Apprentissage de l'architecture d'un réseau de neurones profond sous contraintes de budget

Tom Véniat[*] et Ludovic Denoyer[*,†]

[*]Sorbonne Université, LIP6, F-75005, Paris, France
[†]Criteo Research
{tom.veniat, ludovic.denoyer}@lip6.fr

Résumé

Nous proposons d'aborder le problème de la recherche d'architectures de réseaux de neurones permettant d'être efficace tant en performance prédictive qu'en coût d'inférence. Notre méthode peut par exemple trouver un modèle capable de faire des prédictions en moins de 100ms ou en utilisant moins de 50Mb de mémoire. Notre contribution, les Budgeted Super Networks (BSN), est entrainée en utilisant une fonction objectif intégrant un coût maximal pouvant être de différentes natures. Nous l'évaluons sur des problèmes de vision en démontrant sa capacité à s'adapter à trois types de budgets: le nombre d'opérations, la consommation mémoire et la parallélisation de l'architecture obtenue. Notre approche trouve notamment des architectures plus performantes que le ResNet et les CNFs tout en ayant un coût d'inférence inférieur.

Mots-clés

Apprentissage d'architectures de réseaux de neurones, Optimisation sous contrainte de budget, Apprentissage par renforcement, Apprentissage profond.

Abstract

We propose to focus on the problem of discovering neural network architectures efficient in terms of both prediction quality and cost. For instance, our approach is able to learn a neural network able to predict well in less than 100 milliseconds or using less than 50 Mb memory. Our contribution, the Budgeted Super Networks (BSN), are trained using gradient descent techniques applied on a budgeted learning objective function containing a maximum authorized cost. We present a set of experiments on computer vision problems and analyze the ability of our technique to deal with three different costs: the computation cost, the memory consumption cost, and also a distributed computation cost. We particularly show that our model can discover neural network architectures that have a better accuracy than the ResNet and CNF architectures on CIFAR-10 and CIFAR-100, at a lower cost.

Keywords

Neural Architecture Search, Budget Constrained Optimization, Reinforcement Learning, Deep Learning.

1 Introduction

In the Deep Learning community, finding the best Neural Network architecture for a given task is a key problem that is mainly addressed *by hand* or using validation techniques. For instance, in computer vision, this has lead to particularly well-known models like GoogleNet [27] or ResNet [10]. More recently, there is a surge of interest in developing techniques able to automatically discover efficient neural network architectures. Different algorithms have been proposed including evolutionary methods [26, 19, 22] or reinforcement learning-based approaches [31]. But in all cases, this selection is usually based solely on a final predictive performance of the model such as the accuracy.

When facing real-world problems, this predictive performance is not the only measure that matters. Indeed, learning a very good predictive model with the help of a cluster of GPUs might lead to a neural network that can be incompatible with low-resource mobile devices. Another example concerns distributed models in which one part of the computation is made *in the cloud* and the other part is made *on the device*. In such situations, an efficient architecture would have to predict accurately while minimizing the amount of exchanged messages between the cloud and the device. One important research direction is thus to propose models that can learn to take into account the inference cost in addition to the quality of the prediction.

We formulate this issue as a problem of automatically learning a neural network architecture under budgeted constraints. To tackle this problem, we propose a *budgeted learning approach* that integrates a maximum cost directly in the learning objective function. The main originality of our approach with respect to state-of-the-art is the fact that it can be used with any type of costs, existing methods being usually specific to particular constraints like inference speed or memory consumption – see Section 5 for a complete review of state-of-the-art. In our case, we investigate the ability of our method to deal with three different costs: (i) the *computation cost* reflecting the inference speed of the resulting model, (ii) *the memory consumption cost* that measures the final size of the model, and the (iii) *distributed computation cost* that measures the inference

speed when computations are distributed over multiple machines or processors.

Our model called *Budgeted Super Network* (BSN) is based on the following principles: (i) the user provides a (big) *Super Network* (see Section 2) that defines a very large set of possible final network architectures as well as a maximum authorized cost. (ii) Since finding the best architecture that satisfies the cost constraint is an intractable combinatorial problem (Section 3.1), we relax this optimization problem and propose a stochastic model (called *Stochastic Super Networks* – Section 3.2) that can be optimized using policy gradient-inspired methods (Section 3.3). We show that the optimal solution of this stochastic problem corresponds to the optimal constrained network architecture (Proposition 1) validating our approach. We evaluate this model on various computer vision tasks in section 4). The related work is presented in Section 5.

2 Super Networks

We consider the classical supervised learning problem defined by an input space $\mathcal{X}$ and an output space $\mathcal{Y}$. In the following, they correspond to multi-dimensional real-valued spaces. The training set is denoted as $\mathcal{D} = \{(x^1, y^1), ..., (x^\ell, y^\ell)\}$ where $x^i \in \mathcal{X}$, $y^i \in \mathcal{Y}$ and ℓ is the number samples. At last, we consider a model $f : \mathcal{X} \to \mathcal{Y}$ that predicts an output given a particular input.

