Reinforcement Learning for Neural Architecture Search: A Review

Yesmina Jaafra[a,b,c], Jean Luc Laurent[b], Aline Deruyver[a], Mohamed Saber Naceur[c],

[a] ICube Laboratory, Université de Strasbourg, 300 bd Sebastien Brant, 67412 Illkirch, France
[b] Segula Technologies, Parc d'activité de Pissaloup, 8 avenue Jean d'Alembert, 78190 Trappes, France
[c] 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 ex-tracts complex and relevant characteristics allowing the optimal convergence. Designing such architectures requires significant human expertise, substantial computation time and does not always lead to the optimal network. Reinforcement learning (RL) has been extensively used in automating CNN models design generating notable advances and interesting results in the field. This work aims at reviewing and discussing the recent progress of RL methods in Neural Architecture Search task and the current challenges that still require further consideration.

Keywords: Reinforcement Learning, Convolutional Neural Networks, Neural Architecture Search, 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 (Kriesel, 2007). Brain-inspired machine learning imitates in a simplified manner the hierarchical operating mode of biological neurons (Sze, et al., 2017). 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 complex vision tasks (Russakovsky, et al., 2015).

Despite that the origin of ANN and specifically CNN (Lecun, et al., 1990) goes back to Frank Rosenblatt work in the 1950s, neural network models were not broadly used until recently, after researchers overcame certain limits. The most notable advances are the generalization of perceptrons to many layers (Minsky & Papert, 1969), the emergence of backpropagation algorithm as an appropriate training method (Rumelhart, et al., 1986) and the availability of large training datasets and computational resources to learn millions of parameters. The CNN particularity lies in the selective connection of each hidden layer neuron to a subset of neurons in the previous layer allowing hierarchical features extraction. In classification tasks, this means that 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. Much research effort in meta-modeling tries to minimize human intervention in designing neural network architectures.

Inspired from animal behaviorist psychology, reinforcement learning (RL) is a goal-oriented optimization method driven by an impact response or sig-nal (Sutton & Barto, 2018). Properly formalized and converted into practical approaches (Khan, et al., 2012), RL algorithms have achieved major progress in many fields as games (Mnih, et al., 2015), (Silver, et al., 2016), advanced robotic manipulations (Levine, et al., 2016), (Lillicrap, et al., 2016) and real world applications including Neural Architecture Search (NAS). More particularly, the RL-based approach by (Zoph & Le, 2017) that achieved state-of-the-art results on well-known classification benchmarks (CIFAR-10 and Penn Treebank) boosted the attractiveness of NAS and triggered a series of interesting work on this topic of research.

To our knowledge, there are few reviews and surveys related to NAS task in RL settings (Elsken, et al., 2019). They aimed to be quiet generic and exhaustive without specific consideration of some methodological aspects. Differently, the main motivation of our survey is to examine the background of architecture search and focus on RL approaches that gained most of the interest these recent years. In order to achieve this goal, we describe RL state-of-the-art work and shed the light on research choices by making the link between modern hand-crafted models challenges and NAS orientations. Then we tackle the crucial issue of NAS acceleration techniques which will allow a fair access to this research field without the current prerequisite of large-scale computation availability. In this way, we intend to guide future works in automatic CNN architecture design by understanding recent architecture search progress and underlying their specific requirements and constraints.

In the remainder of this paper, we first give a general overview of the task context by defining the field of deep learning and surveying the history of modern CNN hand-crafted architectures. After brie y introducing meta-modeling, we review RL applications in CNN design task according to these dimensions: (1) action and state spaces in plain and modular search methods, then (2) training optimization and acceleration at reward processing level. Finally, we conclude the article with a discussion of future work.

2. NAS Task Background

This section provides an overview of deep CNN basics and most known hand-crafted architectures. The purpose behind it is to gain a better understanding of the techniques and challenges inherent to NAS task in RL settings that will be detailed subsequently.

2.1. Deep Convolutional Learning

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.

During training, an ANN aims at learning two types of parameters: (1) connection weights that assess to which extent a neuron result will impact the output of higher level neuron, (2) 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:

$$output = \left(\sum_i input_i * \ output_i \right) + bias$$

In order to allow the network operating non-linear transformations, an activation function is applied to the previous output as the Rectified Linear Unit (ReLU) (Lecun, et al., 1998):

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

The concept of deep learning refers to machine learning processing within multi-layer ANN (Jarrett, et al., 2009). 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 Backpropagation since parameters updates are spread from final layers to initial ones.

