, and extracting feature on Raspberry Pi 3 with limited computing resources.

Compared with the methods that work on CPUs or/and GPUs platform we reviewed in Section 2, the best evolved DNN in this work takes significant few computation time. For instance, in our previous work [17], the accelerated 7 layers DNN with millions of parameters that designed by trial-and-error, takes 568 ms to recognize the modular robots at an accuracy of 96%. By contrast, the approach in this paper evolves a DNN for recognizing the modular robots on Raspberry Pi 3 at a speed of about 5 fps and an accuracy of about 95.6% in real-world. Thus, NEAT successfully evolved DNNs for real-time recognizing the modular robots with low computation time.

t_{caR}	t_{con}	t_{nn}	t_{total}
124.1ms	59.5ms	6.0ms	**189.6ms**
65.4%	31.4%	3.2%	100.0%

Table 6: The time component of recognizing modular robots on Raspberry Pi 3 with best evolved DNN and Sobel feature extractor. t_{caR} is the computation time that taken for reading an image and searching ROIs on the image. t_{con} is the feature extraction time by Sobel feature extractor. t_{nn} is the computation time.

4.3 Experiments on MNIST dataset

The experiments on robots dataset have successful evolved DNNs for real-time two-class object recognition. The approaches of real-time object recognition on LPCH generally face to the simple task like the recognition of object or non-object. Nevertheless, multi-class object recognition is still likely to be required. We therefore validate our approach on the dataset MNIST (a well-known dataset of handwritten digits) [5] for multi-class object recognition. MNIST dataset is widely used for training and testing in the field of machine learning [19]. MNIST dataset consists of samples of 28 by 28 greyscale pixels. The size of samples is small enough that we therefore directly take the value of pixels as the input of evolved DNNs. Therefore, MNIST is the suitable dataset to provide clear insights of evolving neural networks, which avoids the interference from the poor feature extractors. Even though there are many studies [1, 5, 9, 44] that evolve DNNs on MNIST dataset, they investigate on the goals like high accuracy rather than evolving for both the low computational cost and high accuracy DNNs with fewer parameters. We run the experiments of evolving DNNs on 5 classes digits ($0 \sim 4$) of MNIST rather than the full MNIST dataset with 10 classes digits. Because it generally not face to real-time recognizing 10 classes objects on LPCH. In this work, we aim to validate that our approach can evolve the efficient DNNs with fewer parameters and satisfied accuracy for multi-class object recognition on MNIST dataset, rather than challenging the state-of-the-art accuracy.

The experiments evolve DNNs on the training samples of MNIST dataset with 2 to 5 classes of digits. The accuracy of the best evolved DNNs for 2 to 5 classes of digits is shown in Table 7. Interestingly, the best accuracy of evolved DNNs decreases gradually when the number of classes increases. While the input dimensions of evolved DNNs are less than the evolution on the modular robots dataset, the higher dimension output increases the search space of solutions. NEAT algorithm often takes more iterations to search an satisfied solution in a larger search space. Therefore, the best evolved DNNs perform the lower accuracy for more classes of digits. The evolution of DNNs needs more iterations to achieve the expected accuracy and low computation time. We run the experiments with the parameter settings that are shown in Table 2. Empirically, recognizing digits on MNIST is not complex so that the topological space is not large. Therefore, we use a lower probability of adding neurons *MutateAddNeuronProb* and links *MutateAddLinkProb* on MNIST

[5] Although the handwritten digits in MNIST dataset are not general objects in real time object recognition, they are suitable to observe the evolving neural networks independently because they do not need feature extractor.

Number of classes	2	3	4	5
Accuracy	99.7%	96.0%	91.1%	89.1%

Table 7: The accuracy of best evolved DNNs for multi-class objects (2 to 5 classes) on MNIST dataset.

dataset than the modular robots dataset after preliminary testing. The topologies of best evolved DNNs for recognizing 2 to 5 classes of digits are shown in Figure 7 and the corresponding specification of the topologies is shown in Table 8. As can be seen in Figure 7 and Table 8, the evolution on MNIST dataset achieves the DNNs with few parameters in thousands at the satisfied accuracy.