We first present a family of models called *Super Networks* (S-network), our contribution being a (stochastic) extension of this family. The principle of Super Networks is not new and similar ideas have been already proposed in the literature under different names, e.g Deep Sequential Neural Networks [5], Neural Fabrics [24], or even PathNet [8].

A S-network is composed of a set of layers connected together forming a directed acyclic graph (DAG). In this DAG, each edge is a (small) neural network. A Super Network corresponds to a particular combination of these edges, forming a computation graph. Examples of S-networks are given in Figure 1. More formally, let us denote $l_1, ..., l_N$ a set of layers, N being the number of layers, such that each layer l_i is associated with a particular representation space $\mathcal{X}_i$ which is a multi-dimensional real-valued space. l_1 is the *input layer* while l_N is the *output layer*. We also consider a set of (differentiable) functions $f_{i,j}$ associated to each possible pair of layers such that $f_{i,j} : \mathcal{X}_i \to \mathcal{X}_j$. Each function $f_{i,j}$ will be referred as a *module* in the following: it takes data from $\mathcal{X}_i$ as inputs and transforms these data to $\mathcal{X}_j$. Note that each $f_{i,j}$ can make complex disk/memory/network operations which will have consequences on the inference speed of the S-network. Each $f_{i,j}$ module is associated with parameters in θ, θ being implicit in the notation for sake of clarity.

On top of this structure, a particular architecture $E = \{e_{i,j}\}_{(i,j)\in[1;N]^2}$ is a binary adjacency matrix over the N layers such that E defines a DAG with a single source node l_1 and a single sink node l_N. Different matrices E will thus correspond to different super network architectures. A S-network will be denoted (E, θ) in the following, θ being the parameters of the different *modules*, and E being the architecture of the super network.

Predicting with S-networks: The computation of the output $f(x, E, \theta)$ given an input x and a super network (E, θ) is made through a classic forward algorithm, the main idea being that the output of modules $f_{i,j}$ and $f_{k,j}$ leading to the same layer l_j will be added in order to compute the value of l_j. Let us denote $l_i(x, E, \theta)$ the value of layer l_i for input x, the computation is recursively defined as:

$$\begin{aligned} \text{Input: } & l_1(x, E, \theta) = x \\ \text{Layer Computation: } & l_i(x, E, \theta) = \sum_k e_{k,i} f_{k,i}(l_k(x, E, \theta)) \end{aligned} \quad (1)$$

Note that learning the parameters θ can be made using classical back-propagation and gradient-descent techniques.

3 Learning Cost-constrained architectures

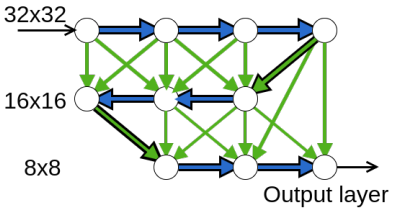
Our main idea is the following: we now consider that the structure E of the S-network (E, θ) describes not a single neural network architecture but a set of possible architectures. Indeed, each sub-graph of E (subset of edges) corresponds to a S-network. A sub-graph of edges will be denoted $H \odot E$ where H corresponds to a binary matrix used as a mask to select edges in E, $\odot$ denoting the Hadamard product between H and E. Our objective will thus be to identify the best matrix H such that the corresponding S-network $(H \odot E, \theta)$ will be a network efficient both in terms of predictive quality and in terms of computation/memory/.. cost.

3.1 Budgeted Architectures Learning

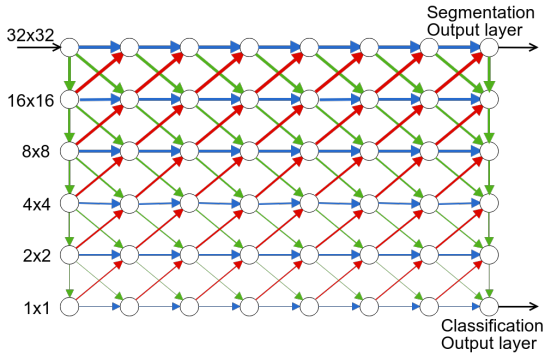
Let us consider H a binary matrix of size $N \times N$. Let us denote $C(H \odot E) \in \mathbb{R}^+$ the computation cost associated to the computation of the S-network $(H \odot E, \theta)$ which can be of different nature (see Section 4). Let us also define $\mathbf{C}$ the maximum computation cost the user would allow. For instance, when solving the problem of *learning a model with a computation time lower than 200 ms* then $\mathbf{C}$ is equal to $200ms$. We aim at solving the following soft constrained budgeted learning problem:

$$\begin{aligned} H^*, \theta^* = \arg\min_{H,\theta} \frac{1}{\ell} \sum_i \Delta(f(x^i, H \odot E, \theta), y^i) \\ + \lambda \max(0, C(H \odot E) - \mathbf{C}) \end{aligned} \quad (2)$$

where λ corresponds to the importance of the cost penalty. Note that the computational cost is specific to the particular infrastructure on which the model is ran. For instance, if $\mathbf{C}$ is the cost in milliseconds, the value of $C(H \odot E)$ will not be the same depending on the device on which the model is used. Finding a solution to this learning problem