CNN (Lecun, et al., 1990) are among the most popular deep learning techniques and conventionally include the following types of layers. (1) 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 (Hubel and Wiesel, 1962). The feature maps issued from convolving different filters (kernels) with an input image or previous layer output produce the volume of the convolutional layer (Krizhevsky, et al., 2012). 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 (Wu & Gu, 2015). (2) 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. Max pooling (Zeiler & Fergus, 2013) is one of the most common type of pooling that consists in retaining the maximum value of a neurons cluster and detecting if a given feature has been identified without recording its ex-act location (Nielsen, 2018). (3) The Fully connected layer operates the high level reasoning (classification for image case) by combining information from all the previous layers. A softmax loss layer is then used to compute the probability distribution of the CNN final outputs.

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 are perfectly adapted to image data specificity of being correlated locally and uncorrelated in global segments. Second, shared weights allow a substantial parameter reduction without altering image processing since the convolution is applicable to the whole image. Last, grid-structured image enables pooling operations that simplify data without losing useful information (Nielsen, 2018).

Deep convolutional networks imply 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.

2.2. 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 (Russakovsky, et al., 2015).

LeNet. As mentioned previously, LeNet (Lecun, et al., 1998) 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 1. 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 was not possible to extend this architecture to more complex applications.

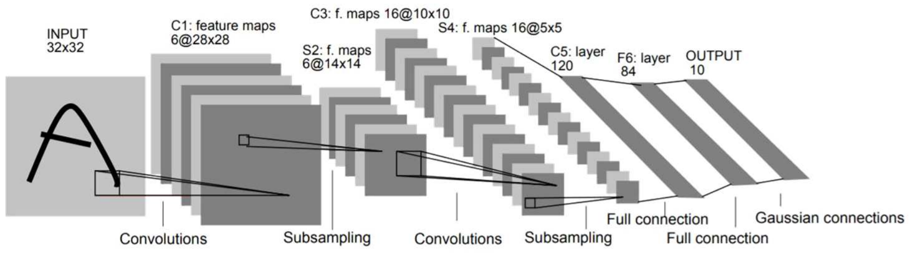


Figure 1: Architecture of LeNet-5 (Lecun, et al., 1998).

AlexNet. The model developed in (Krizhevsky, et al., 2012) is one of the most influential deep CNN that won the ILSVRC competitions in 2012. It is not much different from LeNet. 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 (Srivastava et al., 2014) 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.

VGGNet. Submitted to ILSVRC 2014, VGGNet (Simonyan & Zisserman, 2015) 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 $(3 * 3)$ were used in all layers (lower size than the $11 * 11$ 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 (Zeiler & Fergus, 2014).

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 (Szegedy, et al., 2015) innovated network design by replacing the classical strategy of alternating convolutional and pooling layers with stacked Inception Modules depicted in figure 2. 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 (Szegedy, et al., 2015).

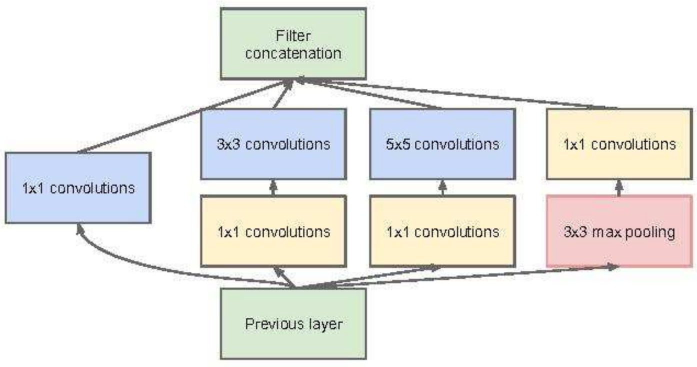


Figure 2: Inception module (Szegedy, et al., 2015).

ResNet. Deep Residual Network (He, et al., 2016) 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 3.

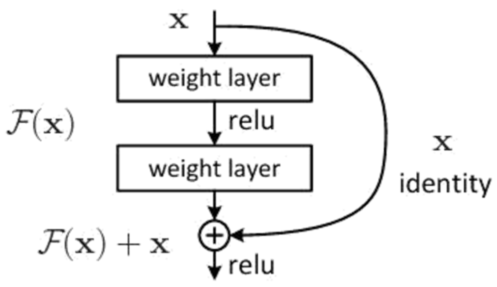

x
weight layer
relu
weight layer
$\mathcal{F}(\mathbf{x})$
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Figure 3: Residual learning: a building block (He, et al., 2016)

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. After ResNet success, which exceeded human-level accuracy, 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 (Szegedy, et al., 2017) is a new release of GoogLeNet that involved many more layers than the initial version. Inception-ResNet (Szegedy, et al., 2017) 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) (Huang, et al., 2017) where each dense block layer is connected via skip connections to all subsequent ones allowing the learning of new features.