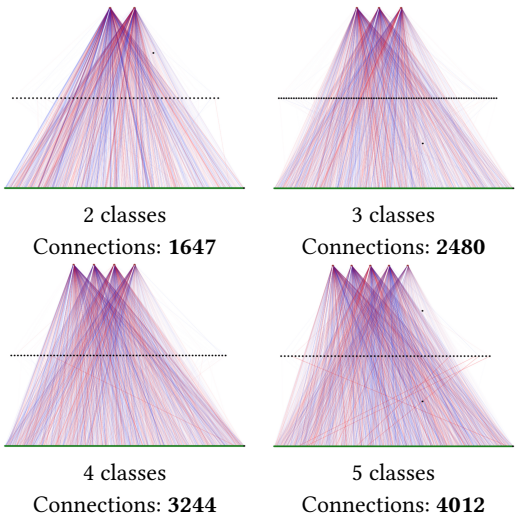

Figure 7: Four topologies of best DNNs in 2000 generations for 2, 3, 4, 5 classes of digits respectively. The red connections indicate positive weights, blue connections indicate negative weights. The green points represent input nodes. The black points are hidden nodes.

Nodes / Classes / Layer	2 (0,1)	3 (0~2)	4 (0~3)	5 (0~4)
Input layer	784+1	784+1	784+1	784+1
Hidden layer 1	52	1	66	1
Hidden layer 2	1	88	-	59
Hidden layer 3	-	-	-	1
Output layer	2	3	4	5
Total connections	1647	2480	3244	4012

Table 8: Illustration of the topologies of best evolved DNNs that recognizing 2 to 5 classes of digits.

5 CONCLUSION

In this paper, we present an approach to evolve the efficient DNNs with fewer parameters for real-time object recognition on LPCH. Our approach reduces the parameters of DNNs in two ways. First, NEAT algorithm is used to evolve DNNs starting from the elementary initial topology where all input nodes are connected to all output nodes. This often reduces the parameters of DNNs from millions even billions to thousands. However, the NEAT generally increase the topology of DNNs by adding nodes and connections for achieving the higher accuracy. The evolved DNNs with thousands of parameters may still contain many redundant parameters. Therefore, the evolved DNNs need to be further selected by fitness functions. Second, we therefore propose the fitness functions that consist of computation time and accuracy. In such way, the evolved DNNs are further encouraged for both fewer parameters and high accuracy. Finally, our approach successfully evolves DNNs that achieve an accuracy of 95.6% and a speed at 5 fps on LPCH, Raspberry Pi 3, for recognizing the modular robots. Furthermore, we test the approach on 5 classes digits of MNIST dataset. The results show that our approach successfully evolves DNNs with fewer parameters and reasonable accuracy for recognizing the handwritten digits.

This work aims to provide a method that can evolve the efficient DNNs with fewer parameters and a satisfied accuracy for real-time object recognition on LPCH rather than challenging the state-of-the-art in accuracy. While this work successfully achieves the goal that evolving DNNs with fewer parameters and desired accuracy for real-time object recognition on LPCH, there are still some interesting ideas to further study in the further. First, we observe that NEAT algorithm shows remarkable performance on evolving the topology of DNNs. But the evolution on weights of evolved DNNs is observed difficult for the convergence towards an remarkable accuracy. Therefore, a hybrid approach to combine the strength of NEAT and back-propagation could be explored in future research, which is promising to surpass the accuracy achieved in this work. Furthermore, reducing the dimension of the feature with new extractor is also important for further work which reduces the search space of NEAT algorithm.

ACKNOWLEDGMENTS

The authors would like to thank Dr. Diederik M. Roijers and Dr. Jadran Sirotkovic for their valuable comments and helpful suggestions.

REFERENCES

[1] Filipe AssunÃğão, Nuno LourenÃğo, Penousal Machado, and Bernardete Ribeiro. 2018. DENSER: deep evolutionary network structured representation. *Genetic Programming and Evolvable Machines* (2018), 1–31.

[2] Yu Cheng, Felix X. Yu, Rogerio S. Feris, Sanjiv Kumar, Alok Choudhary, and Shi-Fu Chang. 2015. An Exploration of Parameter Redundancy in Deep Networks with Circulant Projections. In *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV) (ICCV '15)*. IEEE Computer Society, Washington, DC, USA, 2857–2865.

[3] A. Cully and J. . Mouret. 2016. Evolving a Behavioral Repertoire for a Walking Robot. *Evolutionary Computation* 24, 1 (March 2016), 59–88.

[4] N. Dalal and B. Triggs. 2005. Histograms of oriented gradients for human detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, Vol. 1. 886–893 vol. 1.

[5] Travis Desell. 2017. Large Scale Evolution of Convolutional Neural Networks Using Volunteer Computing. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion (GECCO '17)*. ACM, New York, NY, USA, 127–128. https://doi.org/10.1145/3067695.3076002

[6] Marc Ebner. 2009. A Real-Time Evolutionary Object Recognition System. In *Genetic Programming*, Leonardo Vanneschi, Steven Gustafson, Alberto Moraglio, Ivanoe De Falco, and Marc Ebner (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 268–279.

