Learning Time/Memory-Efficient Deep Architectures with Budgeted Super Networks

Supplementary Material

Demonstration of Proposition 1

Let us consider the stochastic optimization problem defined in Equation 3. The schema of the proof is the following:

- First, we lower bound the value of Equation 3 by the optimal value of Equation 2.
- Then we show that this lower bound can be reached by some particular values of Γ and θ in Equation 3. Said otherwise, the solution of Equation 3 is equivalent to the solution of 2.

Let us denote:

$$B(H \odot E, \theta, \lambda) = \frac{1}{\ell} \sum_i \Delta(f(x^i, H \odot E, \theta), y^i) + \lambda \max(0, C(H \odot E) - C) \quad (8)$$

Given a value of Γ, let us denote $supp(\Gamma)$ all the H matrices that can be sampled following Γ. The objective function of Equation 3 can be written as:

$$\begin{aligned} E_{H\sim\Gamma}[B(H \odot E, \theta, \lambda)] &= \sum_{H \in supp(\Gamma)} B(H \odot E, \theta, \lambda) P(H|\Gamma) \\ &\geq \sum_{H \in supp(\Gamma)} B((H \odot E)^*, \theta^*, \lambda) P(H|\Gamma) \\ &= B((H \odot E)^*, \theta^*, \lambda) \end{aligned} \quad (9)$$

where $(H \odot E)^*$ and θ^* correspond to the solution of:

$$(H \odot E)^*, \theta^* = \arg\min_{H,\theta} B(H \odot E, \theta, \lambda) \quad (10)$$

Now, it is easy to show that this lower bound can be reached by considering a value of Γ^* such that $\forall H \in supp(\Gamma), H \odot E = (H \odot E)^*$. This corresponds to a value of Γ where all the probabilities associated to edges in E are equal to 0 or to 1.

Gradient computation

$$\nabla_{\theta,\Gamma} \mathcal{L}(x, y, E, \Gamma, \theta) = \nabla_{\theta,\Gamma} \mathbb{E}_{H\sim\Gamma} \mathcal{D}(x, y, \theta, E, H) \quad (11)$$

$$= \nabla_{\theta,\Gamma} \sum_H P(H|\Gamma) \mathcal{D}(x, y, \theta, E, H) \quad (12)$$

$$= \sum_H \nabla_{\theta,\Gamma} (P(H|\Gamma) \mathcal{D}(x, y, \theta, E, H)) \quad (13)$$

$$= \sum_H \nabla_{\theta,\Gamma} (P(H|\Gamma)) \mathcal{D}(x, y, \theta, E, H) + P(H|\Gamma) \nabla_{\theta,\Gamma} \mathcal{D}(x, y, \theta, E, H) \quad (14)$$

$$= \sum_H P(H|\Gamma) \nabla_{\theta,\Gamma} \log P(H|\Gamma) \mathcal{D}(x, y, \theta, E, H) + P(H|\Gamma) \nabla_{\theta,\Gamma} \mathcal{D}(x, y, \theta, E, H) \quad (15)$$

Using Equation 4:

$$= \sum_H P(H|\Gamma) ((\nabla_{\theta,\Gamma} \log P(H|\Gamma)) \mathcal{D}(x, y, \theta, E, H) + \nabla_{\theta,\Gamma} \Delta(f(x, H \odot E, \theta), y)) \quad (16)$$

$$= \sum_H P(H|\Gamma) \left[(\nabla_{\theta,\Gamma} \log P(H|\Gamma)) \mathcal{D}(x, y, \theta, E, H) \right] + \sum_H P(H|\Gamma) \left[\nabla_{\theta,\Gamma} \Delta(f(x, H \odot E, \theta), y) \right] \quad (17)$$

$$= \sum_H P(H|\Gamma) \left[(\nabla_{\theta,\Gamma} \log P(H|\Gamma)) \Delta(f(x, H \odot E, \theta), y) \right] + \lambda \sum_H P(H|\Gamma) \left[(\nabla_{\theta,\Gamma} \log P(H|\Gamma)) \max(0, C(H \odot E) - \mathbf{C}) \right] + \sum_H P(H|\Gamma) \left[\nabla_{\theta,\Gamma} \Delta(f(x, H \odot E, \theta), y) \right] \quad (18)$$