The output of hand-crafted CNN architectures described above provides the basics of recent RL-NAS work that will be depicted in next section. It offers design alternatives and efficient solutions to the most common issues faced by deep CNN algorithms.

3. 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, RL became among the most implemented approaches (Baker, et al., 2018).

3.1. 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 previous section) 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 (Bengio, et al., 2013) 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. This process is discussed with more details in the next paragraph according to the learning concept used for optimization.

3.2. 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. The early work on NAS was dominated by Genetic and Bayesian optimization based-methods (Bergstra, et al., 2011), (Domhan, et al., 2015), (Floreano, et al., 2008), (Stanley, et al., 2009).

Evolutionary algorithms strategy of hyperparameters optimization consists in modifying a set of candidate solutions (population) on the basis of a number of rules (operators). Following an iterative procedure of mutation, crossover and selection (Eiben & Smith, 2015), 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.

Bayesian optimization is an efficient way to optimize black-box objective functions $f : X \rightarrow R$ that are slow to evaluate (Brochu, et al., 2010). 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. Hence, 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 (Klein, et al., 2017). Some of the most used probabilistic surrogate (regression) models are Gaussian processes (Rasmussen & Williams, 2005), random forests (Breiman, 2001) and tree structured Parzen estimator (Bergstra, et al., 2011).

Despite some success achieved by these strategies, they have showed structural drawbacks. Indeed, genetic algorithms take excessive computational resources and are unable to leverage the gradient information in deep learning networks (Xie, et al., 2019), while Bayesian methods are not adapted to variable-size models (Perez-Rua, et al., 2018). Lately, RL meta-controllers emerged as a competitive substitute to previous search methods. The frame of RL is an agent learning through interaction with its environment. 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 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".

The methods developed to resolve RL tasks are based on value functions, policy search or a combination of both strategies (actor-critic methods) (Arulkumaran, et al., 2017). Value function methods consist in estimating the expected reward value $\mathcal{R}$ when reaching a given state s and following a policy π:

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

A recursive form of this function is particularly used in recent Q-learning (Watkins & Dayan, 1992) models assigned to CNN architecture design (Baker, et al., 2017), (Zhong, et al., 2018):

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

where s_t is a current state, a_t is a current action, α is the learning rate, r_{t+1} is the reward earned when transitioning from time t to the next and γ 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 π^*. In this context, deep RL is achieved when deep neural networks are used to approximate one of the RL components: value function, policy or reward function (Tan, et al., 2017).

Among the active fields of designing CNN architectures through deep RL, recurrent neural networks (RNN) arise as a valuable model that handles a set of tasks such as hyperparameters prediction (Zoph & Le, 2017), (Pham, et al., 2018). In fact, a RNN operates sequentially involving hidden units to store processing history, which allows the RL agent 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.

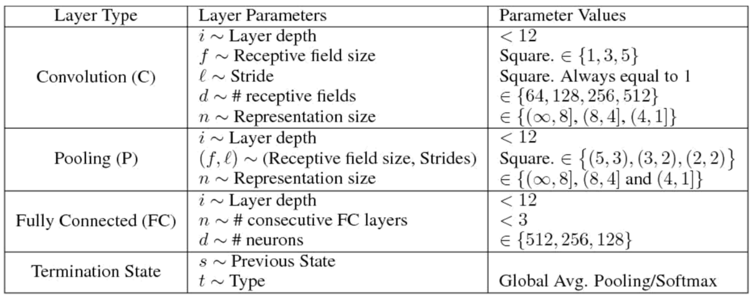
4. Reinforcement Learning for Neural Architecture Search

Various strategies have been developed to operate CNN architectures de-sign for the majority of which RL 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.

4.1. Plain Architecture Design

Some architecture search approaches focus on designing plain CNN which exclusively consists 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.

MetaQNN. This model (Baker, et al., 2017) relies on Q-learning 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 4: 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	$\in \{(\infty, 8], (8, 4]$ 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 4: State space possible parameters (Baker, et al., 2017).

Second, the agent action space is assimilated to the possible layers the agent may move to given a certain number of constraints set intentionally, to enable faster convergence. Figure 5 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.

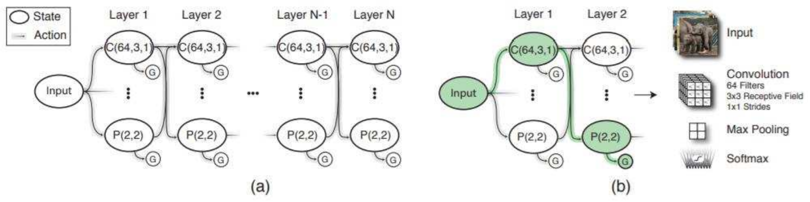


Figure 5: An illustration of the full state and action space (a) and a path that the agent has chosen (b) (Baker, et al., 2017).