(a) **ResNet Fabric**: The ResNet Fabric is a super network that includes the ResNet model as a particular sub-graph. Each row corresponds to a particular size and number of feature maps. Each edge represents a simple *building block* (as described in[10]) i.e two stacked convolution layers + a shortcut connection. We use projection shortcuts (with 1x1 convolutions) for all connections going across different feature map sizes (green edges). Note that here, the subgraph corresponding to the bold edges is a ResNet-20. By increasing the width of the ResNet Fabric, we can include different variants of ResNets (from ResNet-20 with width 3 up to Resnet-110 with a width of 18).



(b) **Convolutional Neural Fabric** [24]: The CNF is a generic architecture that can be used for both **image classification** and **image segmentation**. The layers of the CNF super network are organized in a $W \times H$ matrix. Each row corresponds to a particular resolution of feature maps. The number of features map is constant across the whole network. Each edge represents a convolution layer. The color of an edge represents the difference between input and output maps resolutions. Blue edges keep the same resolution, green edges decrease the resolution (stride > 1) and red edges increase the resolution (upsampling). Feature maps are aggregated (by addition) at each node before being sent to following layers.

Figure 1: This figure illustrates two Super Networks on top of which cost-constrained architectures will be discovered. The ResNet Fabric is a generalization of ResNets [10], while CNF has been proposed in [24]. Our objective is to discover architectures that are efficient both in prediction quality and cost, by sampling edges over these super networks.

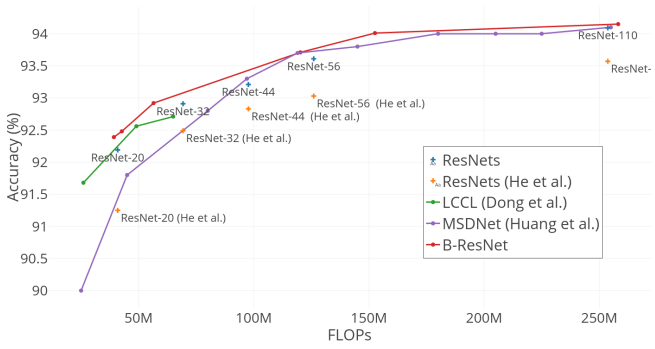


Figure 2: Accuracy/Time trade-off using B-ResNet on CIFAR-10.

is not trivial since it involves the computation of all possible architectures which is prohibitive ($\mathcal{O}(2^N)$ in the worst case). We explain in the next section how this problem can be solved using *Stochastic Super Networks*.

3.2 Stochastic Super Networks

Now, given a particular architecture E, we consider the following stochastic model – called **Stochastic Super Network** (SS-network) – that computes a prediction in two steps:

1. First a sub-graph H is sampled based on a distribution with parameters Γ. This operation will be denoted $H \sim \Gamma$
2. The final prediction is made using this sub-graph i.e by computing $f(x, H \odot E, \theta)$.

A SS-network is thus defined by a triplet (E, Γ, θ), Γ and θ being both learned on a training set.
We can rewrite the budgeted learning objective of Equation 2 as:

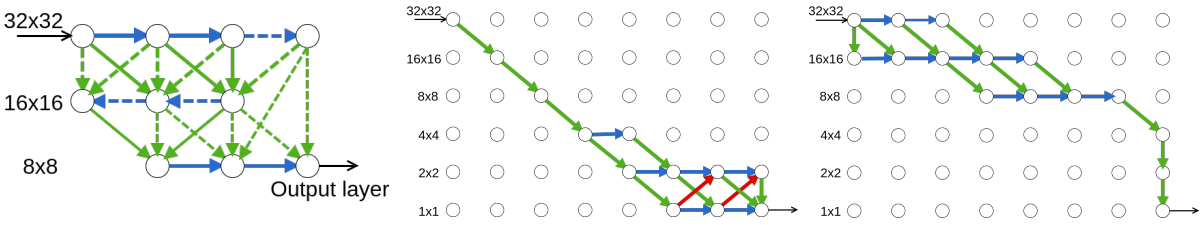
$$\Gamma^*, \theta^* = \arg\min_{\Gamma,\theta} \frac{1}{\ell} \sum_i \mathbb{E}_{H\sim\Gamma} \left[\Delta(f(x^i, H \odot E, \theta), y^i) + \lambda \max(0, C(H \odot E) - \mathbf{C})\right] \quad (3)$$

Proposition 1 *When the solution of Equation 3 is reached, then the models sampled following* (Γ^*) *and using parameters* θ^* *are optimal solution of the problem of Equation 2.*

Said otherwise, solving the stochastic problem will provide a model that has a good predictive performance under the given cost constraint.

