

# A Review of Meta-Reinforcement Learning for Deep Neural Networks Architecture Search

Yesmina Jaafr<sup>a,b,c</sup>, Jean Luc Laurent<sup>b</sup>, Aline Deruyver<sup>a</sup>, Mohamed Saber Naceur<sup>c</sup>,

<sup>a</sup>*ICube Laboratory, Universit de Strasbourg, 300 bd Sbastien Brant, 67412 Illkirch, France*

<sup>b</sup>*Segula Technologies, Parc dactivit de Pissaloup, 8 avenue Jean dAlembert, 78190*

*Trappes, France*

<sup>c</sup>*LTSIRS Laboratory, ENIT, 1002 Tunis, Tunisie*

---

## Abstract

Deep Neural networks are efficient and flexible models that perform well for a variety of tasks such as image, speech recognition and natural language understanding. In particular, convolutional neural networks (CNN) generate a keen interest among researchers in computer vision and more specifically in classification tasks. CNN architecture and related hyperparameters are generally correlated to the nature of the processed task as the network extracts complex and relevant characteristics allowing the optimal convergence. Designing such architectures requires significant human expertise, substantial computation time and doesnt always lead to the optimal network. Model configuration topic has been extensively studied in machine learning without leading to a standard automatic method. This survey focuses on reviewing and discussing the current progress in automating CNN architecture search.

*Keywords:* Deep Learning, Automatic Design, Reinforcement Learning, Meta-Learning, AutoML.

---

## 1. Introduction

”A neuron is nothing more than a switch with information input and output. The switch will be activated if there are enough stimuli of other neurons hitting the information input. Then, at the information output, a pulse is sent to, for example, other neurons ” [1]. Brain-inspired machine learning imitates in a simplified manner the hierarchical operating mode

of biological neurons [2]. The concept of artificial neural networks (ANN) achieved a huge progress from its first theoretical proposal in the 1950s until the recent considerable outcomes of deep learning. In computer vision and more specifically in classification tasks, CNN, which we will examine in this review, are among the most popular deep learning techniques since they are outperforming humans in some vision complex tasks [3].

The origin of CNN that were initially established by [4] goes back to the 1950s with the advent of "perceptron", the first neural network prototyped by Frank Rosenblatt. However, neural network models were not extensively used until recently, after researchers overcame certain limits. Among these advances we can mention the generalization of perceptrons to many layers [5], the emergence of backpropagation algorithm as an appropriate training method for such architectures [6] and, mainly, the availability of large training datasets and computational resources to learn millions of parameters. CNN differ from classical neural networks in the fact that the connectivity of a hidden layer neuron is limited to a subset of neurons in the previous layer. This selective connection endow the network with the ability to operate, implicitly, hierarchical features extraction. For an image classification case, the first hidden layer can visualize edges, the second a specific shape and so on until the final layer that will identify the object.

CNN architecture consists of several types of layers including convolution, pooling, and fully connected. The network expert has to make multiple choices while designing a CNN such as the number and ordering of layers, the hyperparameters for each type of layer (receptive field size, stride, etc.). Thus, selecting the appropriate architecture and related hyperparameters requires a trial and error manual search process mainly directed by intuition and experience. Additionally, the number of available choices makes the selection space of CNN architectures extremely wide and impossible for an exhaustive manual exploration. Many research effort in meta-modeling tries to minimize human intervention in designing neural network architectures. In this paper, we first give a general overview and define the field of deep learning. We then briefly survey the history of CNN architectures. In the following section we review several methods for automating CNN design according to three dimensions: search optimization, architecture design methods (plain or modular) and search acceleration techniques. Finally, we conclude the article with a discussion of future works.

## 2. Background

Before embarking with CNN, we will introduce in this section some basic generalities about artificial networks and deep learning.

### 2.1. Artificial Neural Networks

ANN are a major field of artificial intelligence that attempts to replicate human brain processing. Three types of neural layers distinguish an ANN: input, output and hidden layers. The latter operate transitional representations of the input data evolving from low level features (lines and edges) to higher ones (complex patterns) as far as deeper layers are reached. Figure 1 provide an example of ANN involving classical fully-connected layers where every neuron is connected to all ones of the previous layer.

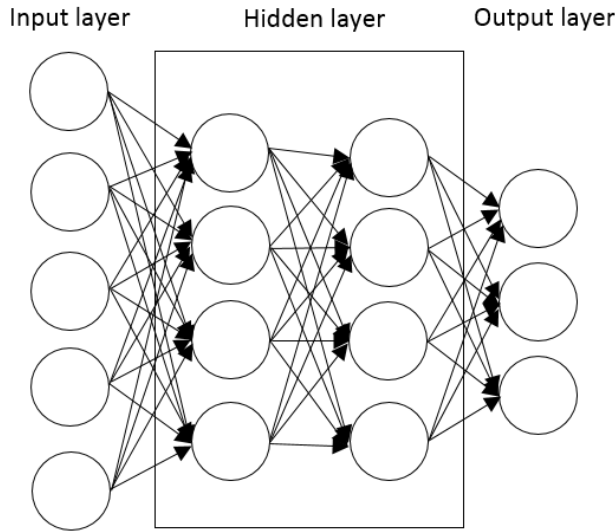


Figure 1: Artificial neural network, containing an input layer, an output layer and two hidden layers.

During training, an ANN aims at learning two types of parameters that will condition its predictive performance. First, connection weights that assess to which extent a neuron result will impact the output of higher level neuron. Second, the bias which is a global estimator of a feature presence across all inputs. Hence, a neuron output can be formalized through a linear combination of weighted inputs and associated bias:

$$output = (\sum_i input_i * weight_i) + bias$$

In order to allow the network operating non-linear transformations, an activation function is applied to the previous output. Equation 1 presents an example of such transformation using one of the most common and efficient activation function which is the Rectified Linear Unit (ReLU) [7]:

$$f(x) = \max(x, 0) \quad (1)$$

## 2.2. Deep learning

The concept of deep learning refers to machine learning processing within multi-layer ANN [8]. The training of these networks relies on a loss function evaluation. For example, in supervised learning the loss is assimilated to the matching accuracy between ANN predictions and real expected outputs. An iterative update procedure is implemented to adjust network parameters according to loss function computed gradient. This procedure is called Back-propagation since parameters updates are spread from final layers to initial ones. Deep learning implies a certain number of challenges such as vanishing/exploding gradient and overfitting. The solutions to these problems will be discussed when developing CNN design architectures in next sections.

## 3. CNN Layers

CNN are widely used in a great number of pattern and image recognition problems. Three main characteristics are making this deep learning technique successful and suitable to visual data. First, local receptive fields perfectly reflect image data specificity to be correlated locally and uncorrelated in global segments. Second, shared weights allows a substantial parameter reduction without altering image processing since the convolution is applicable to the whole image. Last, grid-structured image enable pooling operations that simplify data without losing useful information [9].