NAS. Using a RL approach, (Zoph & Le, 2017) train a recurrent neural network to generate convolutional architectures. Figure 6 shows a RNN controller generating sequentially CNN parameters associated to convolutional layers. Every sequence output is predicted by a softmax classier 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 with RL 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.

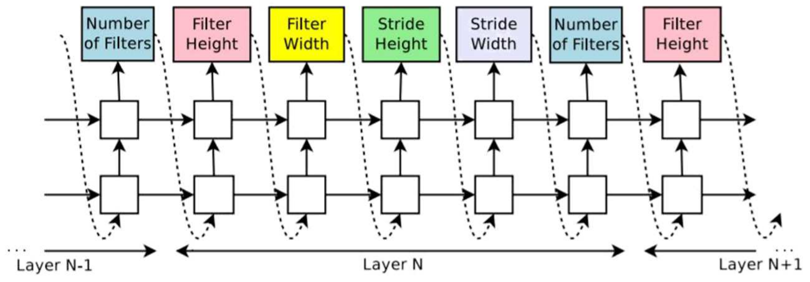


Figure 6: Illustration of the way the agent used to select hyperparameters (Zoph & Le, 2017).

EAS. In their recent work Efficient Architecture Search, (Cai, et al., 2018a) 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 RL 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 (Chen, et al., 2016) and based on the idea of building deeper student networks to reproduce the same processing of an associated teacher network. As shown in figure 7, an encoder network implemented with bidirectional recurrent neural network (Schuster & Paliwal, 1997) 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.

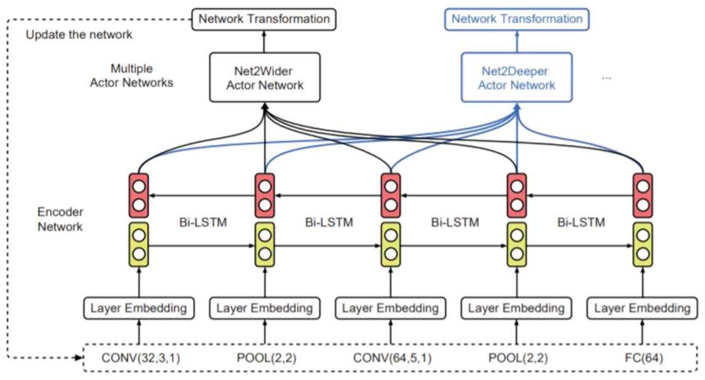


Figure 7: A meta-controller operation for network transformation (Cai, et al., 2018a).

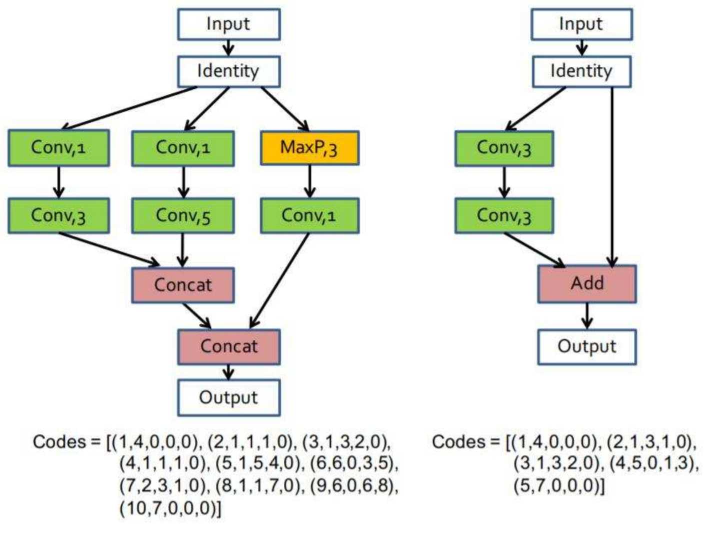
4.2. Modular Architecture Design

Most of recent work on neural architecture search is based on more complex modular (multi-branch) structures inspired by modern architectures presented in section 2. Rather than operating the tedious search over entire networks, this second set of approaches focus on finding building blocks similarly, for example, to the ones used in 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 net-works performance and gives them more transferable ability through minor adaptation.