Edge Sampling: In order to avoid inconsistent architectures where the input layer and the output one are not connected, we sample the edges following the given procedure: For each layer l_i visited following the DAG order of E (from the first layer to the last one) and for all $k < i$: If l_k is connected to the input layer l_1 based on the previously sampled edges, then $h_{k,i}$ is sampled following a Bernoulli distribution with probability[1] $\gamma_{k,i}$. In the other cases then $h_{k,i} = 0$.

[1]Note that $\gamma_{k,i}$ is obtained by applying a *logistic* function over a continuous parameter value, but this is made implicit in our notations.



(a) B-ResNet (b) B-CNF & computation cost (c) B-CNF & memory consumption cost

Figure 3: Discovered architectures: **(Left)** is a low *computation cost* B-ResNet where dashed edges correspond to connections in which the two convolution layers have been removed (only shortcuts connections are kept). **(Center)** is a low *computation cost* B-CNF where high-resolution operations have been removed. **(Right)** is a low memory consumption cost B-CNF: the algorithm has mostly kept all high resolution convolutions since they allow fine-grained feature maps and have the same number of parameters than lower-resolution convolutions. It is interesting to note that our algorithm, constrained with two different costs, automatically learned two different efficient architectures.

3.3 Learning Algorithm

We consider a generic case where the cost-function $C(H \odot E)$ is unknown and can be observed at the end of the computation of the model over an input x. Note that this case also includes stochastic costs where the cost is a random variable, caused by some network latency during distributed computation for example. We now describe the case where C is deterministic, its extension to stochastic values being straightforward.

Let us denote $\mathcal{D}(x, y, \theta, E, H)$ the quality of the model $(H \odot E, \theta)$ on a given training pair (x, y):

$$\mathcal{D}(x, y, \theta, E, H) = \Delta(f(x, H \odot E, \theta), y) + \lambda \max(0, C(H \odot E) - \mathbf{C}) \quad (4)$$

We propose to use a policy gradient inspired algorithm as in [5, 1] for learning. In that case, let us denote $\mathcal{L}(x, y, E, \Gamma, \theta)$ the expectation of $\mathcal{D}$ over the possible sampled matrices H:

$$\mathcal{L}(x, y, E, \Gamma, \theta) = \mathbb{E}_{H \sim \Gamma} \mathcal{D}(x, y, \theta, E, H) \quad (5)$$

The gradient of $\mathcal{L}$ can be written as:

$$\begin{aligned} &\nabla_{\theta,\Gamma} \mathcal{L}(x, y, E, \Gamma, \theta) \\ &= \sum_H P(H|\Gamma) \left[(\nabla_{\theta,\Gamma} \log P(H|\Gamma)) \mathcal{D}(x, y, \theta, E, H) \right] \\ &\quad + \sum_H P(H|\Gamma) \left[\nabla_{\theta,\Gamma} \Delta(f(x, H \odot E, \theta), y) \right] \end{aligned} \quad (6)$$

The first term corresponds to the gradient over the log-probability of the sampled structure H while the second term is the gradient of the prediction loss given the sampled structure $H \odot E$.

Learning can be made using back-propagation and stochastic-gradient descent algorithms as it is made in Deep Reinforcement Learning models.

4 Experiments

4.1 Implementation

We study two particular super network architectures presented in figure 1.

Image classification has been tested on CIFAR-10 and CIFAR-100 [15] while image segmentation has been performed on the Part Label dataset [14].

For these two architectures denoted B-CNF and B-ResNet, we consider three different costs functions: the first one is the (i) *computation cost* computed as the number of operations [2] made by the resulting network as used in [7] or [13]. Note that this cost is highly correlated with the execution time[3]. The second one is the *memory consumption cost*, measured as the number of parameters of the resulting models. At last, the third cost (iii) is the *distributed computation cost* which is detailed in Section 4.3 and corresponds to the ability of a particular model to be efficiently computed over distributed architectures.

4.2 Experimental Protocol and Baselines

We have learned our model with various values of $\mathbf{C}$. For the image classification problem, since we directly compare to ResNet, we have chosen values of $\mathbf{C}$ that corresponds to the cost of the ResNet-20/32/44/56/110 architectures. This allows us to compare the performance of our method at the same cost level than the ResNet variants. When dealing with the B-CNF model, we have chosen $\mathbf{C}$ to be a fraction of the cost of the (full) CNF model, allowing us to measure the quality of our model to discover low-cost architectures.