### 3.1. Convolutional Layer

The convolutional layer is the basic CNN unit that has been inspired by physiological research evidence of hierarchical processing in the visual cortex of mammals [10]. Simple cells detect primitive attributes while more

compound structures are subsequently extracted by complex cells. Thus, convolutional layer consists of a set of feature maps issued from convolving different filters (kernels) with an input image or previous layer output [11]. The 2-dimensional maps are stacked together to produce the resulting volume of the convolutional layer. This process reduces drastically the network complexity since the neurons of a same feature map share the same weights and bias maintaining a low number of parameters to learn [12].

The hyperparameters characterizing a convolutional layer are the depth  $F$  (number of filters), the stride  $S$  (filter movement from a receptive field to the next one) and the zero padding  $P$  to control input size [13]. Assuming that the filter *size* (*height, width, depth*) =  $(h, w, D)$ , the dimensions of the feature maps generated can be obtained according to:

$$(H_1, W_1, F) = ((H + 2P - h)/S + 1, (W + 2P - w)/S + 1, F)$$

Where  $(H, W, D)$  is the size (height, width, depth) of the input image.

### 3.2. Pooling Layer

CNN architectures generally alternate convolution and pooling layers. The latter have the purpose of reducing network complexity and avoid the problem of overfitting. At biological level, pooling is assimilated to the behavior of cortical complex cells that reveal a certain degree of position invariance. A pooling layer neuron is connected to a region of the previous layer by performing a non-parameterized function. Thus it differs from convolution as it doesn't have learnable weights or bias and additionally, it keeps the same depth of the previous layer. Max pooling [14] is one of the most common type of pooling that consists in retaining the maximum value of a neurons cluster. It means that max pooling is detecting if a given feature has been identified in a receptive field without recording the exact location [9].

### 3.3. Fully connected Layer

The convolution layers identify local features in the input data such as edges and shapes. The Fully connected layer operates the high level reasoning (classification for image case) by combining information from all the previous layers. As in a regular ANN, neurons at this level are fully connected to all ones in the previous layer. A softmax loss layer is then used to compute the probability distribution of the CNN final outputs.

## 4. CNN Architecture History

This section presents the most influential hand-crafted CNN architectures that have impacted the recent work on automatic architecture design. Most of them won at least one of the "ImageNet Large Scale Visual Recognition Competition" (ILSVRC) challenges [3].

### 4.1. LeNet

As mentioned previously, LeNet [7] was the innovative work that introduced convolutional networks. The model was experimented successfully to classify handwritten digits without any preprocessing of the input image (of size  $32 * 32$  pixels). LeNet architecture is illustrated in figure 2. It consists of an input and an output layers of respective sizes  $32 * 32$  and 10 as well as 6 hidden Layers. The basic idea of this design is to operate multiple convolutions (3) with pooling in-between (2) then transmitting the final signal via a fully-connected layer toward the output layer. Unfortunately, due to the lack of adequate training data and computing power, it wasn't possible to extend this architecture to more complex applications.

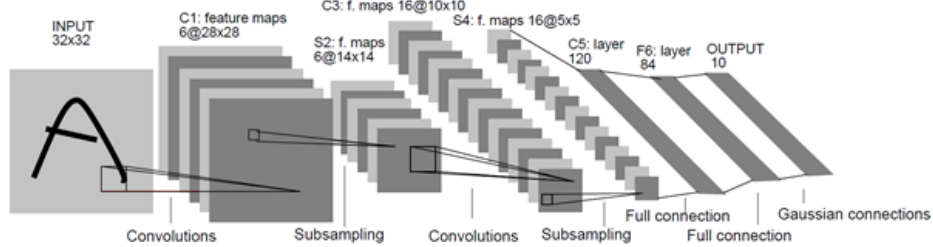


Figure 2: Architecture of LeNet-5 [7].

### 4.2. AlexNet

AlexNet [11] is one of the most influential deep CNN that won the ILSVRC (Imagenet Large Scale Visual Recognition Challenge) competitions in 2012. As shown in figure 3, it is not much different from LeNet. Nevertheless, the corresponding architecture is deeper with 8 layers in total, 5 convolutional and 3 fully connected. The effective contribution of AlexNet lies in several design and training specificities. First, it introduced the Rectified Linear Unit (ReLU) nonlinearity which helped to overcome the problem of

vanishing gradient and boosted a faster training. Furthermore, AlexNet implements a dropout step [15] that consists in setting to zero a predefined percentage of layers' parameters. This technique decreases learned parameters and controls neurons correlation in order to limit overfitting impact. Third, training process convergence is accelerated with momentum and conditional learning rate decrease (e.g. when learning stagnates). Finally, training data volume is increased artificially by generating variations of the original images that are shifted randomly. Thus the network learning is enhanced with the use of invariant representations of the data.

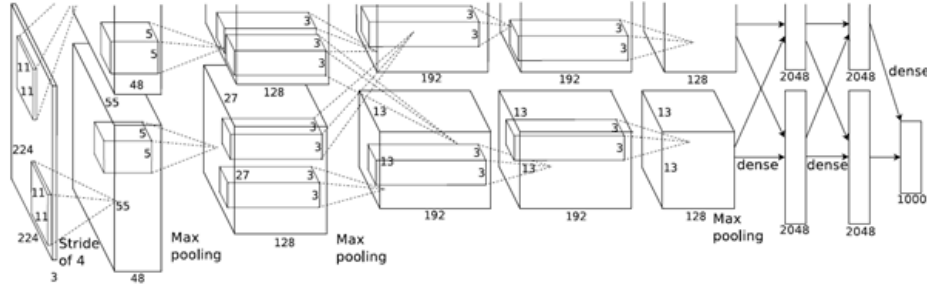


Figure 3: An illustration of AlexNet [11].

#### 4.3. VGGNet

Submitted for the ILSVRC 2014, VGGNet [16] won the second place and demonstrated that deeper architectures achieve better results. Indeed, with its 19 hidden layers, it was much deeper than previous convolutional networks. In order to allow an increase in depth without an exponential growth of the parameters number, smaller convolution filters ( $3 \times 3$ ) were used in all layers (e.g. lower size than the  $11 \times 11$  filters adopted in AlexNet). An additional advantage of using smaller filters consists in reducing overlapping scanned pixels which results in feature maps with more local details [17].

#### 4.4. GoogLeNet

Since it has been demonstrated that a CNN architecture size is positively correlated to its performance, recent efforts focus on how to increase the depth of a CNN while keeping an acceptable number of parameters. Winner of ILSVRC 2014, GoogLeNet [18] innovated network design by replacing the classical strategy of alternating convolutional and pooling layers with stacked Inception Modules depicted in figure 4. Despite being deeper than VGGNet

with 22 hidden layers, GoogLeNet requires outstandingly fewer parameters due to this sparse connection technique. Within an inception module, several convolutions with different scales and pooling are performed in parallel then concatenated in one single layer. This enables the CNN to detect patterns of various sizes within the same layer and avoid heavy parameters redundancies [18]. GoogLeNet hidden layers consist of 3 convolutions, 9 inception blocks (2 layers deep each one) and one fully connected.