BlockQNN. One of the first approaches implementing block-wise architecture search, BlockQNN (Zhong, et al., 2018) automatically builds convolutional networks using Q-Learning reinforcement technique (Watkins, 1989) with epsilon-greedy as exploration strategy (Mnih, et al., 2015). 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 8 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 9 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	**T**	1	1, 3, 5	**K**	0
Max Pooling	**T**	2	1, 3	**K**	0
Average Pooling	**T**	3	1, 3	**K**	0
Identity	**T**	4	0	**K**	0
Elemental Add	**T**	5	0	**K**	**K**
Concat	**T**	6	0	**K**	**K**
Terminal	**T**	7	0	0	0

Figure 8: Network structure code space (Zhong, et al., 2018).



Codes = [(1,4,0,0,0), (2,1,1,1,0), (3,1,3,2,0), (4,1,1,1,0), (5,1,5,4,0), (6,6,0,3,5), (7,2,3,1,0), (8,1,1,7,0), (9,6,0,6,8), (10,7,0,0,0)]

Codes = [(1,4,0,0,0), (2,1,3,1,0), (3,1,3,2,0), (4,5,0,1,3), (5,7,0,0,0)]

Figure 9: Representative block exemplars with their network structure codes (Zhong, et al., 2018).

PNAS. Progressive neural architecture search (Liu et al., 2018a) 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 is 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. 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 approach (Zoph & Le, 2017), 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 10.

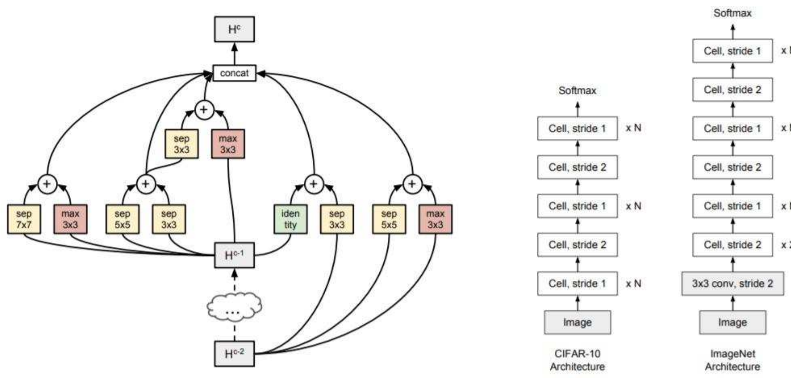


Figure 10: The best selected cell architecture of PNAS (Liu, et al., 2018a).

ENAS. Efficient neural architecture search (Pham, et al., 2018) comes in the continuity of previous work NAS (Zoph & Le, 2017) and PNAS (Liu, et al., 2018a). It explores a cell-based search space through a controller RNN trained with RL. 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 wise-separable convolutions of filter sizes $3 * 3$ and $5 * 5$ (Chollet, 2017), 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 11 illustrates the design of a 4-node cell. At the end, the entire CNN is built by stacking N times convolutional cells.

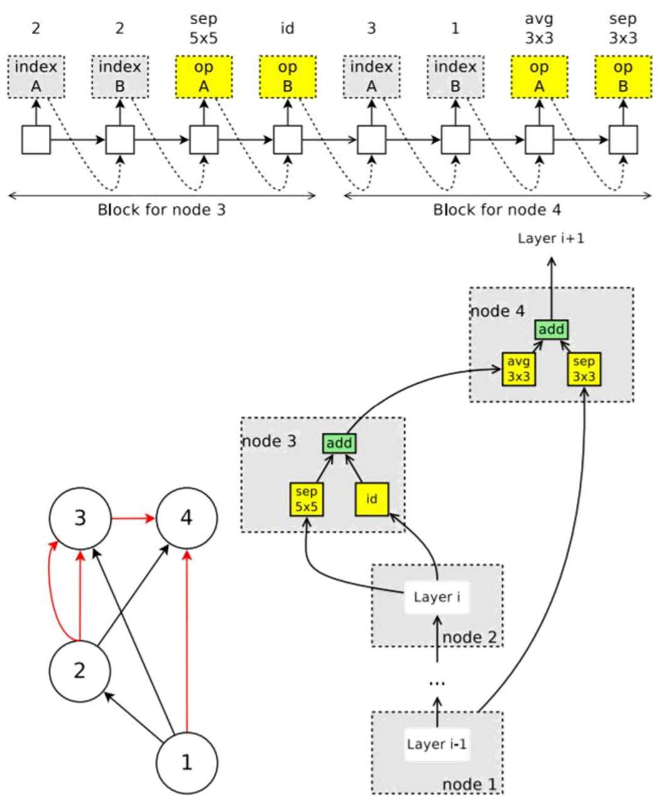


Figure 11: Illustration of 4-nodes cell (Pham, et al., 2018).

Another contribution of ENAS consists in sampling mini-batches from validation dataset to train designed models. The models with the best ac-curacy 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.