For each experiment, multiple versions of the model are evaluated over the validation set during leaning. Since our evaluation now involves both a cost and an accuracy, we select the best models using the pareto front on the

[2] The number of Mult-Add operations required to evaluate a network.

[3] Expressing constraint directly in *milliseconds* has been also investigated, with results similar to the ones obtain using the computation cost, and are not presented here.

cost/accuracy curve on the validation set. The reported performance are then obtained by evaluating these selected models over the test set.

The learning algorithm is a classical gradient-descent optimization technique, SGD with a momentum of 0.9. For all models and all cost functions, we select the λ hyper-parameter based on the order of magnitude m of the maximum authorized cost $\mathbf{C}$. λ is determined using cross-validation on values logarithmically spaced between 10^{m-1} and 10^{m+1}.

For each experiment, we give the performance of both reference models (ResNet [10] and CNF [24]), and of related existing models i.e Low Cost Collaborative Layer (LCCL)[7] and MSDNet [13] (under the anytime classification settings). Note that the baselines methods have been designed to reduce exclusively the *computation cost*, while our technique is able to deal with any type of cost. We provide the performance of our budgeted version of ResNet (B-ResNet) and of our budgeted version of CNF (B-CNF). Note that, for a fair comparison, we present the published results of ResNet and CNF, but also the ones that we have obtained by training these models by ourselves, **our results being of better quality** than the previously published performance.

Experimental results.

Reducing the computation cost: We first consider the *computation cost*. Figure 2 and Table 1 show the performance of different models over CIFAR-10. Each point corresponds to a model evaluated both in term of accuracy and *computation cost*. When considering the B-ResNet model, and by fixing the value of $\mathbf{C}$ to the computation costs of the different ResNet architectures, we obtain budgeted models that have approximatively the same costs than the ResNets, but with a higher accuracy. For example, ResNet-20 obtains an accuracy of 92.19% at a cost of 40.9×10^6 flop, while B-ResNet is able to discover an architecture with 92.39% accuracy at a slightly lower cost (39.25×10^6 flop). Moreover, the B-ResNet model also outperforms existing approaches like MSDNet or LCCL, particularly when the *computation cost* is low i.e for architectures that can be computed at a high speed. When comparing CNF to B-CNF, one can see that our approach is able to considerably reduce the computation cost while keeping a high accuracy. For example, one of our learned models obtained an accuracy of 93.14% with a cost of 103×10^6 flop while CNF has an accuracy of 92.54% for a cost of 406×10^6 flop. Note that the same observations can be drawn for CIFAR-100 (Table 2).

Figure 3a and 3b illustrate two architectures discovered by B-ResNet and B-CNF with a low computation cost. One can see that B-ResNet has converged to an architecture which is a little bit different than the standard ResNet architecture, explaining why its accuracy is better. On the CNF side, our technique is able to extract a model that has a minimum of high-resolution convolutions operations, resulting in a high speedup.

Model	FLOPs (millions)	Accuracy
ResNet [10]		our/*original*
ResNet-110	253.70	94.09/*93.57*
ResNet-56	126.01	93.61/*93.03*
ResNet-44	97.64	93.21/*92.83*
ResNet-32	69.27	92.91/*92.49*
ResNet-20	40.90	92.19/*91.25*
Low Cost Collaborative Layer [7]		
LCCL (ResNet-110)	166	93.44
LCCL (ResNet-44)	65	92.71
LCCL (ResNet-32)	49	92.56
LCCL (ResNet-20)	26	91.68
Multi Scale DenseNet [13] (values read on plot)		
MSDNet	≈ 255	94.1
	≈ 225	94.0
	≈ 205	94.0
	≈ 180	94.0
	≈ 145	93.8
	≈ 119	93.7
	≈ 97	93.3
	≈ 80	92.8
	≈ 45	91.8
	≈ 25	90.0
Budgeted ResNet		
B-ResNet	407.51	94.29
	258.20	94.15
	152.60	94.01
	120.20	93.71
	56.47	92.92
	42.69	92.48
	39.25	92.39
Convolutional Neural Fabrics [24]		our/*original*
CNF W=8	2,219.00	94.83/*90.58*
CNF W=4	1,010.00	93.75/*87.91*
CNF W=2	406.00	92.54/*86.21*
CNF W=1	54.00	89.91
Budgeted CNF		
B-CNF	2,150.00	94.92
	1,407.00	94.85
	1,144.00	94.69
	103.00	93.14
	85.00	92.17

Table 1: Accuracy/speed trade-off on CIFAR-10 using B-ResNet and B-CNF. Values reported as *our* corresponds to results we obtained when training a reproduction of the models, *original* corresponds to values from the original article.