[7] A.E. Eiben and J. Smith. 2015. From evolutionary computation to the evolution of things. *Nature* 521, 7553 (May 2015), 476–482.

[8] B. Evans, H. Al-Sahaf, B. Xue, and M. Zhang. 2018. Evolutionary Deep Learning: A Genetic Programming Approach to Image Classification. In *2018 IEEE Congress on Evolutionary Computation (CEC)*. 1–6. https://doi.org/10.1109/CEC.2018.8477933

[9] Chrisantha Fernando, Dylan Banarse, Charles Blundell, Yori Zwols, David Ha, Andrei A. Rusu, Alexander Pritzel, and Daan Wierstra. 2017. PathNet: Evolution Channels Gradient Descent in Super Neural Networks. *CoRR* abs/1701.08734 (2017).

[10] R. Girshick, J. Donahue, T. Darrell, and J. Malik. 2014. Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*. 580–587.

[11] Evert Haasdijk, Andrei A. Rusu, and A. E. Eiben. 2010. HyperNEAT for Locomotion Control in Modular Robots. In *Proceedings of the 9th International Conference on Evolvable Systems: From Biology to Hardware (ICES'10)*. Springer-Verlag, Berlin, Heidelberg, 169–180.

[12] Song Han, Jeff Pool, John Tran, and William Dally. 2015. Learning both Weights and Connections for Efficient Neural Network. In *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett (Eds.). Curran Associates, Inc., 1135–1143.

[13] M. Hausknecht, J. Lehman, R. Miikkulainen, and P. Stone. 2014. A Neuroevolution Approach to General Atari Game Playing. *IEEE Transactions on Computational Intelligence and AI in Games* 6, 4 (Dec 2014), 355–366.

[14] Jacqueline Heinerman, Alessandro Zonta, Evert Haasdijk, and Agoston Endre Eiben. 2016. On-line evolution of foraging behaviour in a population of real robots. In *European Conference on the Applications of Evolutionary Computation*. Springer, 198–212.

[15] M. Jelisavcic, M. de Carlo, E. Hupkes, P. Eustratiadis, J. Orlowski, E. Haasdijk, J. E. Auerbach, and A. E. Eiben. 2017. Real-World Evolution of Robot Morphologies: A Proof of Concept. *Artificial Life* 23, 2 (May 2017), 206–235.

[16] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger (Eds.). Curran Associates, Inc., 1097–1105.

[17] Gongjin Lan, Jesús Benito-Picazo, Diederik. M. Roijers, Enrique DomÃŋnguez, and A. E. Eiben. 2018. Real-Time Robot Vision on Low-Performance Computing Hardware. In *2018 15th International Conference on Control, Automation, Robotics and Vision (ICARCV)*. 1959–1965.

[18] Gongjin Lan, Milan Jelisavcic, Diederik M. Roijers, Evert Haasdijk, and A. E. Eiben. 2018. Directed Locomotion for Modular Robots with Evolvable Morphologies. In *Parallel Problem Solving from Nature – PPSN XV*. Springer International Publishing, Cham, 476–487.

[19] Yann LeCun. 1998. The MNIST database of handwritten digits. *http://yann.lecun.com/exdb/mnist/* (1998).

[20] Shaohui Lin, Rongrong Ji, Chao Chen, and Feiyue Huang. 2017. ESPACE: Accelerating Convolutional Neural Networks via Eliminating Spatial and Channel Redundancy.. In *AAAI*. 1424–1430.

[21] Baoyuan Liu, Min Wang, Hassan Foroosh, Marshall Tappen, and Marianna Pensky. 2015. Sparse Convolutional Neural Networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[22] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. 2016. SSD: Single Shot MultiBox Detector. In *Computer Vision – ECCV 2016*, Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling (Eds.). Springer International Publishing, Cham, 21–37.

[23] Christos Louizos, Karen Ullrich, and Max Welling. 2017. Bayesian Compression for Deep Learning. *Conference on Neural Information Processing Systems (NIPS)* (2017).

[24] Ping Luo, Yonglong Tian, Xiaogang Wang, and Xiaoou Tang. 2014. Switchable Deep Network for Pedestrian Detection. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[25] H. Mao, S. Yao, T. Tang, B. Li, J. Yao, and Y. Wang. 2017. Towards Real-Time Object Detection on Embedded Systems. *IEEE Transactions on Emerging Topics in Computing* PP, 99 (2017), 1–1.

