Adaptative Inference Cost With Convolutional Neural Mixture Models

Adria Ruiz Jakob Verbeek
Univ. Grenoble Alpes, Inria, CNRS, Grenoble INP, LJK, 38000 Grenoble, France
firstname.lastname@inria.fr

Abstract

Despite the outstanding performance of convolutional neural networks (CNNs) for many vision tasks, the required computational cost during inference is problematic when resources are limited. In this context, we propose Convolutional Neural Mixture Models (CNMMs), a probabilistic model embedding a large number of CNNs that can be jointly trained and evaluated in an efficient manner. Within the proposed framework, we present different mechanisms to prune subsets of CNNs from the mixture, allowing to easily adapt the computational cost required for inference. Image classification and semantic segmentation experiments show that our method achieve excellent accuracy-compute trade-offs. Moreover, unlike most of previous approaches, a single CNMM provides a large range of operating points along this trade-off, without any re-training.

1. Introduction

Convolutional neural networks (CNNs) form the basis of many state-of-the-art computer vision models. Despite their outstanding performance, the computational cost of inference in these CNN-based models is typically very high. This holds back applications on mobile platforms, such as autonomous vehicles, drones, or phones, where computational resources are limited, concurrent data-streams need to be processed, and low-latency prediction is critical.

To accelerate CNNs we can reduce their complexity before training, *e.g.* by decreasing the number of filters or network layers. This solution, however, may lead to sub-optimal results given that over-parametrization plays a critical role in the optimization of deep networks [7, 9]. Fortunately, other studies have found a complementary phenomena: given a trained CNN, a large number of its filters are redundant and do not have a significant impact on the final prediction [26]. Motivated by these two findings, much research has focused on accelerating CNNs using network pruning [11, 19, 22, 35, 38, 41, 47, 56]. Pruning can be applied at multiple levels, *e.g.* by removing independent filters [35, 41], groups of them [11, 19], or entire layers [56].

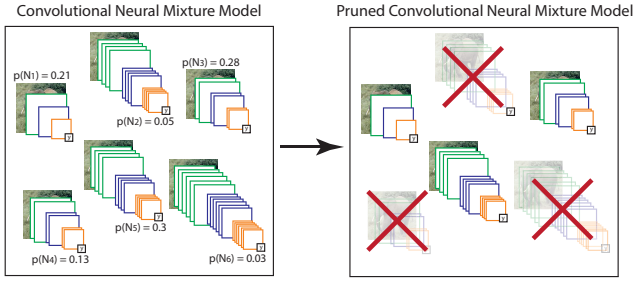

Figure 1. A Convolutional Neural Mixture Model embeds a large number of CNNs. Weight sharing enables efficient joint training of all networks and computation of the mixture output. The learned mixing weights can be used to remove networks from the mixture, and thus reduce the computational cost of inference.

Despite the encouraging results of these methods, their ability to provide a wide range of operating points along the trade-off between accuracy and computation is limited. The reason is that these approaches typically require to train a separate model for each specific pruning level.

In this paper, we propose Convolutional Neural Mixture Models (CNMMs), which provide a novel perspective on network pruning. A CNMM define a distribution over a large number of CNNs. The mixture is naturally pruned by removing networks with low probabilities, see Figure 1. Despite the appealing simplicity of this approach, it presents several challenges. First, learning a large ensemble of CNNs may require a prohibitive amount of computation. Second, even if many networks in the mixture are pruned, their independent evaluation during inference is likely to be less efficient than computing the output of a single large model. In order to ensure tractability, we design a parameter-sharing scheme between different CNNs. This enables us to (i) jointly train all the networks, and (ii) efficiently compute an approximation of the mixture output without independently evaluating all the networks.

Image classification and semantic segmentation experiments show that CNMMs achieve an excellent trade-off between prediction accuracy and computational cost. Unlike most previous network pruning approaches, a single CNMM model achieves a wide range of operating points along this trade-off without any re-training.

2. Related work

Neural network ensembles. Learning ensembles of neural networks is a long-standing research topic. Seminal works explored different strategies to combine the outputs of different networks to obtain more accurate predictions [30, 46, 61]. Recently, the success of deep models has renewed interest in ensemble methods.