Reducing the memory consumption: Similar experiments have been made considering the *memory consumption cost* that measures the number of parameters of the learned architectures. We want to demonstrate here the ability of our technique to be used with a large variety of

Model	FLOPs (millions)	Accuracy (%)
ResNet-110	253.7	71.85
ResNet-56	126	70.57
ResNet-44	97.64	70.28
ResNet-32	69.27	69.28
ResNet-20	40.9	67.14
MSDNet [13]	215	76
	180	75
	150	74
	109	72.5
	80	71
	45	67.5
	15	62.5
B-ResNet	349.5	73.28
	115.09	71.46
	69.84	70.27
	64.96	70.12
	46.29	69.02
	39.22	68.45

Table 2: Accuracy/speed trade-off on Cifar-100 using ResNet Fabrics.

Model	# params (millions)	Accuracy (%)
ResNet [10]		our/*original*
ResNet-110	1.73	94.09/*93.57*
ResNet-56	0.86	93.61/*93.03*
ResNet-44	0.66	93.21/*92.83*
ResNet-32	0.47	92.91/*92.49*
ResNet-20	0.27	92.19/*91.25*
Budgeted ResNet		
B-ResNet	4.38	94.35
	2.27	94.2
	1.29	93.85
	0.48	93.42
	0.34	92.72
	0.3	92.52
	0.29	92.17
Convolutional Neural Fabrics [24]		our/*original*
CNF W=8	18.04	94.83/*90.58*
CNF W=4	8.58	93.75/*87.91*
CNF W=2	3.85	92.54/*86.21*
CNF W=1	0.74	89.91
Budgeted CNF		
B-CNF	7.56	94.88
	4.98	94.58
	4.28	94.55
	3.67	94.42
	2.65	94.00
	1.19	93.53

Table 3: Accuracy/memory trade-off on Cifar-10 using B-ResNet and B-CNF.

Figure 4: The figure illustrates the evolution of the loss function during learning iterations for CNF and B-CNF, and also the entropy of Γ. the period between iteration 0 and 50 is the burn-in phase, at iteration 150, the learning rate has been decreased to increase the convergence speed.

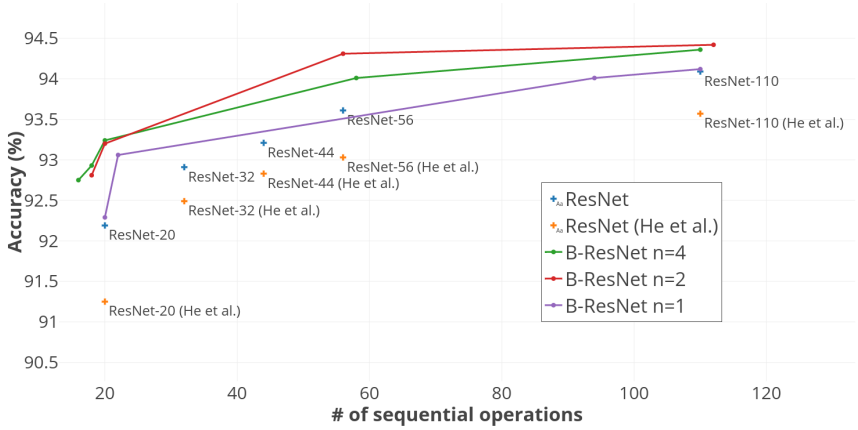
costs, and not only to reduce the computation time. Table 3 illustrates the results obtained on CIFAR-10. As with the *computation cost*, one can see that our approach is able to discover architectures that, given a particular memory cost, obtain a better accuracy. For example, for a model which size is ≈ 0.47 millions parameters, ResNet-32 has a classification error of 7.81% while B-ResNet only makes 6.58% error with ≈ 0.48 million parameters.

Image Segmentation: We also perform experiments on the image segmentation task using the Part Label dataset with CNF and B-CNF since CNF can also be used for this task – Table 4. In that case, the model computes a map of pixel probabilities, the output layer being now located at the top-right position of the CNF matrix. It is thus more difficult to reduce the overall computation cost. On the Part Label dataset, we are able to learn a BSN model with a computation gain of 40%. Forcing the model to reduce further the computation cost by decreasing the value of $\mathbf{C}$ results in inconsistent models i.e models where the input and output layers are not connected. At a Flop gain level of 40%, BSN obtains an error rate of 4.57%, which can be compared with the error of 4.94% for the full model.

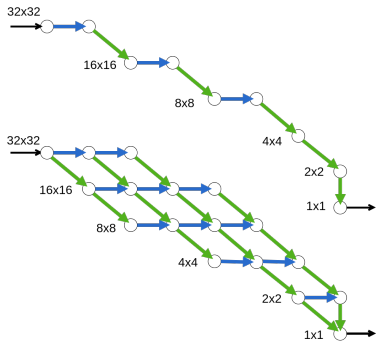
Learning Dynamics: Figure 4 illustrates the learning dynamics of B-CNF and CNF. First, one can see (entropy curve) that the model becomes deterministic at the end of the learning iteration, and thus converges to a unique architecture. Moreover, the training speed of B-CNF and CNF are comparable showing that our method does not result in a slower training procedure. Note that the figure illustrates the fact that during a burn-in period, we don't update the edges probabilities, which allows us to obtain a faster convergence speed.