Segmentation architecture

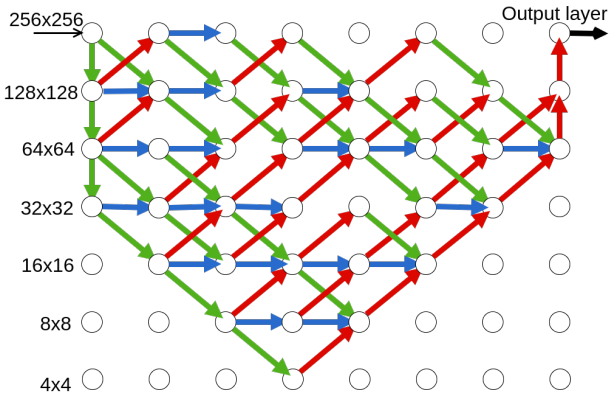

Figure 6: Segmentation architecture

Figure 6 is an example of segmentation architecture discovered on the Part Label dataset using the *flop cost*. It is interesting to note that only one layer with 256x256 input and output is kept and that most of the computations are done at lower less-expensive layers.

Model Selection Protocol

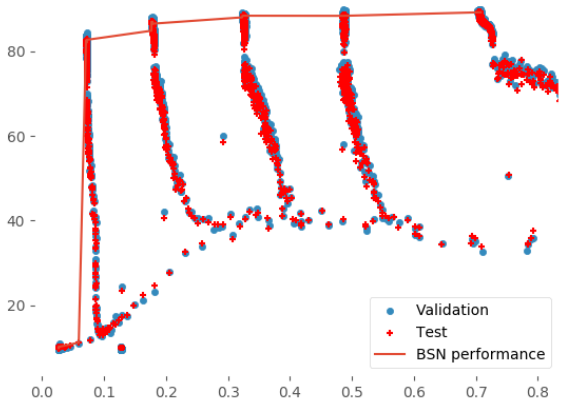

Figure 7: Model selection

The selection of reported models is obtained by learning many different models, computing the Pareto front of the accuracy/cost curve on the validation set, and reporting the performance obtained on the test set. This is illustrated in figure 7 where many different models are reported on the validation set(blue circles) with the corresponding performance on the test set (red crosses).

Considering non-differentiable costs

Stochastic costs in the REINFORCE algorithm:

As explained previously, the proposed algorithm can also be used when the cost $C(H \odot E)$ is a stochastic function that depends on the environment e.g the network latency, (or even on the input data x). Our algorithm is still able to learn with such stochastic costs since the only change in the learning objective is that the expectation is now made on both H and C (and x if needed). This property is interesting since it allows to discover efficient architecture on stochastic operational infrastructure.

Distributed computation cost Taking the real-life example of a network which will, once optimized, have to run on a given computing infrastructure, the *distributed computation cost* is a measure of how "parallelizable" an architecture is. This cost function takes the following three elements as inputs (i)A network architecture (represented as a graph for instance), (ii)An allocation algorithm and (iii) a maximum number of concurrent possible operations. The cost function then returns the number of computation cycles required to run the architecture given the allocation strategy.

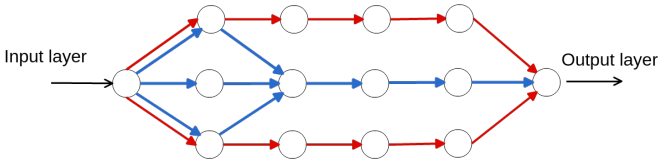

Figure 8: Two networks illustrating the need to have a cost function evaluating the global architecture of a network. Considering an environment with $n = 2$ machines performing computations in parallel, the blue network composed of 9 computational modules has a *distributed computation cost* of 6 while the red network, composed of 10 modules, has a smaller cost of 5.

Additional Architecture Details

ResNet Fabric

Based on the ResNet architecture, the structure of a ResNet Fabric is a stack of k groups of layers, each group being composed of $2n$ layers where n represents the width of the Fabric. The feature maps size and number of filters stay constant across the layers of each group and are modified between groups.

Due to its linear structure, the standard ResNet architecture spans a limited number of possible (sub-)architectures. In order to increase the size of the search space, we add several connections between groups as shown in 1a: each block in the second to last groups receives two (for the first

and last block of each group) or three (for every other block) inputs from preceding groups. To stay consistent with the rest of the network, each connection is a *basic block* [11] composed of 2 convolutional layers and a shortcut connection.