For this purpose, many approaches have been explored. For instance, [31, 62] used bagging [3] and boosting [49] to train multiple networks. Other works have considered to learn diverse models by employing different parameter initializations [34], or re-training a subset of layers [60]. While these strategies are effective to learn diverse networks, their main limitation is the required training cost. In practice, training a deep model can take multiple days, and therefore large ensembles may have a prohibitive cost. To reduce the training time, it has been suggested [18, 39] to train a single network and to use parameters from multiple iterations of the optimization process to define the ensemble. Despite the efficiency of this method during training, this approach does not reduce inference cost, since multiple networks must be evaluated independently at test time.

An alternative strategy to allow efficient training and inference is to use implicit ensembles [11, 21, 32, 47]. By relying on sampling, these methods allow to jointly train all the individual components in the ensemble and perform approximate inference during testing. Bayesian neural networks (BNNs) fall in this paradigm and use a distribution over parameters, rather than a single point estimate [11, 28, 40, 47]. A sample from the parameter distribution can be considered as an individual network. Other works have implemented the notion of implicit ensembles by using dropout [52] mechanisms. Dropping neurons can be regarded as sampling over a large ensemble of different networks [2]. Moreover, scaling outputs during testing according to the dropout probability can be understood as an approximated inference mechanism. Motivated by this idea, different works have applied dropout over individual weights [10], network activations [50], or connections in multi-branch architectures [12, 32]. Interestingly, it has been observed that ResNets [13] behave like an ensemble of models, where some residual connections can be removed without significantly reducing prediction accuracy [55]. This idea was used by ResNets with stochastic depth [21], where different dropout probabilities are assigned to the residual connections.

Our proposed Convolutional Neural Mixture Model is an implicit ensemble defining a mixture distribution over an exponential number of CNNs. This allows to use the learned probabilities to prune the model by removing non-relevant networks. Using a mixture of CNNs for model pruning is a novel approach, which contrasts to previous methods employing ensembles for other purposes such as boosting performance [8, 34], improving learning dynamics [21], or uncertainty estimation [24, 31].

Efficient inference in deep networks. A number of strategies have been developed to reduce the inference time of CNNs, including the design of efficient convolutional operators [16, 23, 58], knowledge distillation [4, 15], neural architecture search [14, 63], weight compression [43, 54], and quantization [25, 36]. Network pruning has emerged as one of the most effective frameworks for this purpose [11, 33, 35, 56]. Pruning methods aim to remove weights which do not have a significant impact on the network output. Among these methods, we can differentiate between two main strategies: online and offline pruning. In offline pruning, a network is first optimized for a given task using standard training. Subsequently, non-relevant weights are identified using different heuristics including their norm [35], similarity to other weights [51], or second order derivatives [33]. The main advantage of this strategy is that it can be applied to any pre-trained network. However, these approaches require a costly process involving several prune/retrain cycles in order to recover the original network performance. Online approaches, on the other hand, perform pruning during network training. For example, sparsity inducing regularization can be used over individual weights [38, 40], groups of them [11, 19, 47], or over the connections in multi-branch architectures [1, 56]. These methods typically have a hyper-parameter, to be set before training, determining the trade-off between the final performance and the pruning ratio.

In contrast to previous approaches, we prune entire CNNs by removing the networks with the smallest probabilities in the mixture. This approach offers two main advantages. First, it does not require to define a hyper-parameter before training to determine the balance between the potential compression and the final performance. Second, the number of removed networks can be controlled after optimization. Therefore, a learned CNMM can be deployed at multiple operating points to trade-off computation and prediction accuracy. For example, across different devices with varying computational resources, or on the same device with different computational constraints depending on the processor load of other processes. The recently proposed Slimmable Neural Networks [57] have also focused on adapting the accuracy-efficiency trade-off at run time. This is achieved by embedding a small set of CNNs with varying widths into a single model. Different from this approach, our CNMMs embed a large number of networks with different depths, which allows for a finer granularity to control the computational cost during pruning.