4.3 Learning Distributed Architectures

At last, we perform a third set of experiments focused on distributed computing where different edges can be computed simultaneously on different computers/processors of a distributed platform. We thus evaluate the quality of an architecture by its ability to be efficiently parallelized. The



(a) Accuracy/number of operation for different number of cores on CIFAR-10 using B-ResNet.



(b) Architectures discovered with B-CNF for different number of cores: $n = 1$ (top) and $n = 4$ (bottom)

Figure 5: Architectures discovered on CIFAR-10 for different number of distributed cores n.

Model	FLOPs(billions)	Accuracy
CNF	35.614	95.06
CNF W=8 [24]	35.614	95.39
B-CNF	28.49	95.21
B-CNF	21.37	95.43

Table 4: Accuracy/Speed trade-off on Part Label using CNF.

distributed computation cost corresponds to the number of steps needed to compute the output of the network e.g on an architecture with $n = 2$ computers, depending on the structure of the network, two edges could be computed simultaneously. If the architecture is a sequence of layers, then this parallelization becomes impossible. Theses experiments allow us to measure the ability of BSN to handle complex costs that cannot be decomposed as a sum of individual modules costs as it is usually done in related works. Results and corresponding architectures are illustrated in Figure 5 for the CIFAR-10 dataset and for both the B-ResNet and the B-CNF architectures. Note that ResNet is typically an architecture that cannot be efficiently distributed since it is a sequences of modules. One can see that our approach is able to find efficient architectures for $n = 2$ and $n = 4$. Surprisingly, when $n = 4$ the discovered architectures are less efficient, which is mainly due to an over fitting on the training set, the cost constraint becoming too large and stopping to act as a regularizer on the network architecture. On Figure 5b, one can see two examples of architectures discovered when $n = 1$ and $n = 4$. The shape of the architecture when $n = 4$ clearly confirm that BSN is able to discover parallelized architectures, and to 'understand' the structure of this complex cost.

5 Related Work

Learning cost-efficient models: One of the first approaches to learn efficient models is to *a posteriori* compress the learned network, typically by pruning some connections. The oldest work is certainly the Optimal Brain Surgeon [9] which removes weights in a classical neural network. The problem of network compression can also be seen as a way to speed up a particular architecture, for example by using quantization of the weights of the network [28]. Other algorithms include the use of hardware efficient operations that allow a high speedup [6].

Efficient architectures: Architecture improvements have been widely used in convolutional neural networks to improve cost efficiency of network components, some examples are the bottleneck units in the ResNet model [10], the use of depthwise separable convolution in Xception [3] and the lightweight MobileNets[12] or the combinaison of pointwise group convolution and channel shuffle in ShuffleNet[29].

End-to-end approaches: A first example of end-to-end approaches is the usage of quantization at training time: different authors trained models using binary weight quantization coupled with full precision arithmetic operations [4],[16]. Recently, [18] proposed a method using half precision floating numbers during training. Another technique proposed by [11], [23] and used in [30, 20] is the distillation of knowledge, which consists of training a smaller network to imitate the outputs of a larger network. Other approaches are dynamic networks which conditionally select the modules to respect a budget objective.[2, 21, 13, 1, 17].

Architecture Search: Different authors have proposed to provide networks with the ability to learn to select the computations that will be applied i.e choosing the right architecture for a particular task. This is the case for example for classification in [5, 31] based on Reinforcement learning techniques, in [25] based on gating mechanisms, in [22] based on evolutionary algorithms or even in [8] based on both RL and evolutionary techniques. With respect to existing methods, the strongest different is that we do not make any assumption concerning the nature of the cost which can be a computation cost, a memory cost, etc... Our model is thus more generic than existing techniques and allow to handle a large variety of problems.

6 Conclusion and Perspectives

We proposed a new model called *Budgeted Super Network* able to automatically discover cost-constrained neural network architectures by specifying a maximum authorized cost. The experiments in the computer vision domain show the effectiveness of our approach. Its main advantage is that BSN can be used for any costs (computation cost, memory cost, etc.) without any assumption on the shape of this cost. A promising research direction is now to study whether this model could be adapted in order to reduce the training time (instead of the test computation time). This could for example be done using meta-learning approaches.

Acknowledgments

This work has been funded in part by grant ANR-16-CE23-0016 "PAMELA" and grant ANR-16-CE23-0006 "Deep in France".