EAS With Path Level Transformation. A developed version of EAS (Cai, et al., 2018a) which adopts network transformation for efficient CNN architecture search is presented in (Cai, et al., 2018b). 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 (Cai, et al., 2018a) where the RL 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, (Tai, et al., 2015) 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 12 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.

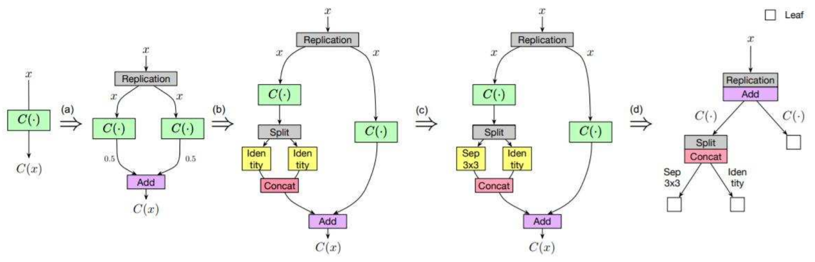


Figure 12: Path-level transformation: from a single layer to a tree-structured motif (Cai, et al., 2018b).

4.3. Architecture search accelerators

RL 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 (Zhong, et al., 2018) 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 RL convergence (Sutton & Barto, 1998). 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 (Zhong, et al., 2018) 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 (Dean et al., 2012) 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. (Zoph & Le, 2017) uses the same parameter server scheme with replication of controllers in order to train various architectures in parallel.

As seen previously, RL 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 (Domhan, et al., 2015) and surrogate models using RNN predictor (Liu, et al., 2018b) 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) (Pham, et al., 2018) and down-sampled data (like images with lower resolution) (Hinz, et al., 2018).

In an original attempt to address this issue, (Ying, et al., 2019) introduce NAS-Bench-101, the first architecture-dataset for NAS. They repeatedly trained and evaluated thousands of CNN cell-based architectures on CIFAR-10 taking years of tensor processing unit (TPU) of computation time. Subsequently, they obtained a large table mapping almost 423 thousand architecture to their accuracies and other metrics according to specific graph connectivity. It will allow running NAS experiments in record times by querying this table instead of fully processing the costly evaluation procedure of explored architectures. Despite NAS-Bench-101 is new and still requires deeper state-of-the-art assessment, it represents an interesting alternative to optimize NAS work.

Network transformation is one of the more recent techniques assigned to accelerate neural architecture search (Cai, et al., 2018b), (Elsken, et al., 2018). It consists in training explored architectures reusing previously trained or existing networks. This modeling feature allows to address a limitation of RL approaches where training is performed with a random initialization of weights. Thus, extending network morphisms (Wei, et al., 2017) 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 is 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.

5. Conclusion

The review of recent work on RL approaches for CNN automatic de-sign raised some common methodological trends, mainly 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 (Zoph & Le, 2017), (Cai, et al., 2018a) given that it leads to simple networks and allows to focus on the method itself before switching to more complex structures with modular design (Pham, et al., 2018), (Cai, et al., 2018b). We notice as well the use of prediction techniques to compute explored architectures rewards before full training the most promising ones (Domhan, et al., 2015), (Pham, et al., 2018). This training acceleration procedure is implemented for performance improvement purpose and requires further consideration to control possible resulting bias in generated models.

The success of current RL approaches in NAS field 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 (Cai, et al., 2018a). Hence, deeper and more revolutionary optimizing methods are required to practically operate CNN automatic de-sign. Transformation approaches based on extended network morphisms (Cai, et al., 2018b) 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 (Liang, et al., 2018) and weights sharing (Bender, et al., 2018) in order to benefit from knowledge transfer contributions.

References

Arulkumaran, K., Deisenroth, M. P., Brundage, M. & Bharath, A. A., 2017. Deep Reinforcement Learning: A Brief Survey. *IEEE Signal Processing Magazine,* 11, Volume 34, pp. 26-38.

Baker, B., Gupta, O., Naik, N. & Raskar, R., 2017. *Designing Neural Network Architectures using Reinforcement Learning.* s.l., s.n.

Baker, B., Gupta, O., Raskar, R. & Naik, N., 2018. *Accelerating Neural Architecture Search using Performance Prediction.* s.l., s.n.

Bender, G. et al., 2018. *Understanding and Simplifying One-Shot Architecture Search.* Stockholmsmässan, Stockholm Sweden, PMLR, pp. 550-559.

Bengio, Y., Courville, A. & Vincent, P., 2013. Representation Learning: A Review and New Perspectives. *IEEE Trans. Pattern Anal. Mach. Intell.,* 8, Volume 35, pp. 1798-1828.