In our experiments, we use stacks of $k = 3$ blocks and $n = \{3, 5, 7, 9, 18\}$ to respectively include the ResNet-$\{20, 32, 44, 56, 110\}$ in the Fabric. Between each block, the feature maps size is reduced by a factor of 2 and the number of feature maps is doubled.

Convolutional Neural Fabric

The second network we use in our experiments is based on the dense Convolutional Neural Fabrics, which can be seen as a multi-layer and multi-scale convolutional neural network. As shown in Figure 1b, this architecture has 2 axis: The first axis represents the different columns (or width) W of the network while the second axis corresponds to different scales (or height) H of output feature maps, the first scale being the size of the input images, each subsequent scale being of a size reduced by a factor of 2 up to the last scale corresponding to a single scalar.

Each layer (l, s) in this fabric takes its input from three different layers of the preceding column: (i) One with a finer scale $(l - 1, s - 1)$ on which a convolution with stride 2 is applied to obtain feature maps having half the size of the input, (ii) one with the same scale $(l - 1, s)$ on which a convolution with stride 1 is applied to obtain feature map of the same resolution as the input and (iii) one with a coarser scale $(l - 1, s + 1)$ on which convolution with stride 1 is applied after a factor 2 up-sampling to obtain feature maps having twice the size of the input. The three feature blocks are then added before passing through the ReLU activation function to obtain the final output of this layer (l, s).

The first and last columns are the only two which have vertical connections within scales of the same layer (as can be seen in Figure 1b). This is made to allow the propagation of the information to all nodes in the first column and to aggregate the activations of the last column to compute the final prediction. A more detailed description of this architecture can be found in the CNF original article.

We used two different Convolutional Neural Fabrics in our experiments: One for the classification task (CIFAR-10 and CIFAR-100) with $W = 8$ columns, $H = 6$ scales and 128 filters per convolution and one for the segmentation task (Part Label) with $W = 8$ layers, $H = 9$ scales (from 256x256 to 1x1 feature map sizes) and 64 filters per convolution.

Additional Learning Details

Datasets

CIFAR-10. The CIFAR-10 dataset consists of 60k 32x32 images with 10 classes and 6000 images per class. The dataset is decomposed in 50k training and 10k testing images. We split the training set following the standard, i.e 45k training samples and 5k validation samples. We use two data augmentation techniques: padding the image to 36x36 pixels before extracting a random crop of size 32x32 and horizontally flipping. Images are then normalized in the range [-1,1].

CIFAR100. The CIFAR-100 dataset is similar to CIFAR-10, with 100 classes and 600 images per class. We use the same train/validation split and data augmentation technique as with CIFAR-10.

Part Labels. The Part Labels dataset is a subset of the LFW dataset composed of 2927 250x250 face images in which each pixel is labeled as one of the Hair/Skin/Background classes. The standard split contains 1500 training samples, 500 validation samples and 927 test samples. Images are zero-padded from 250x250 to 256x256. We use horizontal flipping as data augmentation. Images are then normalized in the range [-1,1].

Learning procedure

When training our budgeted models, we first train the network for 50 "warm-up" epochs during which no sampling is done (The whole super network is trained). After this warm-up phase, the probability of each edge is initialized and we start sampling architectures.

The real-valued distribution parameter associated with each layer (and used to generate the probability of sampling the edge) are all initialized to 3, resulting in a ≈ 0.95 initial probability once passed through the sigmoid activation function.

On CIFAR-10 and CIFAR-100 datasets we train all models for 300 epochs. We start with a learning rate of 10^{-1} and divide it by 10 after 150 and 225 epochs. On Part Label dataset all models are trained for 200 epochs with a learning rate initialized to 10^{-1} and divided by 10 after 130 epochs.

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}.