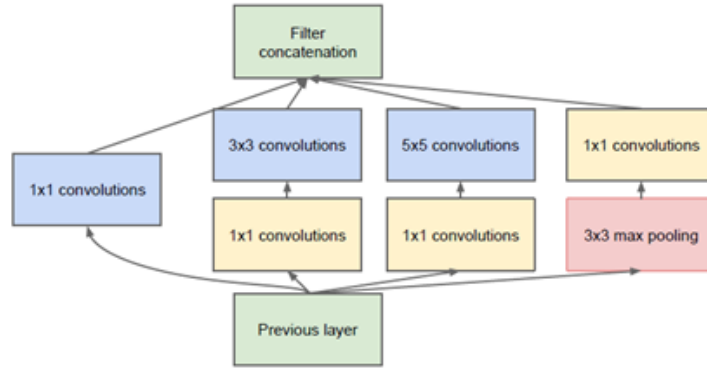


Figure 4: Inception module [18].

#### 4.5. ResNet

Deep Residual Network (ResNet) [19], was the first neural network to exceed human-level accuracy in ImageNet Challenge (ILSVRC 2015). Thanks to residual connections, such kind of architecture went deeper and was implemented with multiple versions of 34, 50, 101 and 152 layers. Indeed, one of the difficulties with very deep networks training is the vanishing gradient during error backpropagation which penalizes the appropriate update of earlier layers weights. ResNet main contribution consists in dividing convolutional layers into residual blocks. Each block is bypassed by a residual (skip) connection that forwards the block input using an identity mapping. The final output is the summation of the block output and the mapped input as illustrated in figure 5.

By adding skip connections, backpropagation can be operated without any interference with previous layers which allows to prevent vanishing gradient and train very deep architectures. ResNet-101 consists of one convolutional layer followed by 33 residual blocks (3 layers deep each one), and one fully connected layer.



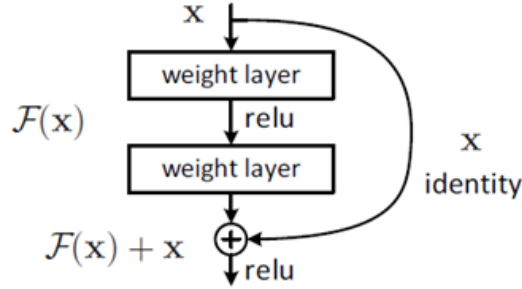


Figure 5: Residual learning: a building block [19]

#### 4.6. More Networks

After ResNet [19] success, which exceeded human-level accuracy in (ILSVRC 2015), the so-called modern hand-crafted CNN are still being designed on the basis of previous models looking for more efficiency and lower training time. Inception-v4 [20] is a new release of GoogLeNet that involved many more layers than the initial version. Inception-ResNet [20] is built as a combination of an Inception network and a ResNet, joining inception blocks and residual connections. The last example of this section is DenseNet (Dense Convolutional Networks) [21] where each dense block layer is connected via skip connections to all subsequent ones allowing the learning of new features.

### 5. Meta-modeling for CNN automatic architecture design

Meta-modeling for neural network architectures design aims at reducing the intervention of human expertise in this process. The earliest meta-modeling methods were based on genetic algorithms and Bayesian optimization then more recently, reinforcement learning became among the most implemented approaches [22].

#### 5.1. Context of automation

The performance of a neural network and particularly a CNN mainly depends on the setting of the model structure, the training process, and the data representation. All of these variables are controlled through a number of hyperparameters and impact the learning process to a large extent. In order to achieve an optimal performance of CNN, these hyperparameters including the depth of the network, learning rates, layer type, number of units per layer, dropout rates, etc., should be then carefully tuned. On the

other hand, the advent of deeper and more complex modern architectures (see section 4) is increasing the number and the types of hyperparameters. Hence, tuning step and more generally CNN architectures search become very expensive and heavy for an expert trial-and-error procedure.

Additionally, CNN parameters setting is considered as a black-box [23] optimization problem because of the unknown nature of the mapping between the architecture, the performance, and the learning task. In this context, automatic design solutions are highly required and instigates a large volume of research. The task of CNN hyperparameters tuning has been handled through meta-modeling that consists in applying machine learning models for designing CNN architectures. Three meta-modeling approaches are generally used in the literature of architecture search and will be described in the next paragraph: bayesian optimization, evolutionary algorithms and reinforcement learning.

## 5.2. Meta-controllers

Meta-modeling approaches perform iterative selection from the hyperparameters space and build associated architectures that are then trained and evaluated. Accuracies records are fed to meta-modeling controllers (meta-controllers) to guide next architectures sampling. Meta-controllers for CNN design are mainly based on bayesian optimization ([24], [25]), evolutionary algorithms ([26], [27]) or more recently on reinforcement learning ([28], [29]).

Bayesian optimization is an efficient way to optimize black-box objective functions  $f : X \rightarrow R$  that are slow to evaluate [30]. It aims at finding an input  $x = \arg \min_{x \in X} f(x)$  that globally minimizes  $f$  where in the context of a machine learning algorithm,  $x$  refers to the set of hyperparameters to optimize. The problem with this kind of optimization is that evaluating the objective function is very costly due to the great number of hyperparameters and the complex nature of models like deep neural networks. In order to overcome this problem, bayesian approaches propose probabilistic surrogate reconstruction of the objective function  $p(f|D)$  where  $D$  is a set of past observations. The evaluation of the empirical function is much cheaper than the true objective function [31]. Some of the most used probabilistic surrogate (regression) models are gaussian processes [32], random forests [33] and tree-structured Parzen estimator [24].

Briefly, the processing of a bayesian optimization consists in building an empirical (probabilistic) model of the objective function. Then, iteratively, the model identifies a set of optimal hyperparameters for which the objective

function returns corresponding results (e.g. loss values). Each feedback allows the update of the surrogate model and the guidance of hyperparameters predictions until the process reaches a termination condition.

Evolutionary algorithms present another strategy of hyperparameters optimization that modifies a set of candidate solutions (population) on the basis of a number of rules (operators). Following an iterative procedure of mutation, crossover and selection [34], an evolutionary algorithm initializes, in a first step, a set of  $N$  random networks to create a primary population. The second step consists in introducing a fitness function to score each network through its classification accuracy and keep the top ranked networks to construct the next generation. The evolutionary process continues until a termination criteria is met, which is generally defined as the maximum number of allowed generations. One of the advantage of evolutionary algorithms is the adaptation to complex combination of discrete (layer type) and continuous (learning rate) hyperparameters which is suitable to neuronal network optimization models [35].

An important approach for goal-oriented optimization is reinforcement learning (RL) inspired from behaviorist psychology [36]. The frame of RL is an agent learning through interaction with its environment (figure 6). Thus the agent adapts its behavior (transition to a state  $s_{t+1}$ ) on the basis of observed consequences (rewards) of an action  $a_t$  taken in state  $s_t$ . The agent purpose is to learn a policy  $\pi$  that is able to identify the optimal sequence of actions maximizing the expected cumulative rewards. The environment return reinforces the agent to select new actions to improve learning process, hence the name of reinforcement learning.

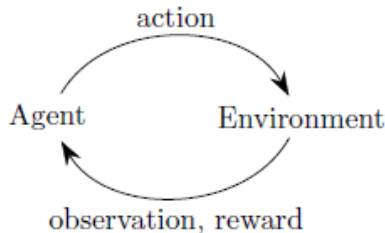


Figure 6: Illustration of the RL process.