References

[1] E. Bengio, P. Bacon, J. Pineau, and D. Precup. Conditional computation in neural networks for faster models. *CoRR*, abs/1511.06297, 2015.

[2] T. Bolukbasi, J. Wang, O. Dekel, and V. Saligrama. Adaptive neural networks for fast test-time prediction. *CoRR*, abs/1702.07811, 2017.

[3] F. Chollet. Xception: Deep learning with depthwise separable convolutions. *CoRR*, abs/1610.02357, 2016.

[4] M. Courbariaux, Y. Bengio, and J. David. Binaryconnect: Training deep neural networks with binary weights during propagations. *CoRR*, abs/1511.00363, 2015.

[5] L. Denoyer and P. Gallinari. Deep sequential neural network. *CoRR*, abs/1410.0510, 2014.

[6] J. Devlin. Sharp Models on Dull Hardware: Fast and Accurate Neural Machine Translation Decoding on the CPU, 2017.

[7] X. Dong, J. Huang, Y. Yang, and S. Yan. More is less: A more complicated network with less inference complexity. *CoRR*, abs/1703.08651, 2017.

[8] C. Fernando, D. Banarse, C. Blundell, Y. Zwols, D. Ha, A. A. Rusu, A. Pritzel, and D. Wierstra. Pathnet: Evolution channels gradient descent in super neural networks. *CoRR*, abs/1701.08734, 2017.

[9] B. Hassibi and D. G. Stork. Second order derivatives for network pruning: Optimal brain surgeon. In *Advances in Neural Information Processing Systems 5, [NIPS Conference]*, pages 164–171, San Francisco, CA, USA, 1993. Morgan Kaufmann Publishers Inc.

[10] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[11] G. E. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. *CoRR*, abs/1503.02531, 2015.

[12] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017.

[13] G. Huang, D. Chen, T. Li, F. Wu, L. van der Maaten, and K. Q. Weinberger. Multi-scale dense convolutional networks for efficient prediction. *CoRR*, abs/1703.09844, 2017.

[14] A. Kae, K. Sohn, H. Lee, and E. Learned-Miller. Augmenting CRFs with Boltzmann machine shape priors for image labeling. In *CVPR*, 2013.

[15] A. Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

[16] L. Lu. Toward computation and memory efficient neural network acoustic models with binary weights and activations. *CoRR*, abs/1706.09453, 2017.

[17] M. McGill and P. Perona. Deciding how to decide: Dynamic routing in artificial neural networks. *CoRR*, abs/1703.06217, 2017.

[18] P. Micikevicius, S. Narang, J. Alben, G. F. Diamos, E. Elsen, D. Garcia, B. Ginsburg, M. Houston, O. Kuchaiev, G. Venkatesh, and H. Wu. Mixed precision training. *CoRR*, abs/1710.03740, 2017.

[19] R. Miikkulainen, J. Z. Liang, E. Meyerson, A. Rawal, D. Fink, O. Francon, B. Raju, H. Shahrzad, A. Navruzyan, N. Duffy, and B. Hodjat. Evolving deep neural networks. *CoRR*, abs/1703.00548, 2017.

[20] F. Nan and V. Saligrama. Adaptive Classification for Prediction Under a Budget. *ArXiv e-prints*, May 2017.

[21] A. Odena, D. Lawson, and C. Olah. Changing model behavior at test-time using reinforcement learning. *CoRR*, abs/1702.07780, 2017.

[22] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, Q. V. Le, and A. Kurakin. Large-scale evolution of image classifiers. *CoRR*, abs/1703.01041, 2017.

[23] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio. Fitnets: Hints for thin deep nets. *CoRR*, abs/1412.6550, 2014.

[24] S. Saxena and J. Verbeek. Convolutional neural fabrics. *CoRR*, abs/1606.02492, 2016.

[25] R. K. Srivastava, K. Greff, and J. Schmidhuber. Highway networks. *CoRR*, abs/1505.00387, 2015.

[26] K. O. Stanley and R. Miikkulainen. Efficient reinforcement learning through evolving neural network topologies. In *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference, New York, USA, 9-13 July 2002*, pages 569–577, 2002.

[27] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014.

[28] V. Vanhoucke, A. Senior, and M. Z. Mao. Improving the speed of neural networks on cpus. In *Deep Learning and Unsupervised Feature Learning Workshop, NIPS 2011*, 2011.

[29] X. Zhang, X. Zhou, M. Lin, and J. Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. *CoRR*, abs/1707.01083, 2017.

[30] H. Zhu, F. Nan, I. Paschalidis, and V. Saligrama. Sequential Dynamic Decision Making with Deep Neural Nets on a Test-Time Budget. *ArXiv e-prints*, May 2017.

[31] B. Zoph and Q. V. Le. Neural architecture search with reinforcement learning. *CoRR*, abs/1611.01578, 2016.