Bergstra, J. S., Bardenet, R., Bengio, Y. & Kégl, B., 2011. Algorithms for Hyper-Parameter Optimization. Dans: *Advances in Neural Information Processing Systems 24.* s.l.:Curran Associates, Inc., pp. 2546-2554.

Breiman, L., 2001. Random Forests. *Mach. Learn.,* Volume 45, pp. 5-32.

Brochu, E., Brochu, T. & Freitas, N., 2010. *A Bayesian Interactive Optimization Approach to Procedural Animation Design.* Goslar, Eurographics Association, pp. 103-112.

Cai, H. et al., 2018a. *Efficient Architecture Search by Network Transformation.* s.l., s.n., pp. 2787-2794.

Cai, H. et al., 2018b. *Path-Level Network Transformation for Efficient Architecture Search.* Stockholmsmässan, Stockholm Sweden, PMLR, pp. 678-687.

Chen, T., Goodfellow, I. & Shlens, J., 2016. *Net2Net: Accelerating Learning via Knowledge Transfer.* s.l., s.n.

Chollet, F., 2017. *Xception: Deep Learning with Depthwise Separable Convolutions.* s.l., s.n., pp. 1800-1807.

Domhan, T., Springenberg, J. T. & Hutter, F., 2015. *Speeding Up Automatic Hyperparameter Optimization of Deep Neural Networks by Extrapolation of Learning Curves.* Buenos, AAAI Press, pp. 3460-3468.

Eiben, A. E. & Smith, J. E., 2015. *Introduction to Evolutionary Computing.* 2nd éd. s.l.:Springer Publishing Company, Incorporated.

Elsken, T., Metzen, J. H. & Hutter, F., 2018. *Simple and efficient architecture search for Convolutional Neural Networks.* s.l., s.n.

Elsken, T., Metzen, J. H. & Hutter, F., 2019. Neural Architecture Search: A Survey. *Journal of Machine Learning Research,* Volume 20, pp. 1-21.

Floreano, D., Dürr, P. & Mattiussi, C., 2008. *Neuroevolution: from architectures to learning.* s.l.:s.n.

He, K., Zhang, X., Ren, S. & Sun, J., 2016. *Deep Residual Learning for Image Recognition.* s.l., s.n., pp. 770-778.

Hinz, T., Navarro-Guerrero, N., Magg, S. & Wermter, S., 2018. Speeding up the Hyperparameter Optimization of Deep Convolutional Neural Networks. *International Journal of Computational Intelligence and Applications,* 6.Volume 17.

Huang, G., Liu, Z., Maaten, L. & Weinberger, K. Q., 2017. *Densely Connected Convolutional Networks.* s.l., s.n., pp. 2261-2269.
Jarrett, K., Kavukcuoglu, K., Ranzato, M. & LeCun, Y., 2009. *What is the best multi-stage architecture for object recognition?.* s.l., s.n., pp. 2146-2153.
Khan, S. G. et al., 2012. Reinforcement Learning and Optimal Adaptive Control: An Overview and Implementation examples. *Annual Reviews in Control,* 4, Volume 36(1), pp. 42-59.
Klein, A. et al., 2017. *Fast Bayesian Optimization of Machine Learning Hyperparameters on Large Datasets.* s.l., PMLR, pp. 528-536.
Kriesel, D., 2007. *A Brief Introduction to Neural Networks.* s.l.:s.n.
Krizhevsky, A., Sutskever, I. & Hinton, G. E., 2012. *ImageNet Classification with Deep Convolutional Neural Networks.* USA, Curran Associates Inc., pp. 1097-1105.
Lecun, Y. et al., 1990. Handwritten Digit Recognition with a Back-Propagation Network. Dans: D. S. Touretzky, éd. *Advances in Neural Information Processing Systems 2.* s.l.:Morgan-Kaufmann, pp. 396-404.
Lecun, Y., Bottou, L., Bengio, Y. & Haffner, P., 1998. Gradient-based learning applied to document recognition. *Proceedings of the IEEE,* Volume 86, pp. 2278-2324.
Levine, S., Finn, C., Darrell, T. & Abbeel, P., 2016. End-to-end Training of Deep Visuomotor Policies. *J. Mach. Learn. Res.,* 1, Volume 17, pp. 1334-1373.
Liang, J., Meyerson, E. & Miikkulainen, R., 2018. *Evolutionary Architecture Search for Deep Multitask Networks.* New York, NY, USA, ACM, pp. 466-473.
Lillicrap, T. P. et al., 2016. Continuous control with deep reinforcement learning. *ICLR.*
Liu, C. et al., 2018a. *Progressive Neural Architecture Search.* Cham, Springer International Publishing, pp. 19-35.
Liu, H. et al., 2018b. *Hierarchical Representations for Efficient Architecture Search.* s.l., s.n.
Minsky, M. & Papert, S., 1969. *Perceptrons: An Introduction to Computational Geometry.* Cambridge(MA): MIT Press.
Mnih, V. et al., 2015. Human-level control through deep reinforcement learning. *Nature,* Volume 518, pp. 529-533.
Nielsen, M. A., 2018. *Neural Networks and Deep Learning.* s.l.:Determination Press.
Perez-Rua, J.-M., Baccouche, M. & Pateux, S., 2018. *Efficient Progressive Neural Architecture Search.* s.l., BMVA Press, p. 150.
Pham, H. et al., 2018. *Efficient Neural Architecture Search via Parameters Sharing.* Stockholmsmässan, Stockholm Sweden, PMLR, pp. 4095-4104.
Rasmussen, C. E. & Williams, C. K. I., 2005. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning).* s.l.:The MIT Press.
Rumelhart, D. E., Hinton, G. E. & Williams, R. J., 1986. Learning representations by back-propagating errors. *Nature,* 10, Volume 323, pp. 533-536.
Russakovsky, O. et al., 2015. ImageNet Large Scale Visual Recognition Challenge. *Int. J. Comput. Vision,* 12, Volume 115, pp. 211-252.
Schuster, M. & Paliwal, K. K., 1997. Bidirectional Recurrent Neural Networks. *Trans. Sig. Proc.,* Volume 45, pp. 2673-2681.
Silver, D. et al., 2016. Mastering the Game of Go with Deep Neural Networks and Tree Search. *Nature,* 1, Volume 529, pp. 484-489.