The methods developed to resolve reinforcement tasks are based on value functions, policy search or a combination of both strategies (actor-critic methods) [37]. Value function methods consist in estimating the expected

reward value  $R$  when reaching a given state  $s$  and following a policy  $\pi$ :

$$V^\pi(s) = \mathbb{E}[\mathcal{R}|s, \pi]$$

A recursive form of this function is particularly used in recent Q-learning [38] models assigned to CNN architecture design ([39], [40]):

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$$

Where  $s_t$  is a current state,  $a_t$  is a current action,  $\alpha$  is the learning rate,  $r_{t+1}$  is the reward earned when transitioning from time  $t$  to the next and  $\gamma$  is the discount rate.

In contrast to value function methods, policy search methods do not implement a value function and apply, instead, a gradient-based procedure to identify directly an optimal policy  $\pi^*$ . In this context, deep reinforcement learning is achieved when deep neural networks are used to approximate one of the reinforcement learning components : value function, policy or reward function [41].

Among the active fields of designing CNN architectures through deep reinforcement learning, recurrent neural networks (RNN) arise as a valuable model that handles a set of tasks such as hyperparameters prediction ([28], [29]). In fact, a RNN operates sequentially involving hidden units to store processing history, which allows the reinforcement learning to profit from past observations. Long short term memory networks (LSTM), a variant of RNN, offers a more efficient way of evolving conditionally on the basis of previous elements.

## 6. Neural Architecture Search

Various strategies have been developed to operate CNN architectures design for the majority of which reinforcement learning has been selected as meta-controller. This section is assigned to review in detail most recent promising automatic search approaches differentiated according to search spaces specificities and complexity level.

### 6.1. Plain Architecture Design

Some architecture search approaches focus on designing plain CNN which consists exclusively of conventional layers, mainly convolution, pooling and fully-connected. The resulting research space is relatively simple and the approaches contribution lies almost entirely in the design strategy.

### 6.1.1. MetaQNN

MetaQNN model [39] relies on Q-learning, a type of reinforcement learning (refer to previous section for more details), to sequentially select network layers and their parameters among a finite space. This method implies, first, the definition of each learning agent state as a layer with all associated relevant parameters. As an example, 5 layers are depicted in figure 7: convolution (C), pooling (P), fully connected (FC), global average pooling (GAP), and softmax (SM).

Layer Type	Layer Parameters	Parameter Values
Convolution (C)	$i \sim$ Layer depth	$< 12$
	$f \sim$ Receptive field size	Square. $\in \{1, 3, 5\}$
	$\ell \sim$ Stride	Square. Always equal to 1
	$d \sim$ # receptive fields	$\in \{64, 128, 256, 512\}$
	$n \sim$ Representation size	$\in \{(\infty, 8], (8, 4], (4, 1]\}$
Pooling (P)	$i \sim$ Layer depth	$< 12$
	$(f, \ell) \sim$ (Receptive field size, Strides)	Square. $\in \{(5, 3), (3, 2), (2, 2)\}$
	$n \sim$ Representation size	$\in \{(\infty, 8], (8, 4] \text{ and } (4, 1]\}$
Fully Connected (FC)	$i \sim$ Layer depth	$< 12$
	$n \sim$ # consecutive FC layers	$< 3$
	$d \sim$ # neurons	$\in \{512, 256, 128\}$
Termination State	$s \sim$ Previous State $t \sim$ Type	Global Avg. Pooling/Softmax

Figure 7: State space possible parameters [39].

Second, the agent action space is assimilated to the possible layers the agent may move to given a certain number of constraints set intentionally, for the majority, to enable faster convergence. Figure 8 illustrates a set of state and action spaces and an eventual agent path to design a CNN architecture. MetaQNN was evaluated competitive with similar and different hand-crafted CNN architectures as with existing automated network design methods.

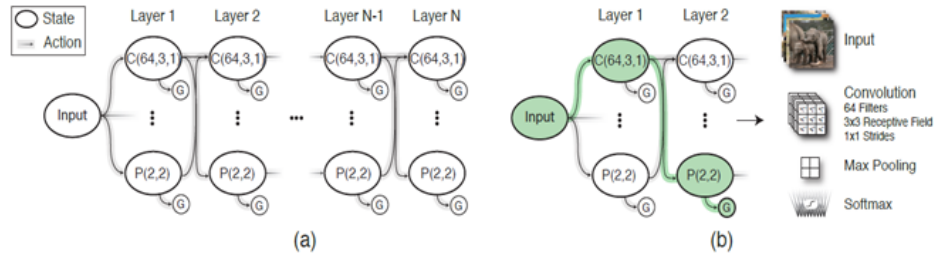


Figure 8: An illustration of the full state and action space (a) and a path that the agent has chosen (b) [39].

### 6.1.2. NAS

Using reinforcement learning, [28] train a recurrent neural network to generate convolutional architectures. Figure 9 shows a RNN controller generating sequentially CNN parameters associated to convolutional layers. Every sequence output is predicted by a softmax classifier then used as input of the next sequence. The parameters set consists of filter height and width, stride height and width and the number of filters per layer. The design of an architecture takes an end once the number of layers reaches a predefined value that increases all along training. The accuracy of the designed architecture is fed as a reward to train the RNN controller through reinforcement learning in order to maximize the expected validation accuracy of the next architectures. The experimentation of the global approach achieved competitive results on CIFAR-10 and Penn Treebank datasets.

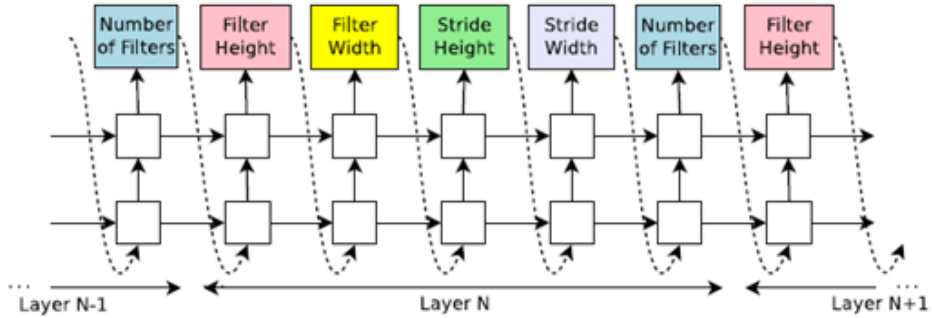


Figure 9: Illustration of the way the agent used to select hyperparameters [28].

### 6.1.3. EAS

In their very recent work Efficient Architecture Search, [42] implement network transformation techniques that allow reusing pre-existing models and efficiently exploring search space for automatic architecture design. This novel approach differs from the previous ones in the definition of reinforcement learning states and actions. The state is the current network architecture while the action involves network transformation operations such as adding, enlarging and deleting layers. Starting point architectures used in experiments are plain CNN which only consist of convolutional, fully-connected and pooling layers. EAS approach is inspired from Net2Net technique introduced in [43] and based on the idea of building deeper student network to reproduce the same processing of an associated teacher network. As shown

in figure 10, an encoder network implemented with bidirectional recurrent neural network [44] feeds actors network with given architectures. The selected actor networks performs 2 types of transformation: widening layers in terms of units and filters and inserting new layers. EAS outperforms similar state-of-the-art models designed either manually or automatically with the attractive advantage of using relatively much smaller computational resources.