[26] Risto Miikkulainen, Jason Liang, Elliot Meyerson, Aditya Rawal, Daniel Fink, Olivier Francon, Bala Raju, Hormoz Shahrzad, Arshak Navruzyan, Nigel Duffy, et al. 2019. Evolving deep neural networks. In *Artificial Intelligence in the Age of Neural Networks and Brain Computing*. Elsevier, 293–312.

[27] W. Ouyang and X. Wang. 2013. Joint Deep Learning for Pedestrian Detection. In *2013 IEEE International Conference on Computer Vision*. 2056–2063.

[28] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. 2016. You Only Look Once: Unified, Real-Time Object Detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 779–788.

[29] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. 2017. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *IEEE Trans. Pattern Anal. Mach. Intell.* 39, 6 (June 2017), 1137–1149.

[30] Sebastian Risi and Julian Togelius. 2015. Neuroevolution in Games: State of the Art and Open Challenges. *IEEE Transactions on Computational Intelligence and AI in Games* 9 (10 2015).

[31] Karen Simonyan and Andrew Zisserman. 2014. Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv 1409.1556* (09 2014).

[32] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. 2012. Practical Bayesian Optimization of Machine Learning Algorithms. In *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger (Eds.). Curran Associates, Inc., 2951–2959.

[33] K. O. Stanley, B. D. Bryant, and R. Miikkulainen. 2005. Real-time neuroevolution in the NERO video game. *IEEE Transactions on Evolutionary Computation* 9, 6 (Dec 2005), 653–668.

[34] Kenneth O Stanley, Jeff Clune, Joel Lehman, and Risto Miikkulainen. 2019. Designing neural networks through neuroevolution. *Nature Machine Intelligence* 1, 1 (2019), 24–35.

[35] K. O. Stanley, D. B. D'Ambrosio, and J. Gauci. 2009. A Hypercube-Based Encoding for Evolving Large-Scale Neural Networks. *Artificial Life* 15, 2 (April 2009), 185–212.

[36] Kenneth O. Stanley and Risto Miikkulainen. 2002. Evolving Neural Networks Through Augmenting Topologies. *Evol. Comput.* 10, 2 (June 2002), 99–127.

[37] Yi Sun, Xiaogang Wang, and Xiaoou Tang. 2015. Deeply learned face representations are sparse, selective, and robust. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2015), 2892–2900.

[38] Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf. 2014. DeepFace: Closing the Gap to Human-Level Performance in Face Verification. *2014 IEEE Conference on Computer Vision and Pattern Recognition* (2014), 1701–1708.

[39] S. Tripathi, G. Dane, B. Kang, V. Bhaskaran, and T. Nguyen. 2017. LCDet: Low-Complexity Fully-Convolutional Neural Networks for Object Detection in Embedded Systems. In *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. 411–420.

[40] Gábor Vásárhelyi, Csaba Virágh, Gergő Somorjai, Tamás Nepusz, Agoston E. Eiben, and Tamás Vicsek. 2018. Optimized flocking of autonomous drones in confined environments. *Science Robotics* 3, 20 (2018).

[41] O Rebecca Vincent and Olusegun Folorunso. 2009. A descriptive algorithm for sobel image edge detection. In *Proceedings of Informing Science & IT Education Conference (InSITE)*, Vol. 40. Informing Science Institute California, 97–107.

[42] B. Wang, Y. Sun, B. Xue, and M. Zhang. 2018. Evolving Deep Convolutional Neural Networks by Variable-Length Particle Swarm Optimization for Image Classification. In *2018 IEEE Congress on Evolutionary Computation (CEC)*. 1–8. https://doi.org/10.1109/CEC.2018.8477735

[43] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. 2016. Learning Structured Sparsity in Deep Neural Networks. In *Advances in Neural Information Processing Systems 29*, D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett (Eds.). Curran Associates, Inc., 2074–2082.

[44] L. Xie and A. Yuille. 2017. Genetic CNN. In *2017 IEEE International Conference on Computer Vision (ICCV)*. 1388–1397. https://doi.org/10.1109/ICCV.2017.154

[45] Jason Yosinski, Jeff Clune, Diana Hidalgo, Sarah Nguyen, Juan Cristobal Zagal, and Hod Lipson. 2011. Evolving robot gaits in hardware: the HyperNEAT generative encoding vs. parameter optimization. In *In Proceedings of the 20th European Conference on Artificial Life*. 890–897.