Simonyan, K. & Zisserman, A., 2015. *Very Deep Convolutional Networks for Large-Scale Image Recognition.* s.l., s.n.
Stanley, K. O., D'Ambrosio, D. B. & Gauci, J., 2009. A Hypercube-based Encoding for Evolving Large-scale Neural Networks. *Artif. Life,* 4, Volume 15, pp. 185-212.
Sutton, R. S. & Barto, A. G., 1998. *Reinforcement learning - an introduction.* s.l.:MIT Press.
Sutton, R. S. & Barto, A. G., 2018 . *Reinforcement Learning: An Introduction.* Second éd. s.l.:The MIT Press.
Szegedy, C., Ioffe, S., Vanhoucke, V. & Alemi, A. A., 2017. *Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning.* s.l., AAAI Press, pp. 4278-4284.
Szegedy, C. et al., 2015. *Going deeper with convolutions.* s.l., s.n., pp. 1-9.
Sze, V., Chen, Y., Yang, T. & Emer, J. S., 2017. Efficient Processing of Deep Neural Networks: A Tutorial and Survey. *Proceedings of the IEEE,* 12, Volume 105, pp. 2295-2329.
Tai, K. S., Socher, R. & Manning, C. D., 2015. *Improved Semantic Representations From Tree-Structured Long Short-Term Memory Networks.* Beijing, Association for Computational Linguistics, pp. 1556-1566.
Tan, F., Yan, P. & Guan, X., 2017. *Deep Reinforcement Learning: From Q-Learning to Deep Q-Learning.* Cham, Springer International Publishing, pp. 475-483.
Watkins, C. J. C. H. & Dayan, P., 1992. Q-learning. *Machine Learning,* Volume 8, pp. 279-292.
Wei, T., Wang, C. & Chen, C. W., 2017. *Modularized Morphing of Neural Networks.* s.l., s.n.
Wu, H. & Gu, X., 2015. *Max-Pooling Dropout for Regularization of Convolutional Neural Networks.* s.l., s.n., pp. 46-54.
Xie, S., Zheng, H., Liu, C. & Lin, L., 2019. *SNAS: stochastic neural architecture search.* s.l., s.n.
Ying, C. et al., 2019. Towards Reproducible Neural Architecture Search. *ICML 2018 - Reproducibility in Machine Learning Workshop,* Volume abs/1902.09635.
Zeiler, M. D. & Fergus, R., 2014. *Visualizing and Understanding Convolutional Networks.* Cham, Springer International Publishing, pp. 818-833.
Zeiler, M. & Fergus, R., 2013. *Stochastic pooling for regularization of deep convolutional neural networks.* s.l., s.n.
Zhong, Z. et al., 2018. *Practical Block-Wise Neural Network Architecture Generation.* s.l., s.n.
Zoph, B. & Le, Q. V., 2017. *Neural Architecture Search with Reinforcement Learning.* s.l., s.n.