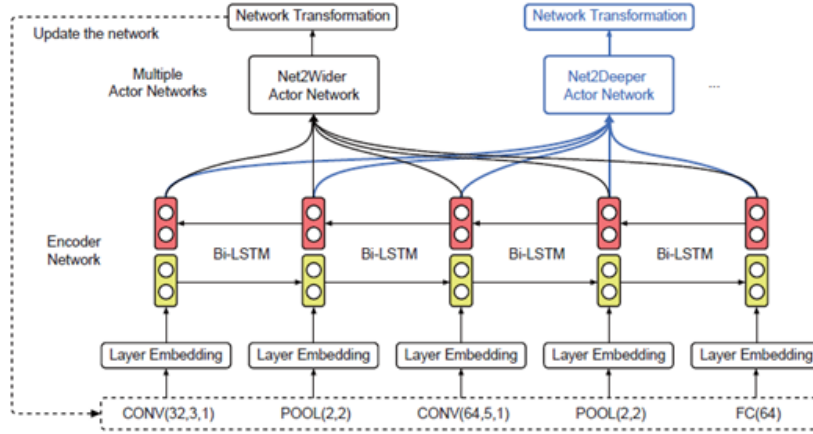


Figure 10: A meta-controller operation for network transformation [42].

## 6.2. Modular Architecture Design

Most of recent work on neural architecture search is based on more complex modular (multi-branch) structures inspired by modern architectures presented in section 4. Rather than operating the tedious search over entire networks, this second set of approaches focus on finding building blocks similarly to the ones used in, e.g. GoogLeNet and ResNet models. These multi-branch elements are then stacked repetitively involving skip connections to build the final deep architecture. As we will see through the models detailed in this section, "block-wise" architecture design reduces drastically search space speeding up search process, enhances generated networks performance and gives them more transferable ability through minor adaptation.

### 6.2.1. BlockQNN

One of the first approaches implementing block-wise architecture search, BlockQNN [40] automatically builds convolutional networks using Q-Learning

reinforcement technique [45] with epsilon-greedy as exploration strategy [46]. The block structure is similar to ResNet and Inception (GoogLeNet) modern networks since it contains shortcut connections and multi-branch layer combinations. The search space of the approach is reduced given that the focus is switched to explore network blocks rather than designing the entire network. The block search space is detailed in figure 11 and consists of 5 parameters: a layer index (its position in the block), an operation type (selected among 7 types commonly used), a kernel size and 2 predecessors layers indexes. Figure 12 depicts 2 different samples of blocks, one with multi-branch structure and the second showing a skip connection. As described in previous sections, the Q-learning model includes an agent, states and actions, where the state represents the current layer of the agent and the action refers to the transition to the next layer. On the basis of defined blocks, the complete network is constructed by stacking them sequentially  $N$  times.

Name	Index	Type	Kernel Size	Pred1	Pred2
Convolution	<b>T</b>	1	1, 3, 5	<b>K</b>	0
Max Pooling	<b>T</b>	2	1, 3	<b>K</b>	0
Average Pooling	<b>T</b>	3	1, 3	<b>K</b>	0
Identity	<b>T</b>	4	0	<b>K</b>	0
Elemental Add	<b>T</b>	5	0	<b>K</b>	<b>K</b>
Concat	<b>T</b>	6	0	<b>K</b>	<b>K</b>
Terminal	<b>T</b>	7	0	0	0

Figure 11: Network structure code space [40].

### 6.2.2. PNAS

Progressive neural architecture search [47] proposes to explore the space of modular structures starting from simple models then evolving to more complex ones, discarding underperforming structures as learning progresses. The modular structure in this approach is called a cell and consists of a fixed number of blocks. Each block is a combination of 2 operators among 8 selected ones such as identity, pooling and convolution. A cell structure is learned first then it's stacked  $N$  times in order to build the resulting CNN. The main contribution of PNAS lies in the optimization of the search process by avoiding direct search in the entire space of cells. This was made possible with the use of a sequential model-based optimization (SMBO) strategy.



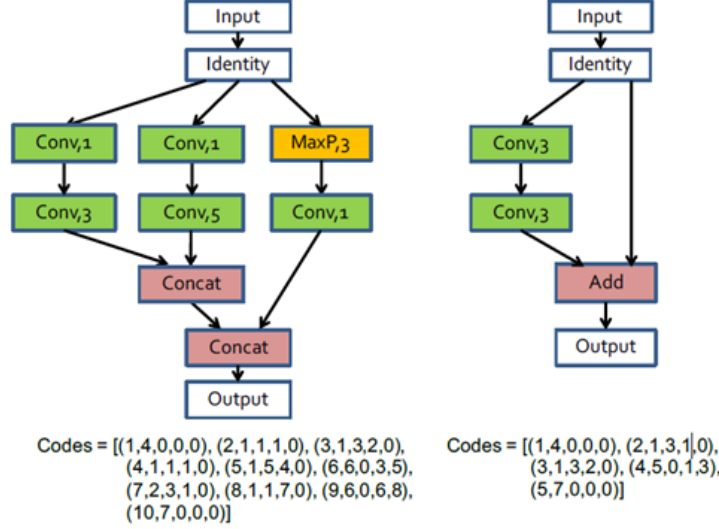


Figure 12: Representative block exemplars with their network structure codes [40].

The initial step consists in building, training and evaluating all possible 1-block cells. Then the cell is expanded to 2-block size exploding the number of total combinations. The innovation brought by PNAS is to predict the performance of the second level cells by training a RNN (predictor) on the performance of previous level ones. Only the  $K$  best cells (i.e. most promising ones) are transferred to the next step of cell size expansion. This process is repeated until the maximum allowed blocks number is reached. With an accuracy comparable to NAS [28] approach, PNAS is up to 5 times faster using a cell maximum size of 5 blocks and  $K$  equal to 256. This result is due to the fact that performance prediction takes much less time than full training of designed cells. The best cell architecture is shown in figure 13.