Forward algorithm

Given the SS-Network (E, Γ, θ) and input x, the evaluation of $f(x, E, \Gamma, \theta)$ is done as follow :

Algorithm 1 Stochastic Super Network forward algorithm

procedure SSN-FORWARD(x, E, Γ, θ)
  $H \sim \Gamma$ ▷ as explained in Section 3.2
  for $i \in [1..N]$ **do**
    $l_i \leftarrow \emptyset$
  end for
  $l_1 \leftarrow x$
  for $i \in [2..N]$ **do**
    $l_i \leftarrow \sum_{k<i} e_{k,i} h_{k,i} f_{k,i}(l_k)$
  end for
  return l_N
end procedure

Additional results

Model	# of sequential operations	Accuracy %
ResNet [11]		our/*original*
ResNet-110	110.00	94.09/*93.57*
ResNet-56	56.00	93.61/*93.03*
ResNet-44	44.00	93.21/*92.83*
ResNet-32	32.00	92.91/*92.49*
ResNet-20	20.00	92.19/*91.25*
Busgeted ResNet		
B-ResNet	110.00	94.36
	58.00	94.01
	20.00	93.24
	18.00	92.93
	16.00	92.75
Convolutional Neural Fabric [25]		our/*original*
CNF W=8	53.00	94.83/*90.58*
CNF W=4	31.00	93.75/*87.91*
CNF W=2	19.00	92.54/*86.21*
CNF W=1	12.00	89.91
Budgeted CNF		
B-ResNet	31.00	94.96
	25.00	94.72
	21.00	94.36
	18.00	93.86

Table 5: Results for *Distributed computation cost* on CIFAR-10 with $n = 4$

Model	# of sequential operations	Accuracy %
ResNet [11]		our/*original*
ResNet-110	112.00	94.09/*93.57*
ResNet-56	58.00	93.61/*93.03*
ResNet-44	46.00	93.21/*92.83*
ResNet-32	34.00	92.91/*92.49*
ResNet-20	22.00	92.19/*91.25*
Budgeted ResNet		
B-ResNet	184.00	94.42
	110.00	94.12
	94.00	94.01
	22.00	93.06
	20.00	92.29
Convolutional Neural Fabrics [25]		our/*original*
CNF W=8	171.00	94.83/*90.58*
CNF W=4	83.00	93.75/*87.91*
CNF W=2	39.00	92.54/*86.21*
CNF W=1	12.00	89.91
Budgeted CNF		
B-CNF	98.00	95.02
	50.00	94.62
	45.00	94.55
	39.00	94.35
	33.00	93.00
	26.00	92.91
	18.00	92.87

Table 6: Results for *Distributed computation cost* on CIFAR-10 with $n = 1$

Model	# of sequential operations	Accuracy (%)
ResNet[11]		
ResNet-110	112.00	71.85
ResNet-56	58.00	70.57
ResNet-44	46.00	70.28
ResNet-32	34.00	69.28
ResNet-20	22.00	67.14
Budgeted ResNet		
B-ResNet	320.00	74.35
	184.00	73.85
	110.00	72.88
	67.00	72.02
	32.00	69.60
	20.00	68.48

Table 7: Results for *Distributed computation cost* on CIFAR-100 with $n = 1$

Model	# of sequential operations	Accuracy %
ResNet [11]		our/*original*
ResNet-110	110.00	94.09/*93.57*
ResNet-56	56.00	93.61/*93.03*
ResNet-44	44.00	93.21/*92.83*
ResNet-32	32.00	92.91/*92.49*
ResNet-20	20.00	92.19/*91.25*
Budgeted ResNet		
B-ResNet	179.00	94.36
	112.00	94.42
	56.00	94.31
	20.00	93.20
	18.00	92.81
Convolutional Neural Fabric [25]		our/*original*
CNF W=8	90.00	94.83/*90.58*
CNF W=4	47.00	93.75/*87.91*
CNF W=2	26.00	92.54/*86.21*
CNF W=1	12.00	89.91
Budgeted CNF		
B-CNF	47.00	94.67
	30.00	94.68
	28.00	94.58
	24.00	94.41
	20.00	94.35
	18.00	92.86

Table 8: Results for *Distributed computation cost* on CIFAR-10 with $n = 2$

Model	# of sequential operations	Accuracy (%)
ResNe [11]		
ResNet110	110.00	71.85
ResNet56	56.00	70.57
ResNet44	44.00	70.28
ResNet32	34.00	69.28
ResNet20	20.00	67.14
Budgeted ResNet		
B-ResNet	179.00	74.35
	112.00	73.85
	49.00	71.84
	29.00	69.94
	22.00	69.09

Table 9: Results for *Distributed computation cost* on CIFAR-100 with $n = 2$