### 6.2.3. ENAS

Efficient neural architecture search [29] comes in the continuity of previous work NAS [28] and PNAS [47]. It explores a cell-based search space through a controller RNN trained with reinforcement learning. The cell structure is similar to PNAS model where block concept is replaced with a node that consists of 2 operations and two skip connections. The controller RNN manages thus 2 types of decisions at each node. First it identifies 2 previous nodes to connect to, allowing the cell to set skip connections. Second, the controller selects 2 operations to implement among a set of 1 identity, 2 depth

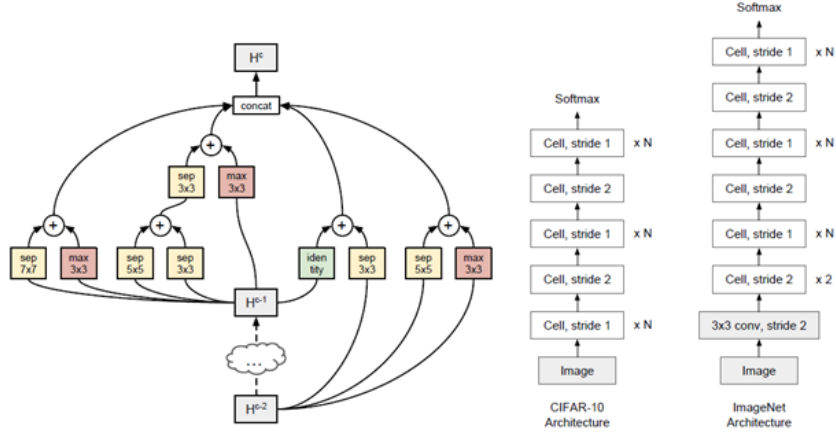


Figure 13: The best selected cell architecture of PNAS [47].

wise-separable convolutions of filter sizes  $3 * 3$  and  $5 * 5$  [48], max pooling and average pooling both of size  $3 * 3$ . Within each node, the operations results are added in order to constitute an input for the next node. Figure 14 illustrates the design of a 4-node cell. At the end, the entire CNN is built by stacking  $N$  times convolutional cells.

Another contribution of ENAS consists in sampling mini-batches from validation dataset to train designed models. The models with the best accuracy are then trained on the entire validation dataset. Additionally, the approach efficiency is greatly improved by implementing a weight sharing strategy. Each node has its own parameters (used when involved operations are activated) that are shared through inheritance by the generated child models. The latter are hence not trained from scratch saving a considerable processing time. ENAS provides competitive results on CIFAR-10 and Penn Treebank datasets. It specifically takes much less time to build the convolution cells than previous approaches that adopt the same strategy of designing modular structures then stack them to obtain a final CNN.

#### 6.2.4. EAS With Path Level Transformation

A developed version of EAS [42] which adopts network transformation for efficient CNN architecture search is presented in [49]. The new approach tackle the constraint of only performing plain architecture modification (layer-level), e.g. adding (removing) units, filters and layers, by using path-level transformation operations. The proposed model is similar to ([42])

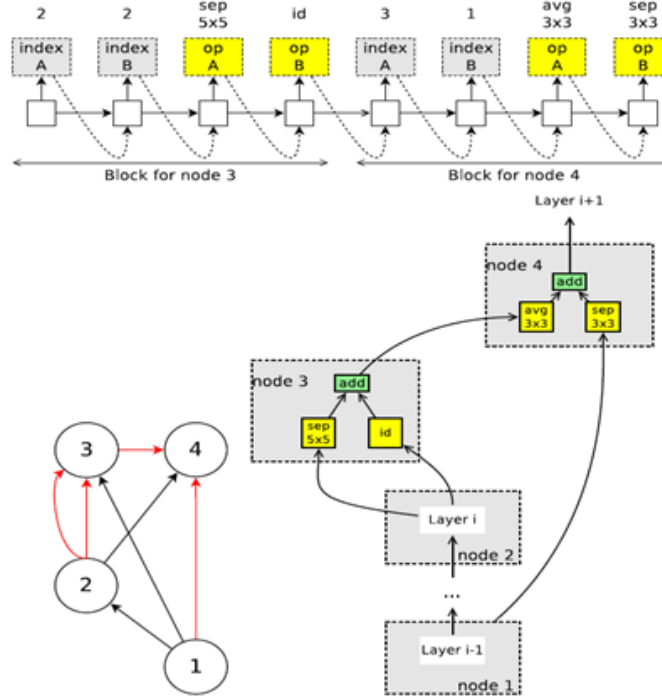


Figure 14: Illustration of 4-nodes cell [29].

where the reinforcement learning meta-controller samples network transformation actions to build new architectures. The latter are then trained and resulting accuracies are used as reward to update the meta-controller. However, certain changes have been implemented in order to adapt search methods to the tree-structured architecture space: using a tree-structured LSTM, ([50]) as meta-controller, defining a new action space consisting of feature maps allocation schemes (replication, skip), merge schemes (add, concatenation, none) and primitive operations (convolution, identity, depthwise-separable convolution, etc.). Figure 15 presents an example of transformation decisions operated by the meta-controller. Experimenting with ResNet and DenseNet architectures as base input, the path level transformation approach achieves competitive performance with state-of-the-art models maintaining low computational resources comparable to EAS approach ones.

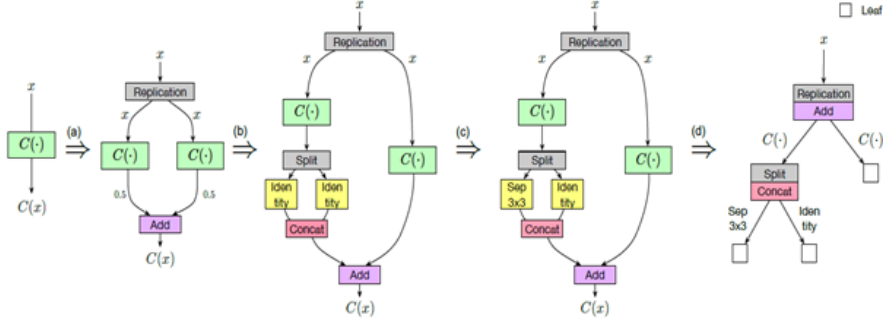


Figure 15: Path-level transformation: from a single layer to a tree-structured motif [49].

### 6.3. Architecture search accelerators

Reinforcement learning methods have been applied successfully to design neural networks. Although multi-branch structures and skip connections improves the efficiency of architectures automatic search, the latter is still computationally expensive (hundreds of GPU hours), time consuming and requires further acceleration of learning process. Thus, in addition to the methods assigned to architectural search optimization and complex component building, some techniques are developed to speed up learning and are depicted in the current section.

Early stopping strategy proposed in [40] enables fast convergence of the learning agent while maintaining an acceptable level of efficiency. This is possible by taking into account intermediate rewards ignored in previous works (set to zero delaying reinforcement learning convergence [36]). In such case, the agent stops searching in an early training phase as the accuracy rewards reach higher levels in fewer iterations. The reward function is redefined in order to include designed block complexity and density and avoid possible poor accuracy resulting from training early stopping.

A second technique is presented in [40] which consists of a distributed asynchronous framework assembling 3 nodes with different functions. The master node is the place where block structures are sampled by agent. Then, in the controller node, the entire network is built from generated blocks and transmitted to multiple compute nodes for training. The framework is a kind of simplified parameter-server [51] and allows the parallel training of designed networks in each compute nodes. Hence, the whole design and learning processing is operated in multiple machines and GPUs. [28] uses the same parameter server scheme with replication of controllers in order to

train various architectures in parallel.

As seen previously, reinforcement learning policies use explored architectures performance as a guiding reward for controllers updates. Training and evaluating every sampled architecture (among hundreds) on validation data is responsible for most of computational load. Extracting architecture performance was consequently subject to several estimation attempts. A number of approaches focus on performance prediction on the basis of past observations. Most of such techniques are based on learning curve extrapolation [25] and surrogate models using RNN predictor [52] that aim at predicting and eliminating poor architectures before full training. Another idea to estimate performance and rank designed architectures is to use simplified (proxy) metrics for training such as data subsets (mini-batches) [29] and down-sampled data (like images with lower resolution) [53].

Network transformation is one of the more recent techniques assigned to accelerate neural architecture search ([49], [54]). It consists in training explored architectures reusing previously trained or existing networks. This modeling feature allows to address a limitation of reinforcement learning approaches where training is performed with a random initialization of weights. Thus, extending network morphisms [55] to initiate architecture search through the transfer of experience and knowledge reflected by reused weights enables the framework to scrutinize the search space efficiently.

Although the techniques presented above have saved substantial computational resources for neural architecture search, there is still more effort needed to examine the extent of bias impact of such techniques on the search process. Indeed, it's crucial to assure that modifications brought through re-sampled data, discarded cases and early convergence do not influence the models original predictions. Further studies are thus required to verify that learning accelerators do not have amplified effect on approaches predictions and validation accuracies.

## 7. Conclusion

The review of recent work trend on automatic design of CNN architectures raised some methodological options that are adopted by the majority of built approaches. Despite some attempts to use design meta-controllers based on evolutionary algorithms ([26], [27]) and Bayesian optimization ([25], [56]), reinforcement learning has shown promising empirical results and stands as the preferred strategy to train design controllers [57].

Another common conception option is the introduction of multi-branch (modular) structures as an elementary component of the entire network which restricts the search space to block/cell level. The plain network design is generally kept as a first step of proposed approaches application ([28], [42]) given that it leads to simple networks and allows to focus on the method itself before switching to more complex structures with modular design ([29], [49]). A third option used in design approaches at a lower scale is the prediction of explored architectures rewards before full training the most promising ones ([25], [29]). This training acceleration technique is implemented for performance improvement purpose and requires further attention to control possible bias impact on the models behavior.

The success of current reinforcement-learning-based approaches to design CNN architectures is widely proven especially for image classification tasks. However, it is achieved at the cost of high computational resources despite the acceleration attempts of most of recent models. Such fact is preventing individual researchers and small research entities (companies and laboratories) from fully access to this innovative technology [42]. Hence, deeper and more revolutionary optimizing methods are required to practically operate CNN automatic design. Transformation approaches based on extended network morphisms [49] are among the first attempts in this direction that achieved drastic decrease in computational cost and demonstrated generalization capacity. Additional future directions to control automatic design complexity is to develop methods for multi-task problems [58] and weights sharing [59] in order to benefit from knowledge transfer contributions.

## References

- [1] D. Kriesel, A Brief Introduction to Neural Networks, 2007.
- [2] V. Sze, Y. Chen, T. Yang, J. S. Emer, Efficient processing of deep neural networks: A tutorial and survey, *Proceedings of the IEEE* 105 (12) (2017) 2295–2329.
- [3] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, L. Fei-Fei, Imagenet large scale visual recognition challenge, *Int. J. Comput. Vision* 115 (3) (2015) 211–252.

- [4] Y. LeCun, B. E. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. E. Hubbard, L. D. Jackel, Handwritten digit recognition with a back-propagation network, in: D. S. Touretzky (Ed.), *Advances in Neural Information Processing Systems 2*, Morgan-Kaufmann, 1990, pp. 396–404.
- [5] M. Minsky, S. Papert, *Perceptrons: An Introduction to Computational Geometry*, MIT Press, Cambridge, MA, USA, 1969.
- [6] D. E. Rumelhart, G. E. Hinton, R. J. Williams, Learning representations by back-propagating errors, *Nature* 323 (1986) 533–536.
- [7] Y. Lecun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, *Proceedings of the IEEE* 86 (11) (1998) 2278–2324.
- [8] K. Jarrett, K. Kavukcuoglu, M. Ranzato, Y. LeCun, What is the best multi-stage architecture for object recognition?, in: *2009 IEEE 12th International Conference on Computer Vision*, 2009, pp. 2146–2153.
- [9] M. A. Nielsen, *Neural Networks and Deep Learning*, Determination Press, 2018.
- [10] D. Hubel, T. Wiesel, Receptive fields, binocular interaction, and functional architecture in the cat’s visual cortex, *Journal of Physiology* 160 (1962) 106–154.
- [11] A. Krizhevsky, I. Sutskever, G. E. Hinton, Imagenet classification with deep convolutional neural networks, in: *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, Curran Associates Inc., USA, 2012, pp. 1097–1105.
- [12] H. Wu, X. Gu, Max-pooling dropout for regularization of convolutional neural networks, in: *Neural Information Processing - 22nd International Conference, ICONIP 2015, Istanbul, Turkey, November 9-12, 2015, Proceedings, Part I*, 2015, pp. 46–54.
- [13] I. Goodfellow, Y. Bengio, A. Courville, *Deep Learning*, MIT Press, 2016.
- [14] M. Zeiler, R. Fergus, Stochastic pooling for regularization of deep convolutional neural networks, in: *Proceedings of the International Conference on Learning Representations (ICLR)*, 2013.

- [15] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, Dropout: A simple way to prevent neural networks from overfitting, *J. Mach. Learn. Res.* 15 (1) (2014) 1929–1958.
- [16] K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, in: *Proceedings of the International Conference on Learning Representations (ICLR)*, 2015.
- [17] M. D. Zeiler, R. Fergus, Visualizing and understanding convolutional networks, in: D. Fleet, T. Pajdla, B. Schiele, T. Tuytelaars (Eds.), *Computer Vision – ECCV 2014*, Springer International Publishing, Cham, 2014, pp. 818–833.
- [18] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, Going deeper with convolutions, in: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 1–9.
- [19] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.
- [20] C. Szegedy, S. Ioffe, V. Vanhoucke, A. A. Alemi, Inception-v4, inception-resnet and the impact of residual connections on learning, in: S. P. Singh, S. Markovitch (Eds.), *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, February 4-9, 2017, San Francisco, California, USA., AAAI Press, 2017, pp. 4278–4284.
- [21] G. Huang, Z. Liu, L. v. d. Maaten, K. Q. Weinberger, Densely connected convolutional networks, in: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 2261–2269.
- [22] B. Baker, O. Gupta, R. Raskar, N. Naik, Accelerating neural architecture search using performance prediction, in: *International Conference on Learning Representations, Workshop*, 2018.
- [23] Y. Bengio, A. Courville, P. Vincent, Representation learning: A review and new perspectives, *IEEE Trans. Pattern Anal. Mach. Intell.* 35 (8) (2013) 1798–1828.



- [24] J. S. Bergstra, R. Bardenet, Y. Bengio, B. Kégl, Algorithms for hyperparameter optimization, in: *Advances in Neural Information Processing Systems 24*, Curran Associates, Inc., 2011, pp. 2546–2554.
- [25] T. Domhan, J. T. Springenberg, F. Hutter, Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves, in: *Proceedings of the 24th International Conference on Artificial Intelligence, IJCAI’15*, AAAI Press, 2015, pp. 3460–3468.
- [26] K. O. Stanley, D. B. D’Ambrosio, J. Gauci, A hypercube-based encoding for evolving large-scale neural networks, *Artif. Life* 15 (2) (2009) 185–212.
- [27] M. Suganuma, S. Shirakawa, T. Nagao, A genetic programming approach to designing convolutional neural network architectures, in: *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO ’17*, ACM, New York, NY, USA, 2017, pp. 497–504.
- [28] B. Zoph, Q. V. Le, Neural architecture search with reinforcement learning, in: *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017.
- [29] H. Pham, M. Guan, B. Zoph, Q. Le, J. Dean, Efficient neural architecture search via parameters sharing, in: J. Dy, A. Krause (Eds.), *Proceedings of the 35th International Conference on Machine Learning*, Vol. 80, PMLR, Stockholmsmssan, Stockholm Sweden, 2018, pp. 4095–4104.
- [30] E. Brochu, T. Brochu, N. de Freitas, A bayesian interactive optimization approach to procedural animation design, in: *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, Eurographics Association, Goslar Germany, Germany, 2010, pp. 103–112.
- [31] A. Klein, S. Falkner, S. Bartels, P. Hennig, F. Hutter, Fast bayesian optimization of machine learning hyperparameters on large datasets, in: *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (AISTATS 2017)*, Vol. 54 of *Proceedings of Machine Learning Research*, PMLR, 2017, pp. 528–536.

- [32] C. E. Rasmussen, C. K. I. Williams, Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning), The MIT Press, 2005.
- [33] L. Breiman, Random forests, *Mach. Learn.* 45 (1) (2001) 5–32.
- [34] A. E. Eiben, J. E. Smith, Introduction to Evolutionary Computing, 2nd Edition, Springer Publishing Company, Incorporated, 2015.
- [35] E. Dufourq, B. A. Bassett, Eden: Evolutionary deep networks for efficient machine learning, in: 2017 Pattern Recognition Association of South Africa and Robotics and Mechatronics (PRASA-RobMech), 2017, pp. 110–115.
- [36] R. S. Sutton, A. G. Barto, Reinforcement learning - an introduction, Adaptive computation and machine learning, MIT Press, 1998.
- [37] K. Arulkumaran, M. P. Deisenroth, M. Brundage, A. A. Bharath, Deep reinforcement learning: A brief survey, *IEEE Signal Processing Magazine* 34 (6) (2017) 26–38.
- [38] C. J. C. H. Watkins, P. Dayan, Q-learning, *Machine Learning* 8 (3) (1992) 279–292.
- [39] B. Baker, O. Gupta, N. Naik, R. Raskar, Designing neural network architectures using reinforcement learning, in: Proceedings of the International Conference on Learning Representations (ICLR), 2017.
- [40] Z. Zhong, J. Yan, W. Wu, J. Shao, C.-L. Liu, Practical block-wise neural network architecture generation, in: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2018.
- [41] F. Tan, P. Yan, X. Guan, Deep reinforcement learning: From q-learning to deep q-learning, in: D. Liu, S. Xie, Y. Li, D. Zhao, E.-S. M. El-Alfy (Eds.), Neural Information Processing, Springer International Publishing, Cham, 2017, pp. 475–483.
- [42] H. Cai, T. Chen, W. Zhang, Y. Yu, J. Wang, Efficient architecture search by network transformation, in: Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI

Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018, pp. 2787–2794.

- [43] T. Chen, I. Goodfellow, J. Shlens, Net2Net: Accelerating learning via knowledge transfer, in: International Conference on Learning Representations (ICLR), 2016.
- [44] M. Schuster, K. Paliwal, Bidirectional recurrent neural networks, *Trans. Sig. Proc.* 45 (11) (1997) 2673–2681.
- [45] C. J. C. H. Watkins, Learning from delayed rewards, Ph.D. thesis, King’s College, Cambridge, UK (1989).
- [46] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, D. Hassabis, Human-level control through deep reinforcement learning, *Nature* 518 (2015) 529–533.
- [47] C. Liu, B. Zoph, M. Neumann, J. Shlens, W. Hua, L.-J. Li, L. Fei-Fei, A. Yuille, J. Huang, K. Murphy, Progressive neural architecture search, in: V. Ferrari, M. Hebert, C. Sminchisescu, Y. Weiss (Eds.), *Computer Vision – ECCV 2018*, Springer International Publishing, Cham, 2018, pp. 19–35.
- [48] F. Chollet, Xception: Deep learning with depthwise separable convolutions, in: 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017, pp. 1800–1807.
- [49] H. Cai, J. Yang, W. Zhang, S. Han, Y. Yu, Path-level network transformation for efficient architecture search, in: J. Dy, A. Krause (Eds.), *Proceedings of the 35th International Conference on Machine Learning*, Vol. 80 of *Proceedings of Machine Learning Research*, PMLR, Stockholmssan, Stockholm Sweden, 2018, pp. 678–687.
- [50] K. S. Tai, R. Socher, C. D. Manning, Improved semantic representations from tree-structured long short-term memory networks, in: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, Association for Computational Linguistics, 2015, pp. 1556–1566.

- [51] J. Dean, G. S. Corrado, R. Monga, K. Chen, M. Devin, Q. V. Le, M. Z. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang, A. Y. Ng, Large scale distributed deep networks, in: Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1, Curran Associates Inc., USA, 2012, pp. 1223–1231.
- [52] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, K. Kavukcuoglu, Hierarchical representations for efficient architecture search, in: International Conference on Learning Representations (ICLR), 2018.
- [53] T. Hinz, N. Navarro-Guerrero, S. Magg, S. Wermter, Speeding up the Hyperparameter Optimization of Deep Convolutional Neural Networks, International Journal of Computational Intelligence and Applications 17 (2).
- [54] F. H. Thomas Elsken, Jan Hendrik Metzen, Simple and efficient architecture search for convolutional neural networks, in: Proceedings of the International Conference on Learning Representations (ICLR), 2018.
- [55] T. Wei, C. Wang, C. W. Chen, Modularized morphing of neural networks, in: International Conference on Learning Representations, Workshop, 2017.
- [56] H. Mendoza, A. Klein, M. Feurer, J. T. Springenberg, F. Hutter, Towards automatically-tuned neural networks, in: F. Hutter, L. Kotthoff, J. Vanschoren (Eds.), Proceedings of the Workshop on Automatic Machine Learning, Vol. 64, PMLR, New York, New York, USA, 2016, pp. 58–65.
- [57] J. Perez-Rua, M. Baccouche, S. Pateux, Efficient progressive neural architecture search, in: British Machine Vision Conference 2018, BMVC 2018, Northumbria University, Newcastle, UK, September 3-6, 2018, BMVA Press, 2018, p. 150.
- [58] J. Liang, E. Meyerson, R. Miikkulainen, Evolutionary architecture search for deep multitask networks, in: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '18, ACM, New York, NY, USA, 2018, pp. 466–473.
- [59] G. Bender, P.-J. Kindermans, B. Zoph, V. Vasudevan, Q. Le, Understanding and simplifying one-shot architecture search, in: J. Dy,

A. Krause (Eds.), Proceedings of the 35th International Conference on Machine Learning, Vol. 80 of Proceedings of Machine Learning Research, PMLR, Stockholmsmssan, Stockholm Sweden, 2018, pp. 550–559.