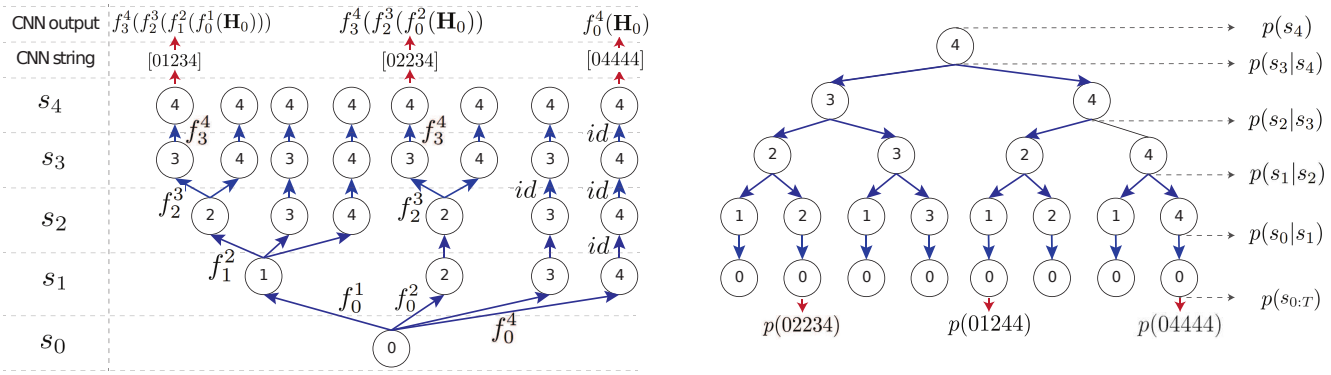

Figure 2. (Left) Illustration of how a large collection of CNNs is represented in a CNMM. Each network is uniquely identified by a non-decreasing sequence $s_{0:T}$, containing numbers from 0 to T. Consecutive entries in the sequence determine the functions $f^{s_t}_{s_{t-1}}$ applied to compute the CNN output. In this manner, sequences with common sub-sequences share functions and their parameters in the corresponding networks. (Right) Illustration of the distribution $p(s_{0:T})$ that defines the mixing weights over the CNN models, here $T = 4$. Each $p(s_{t-1}|s_t)$ is a Bernoulli distribution on whether s_{t-1} equals s_t or $t-1$. This defines a binary tree generating all the valid sequences.

3. Convolutional Neural Mixture Models

Without loss of generality, we consider a CNN as a function $\mathcal{F}(\mathbf{H}_0) = \mathbf{H}_T$ mapping an RGB image $\mathbf{H}_0 \in \mathbb{R}^{W_0 \times H_0 \times 3}$ to a tensor $\mathbf{H}_T \in \mathbb{R}^{W \times H \times C}$. In particular, we assume that $\mathcal{F}$ is defined as a sequence of T operations:

$$\mathcal{F}(\mathbf{H}_0) = f^T_{T-1}(\dots(f^2_1(f^1_0(\mathbf{H}_0)))), \tag{1}$$

where $\mathbf{H}_t = f^t_{t-1}(\mathbf{H}_{t-1})$ is computed from the previous feature map $\mathbf{H}_{t-1}$. We assume that the functions f^t_{t-1} can be either the identity function, or a standard CNN block composed of different operations such as batch-normalization, convolution, activation functions, or spatial pooling. In this manner, the effective depth of the network, *i.e.* the number of non-identity layers f^t_{t-1}, is at most T.

The output tensor $\mathbf{H}_T$ of the CNN is used to make predictions for a specific task. For example, in image classification, a linear classifier over $\mathbf{H}_T$ can be used in order to estimate the class probabilities for the entire image. For semantic segmentation the same linear classifier is used for each spatial position in $\mathbf{H}_T$.

Given these definitions, a convolutional neural mixture model (CNMM) defines a distribution over output $\mathbf{H}_T$ as:

$$p(\mathbf{H}_T|\mathbf{H}_0) = \sum_{\mathcal{F}\in\mathbb{F}} p(\mathcal{F})p(\mathbf{H}_T|\mathbf{H}_0, \mathcal{F}), \tag{2}$$

where $\mathbb{F} = \{\mathcal{F}_1, \mathcal{F}_2, ..., \mathcal{F}_K\}$ is a finite set of CNNs, $p(\mathbf{H}_T|\mathbf{H}_0, \mathcal{F}_k)$ is a delta function centered on the output $\mathcal{F}_k(\mathbf{H}_0)$ of each network, and $p(\mathcal{F})$ defines the mixing weights over the CNNs in $\mathbb{F}$.

3.1. Modelling a distribution over CNNs

We now define mixtures that contain a number of CNNs that is exponential in the maximum depth T, in a way that allows us to manipulate these mixtures in a tractable manner.

Each component in the mixture is a chain-structured CNN uniquely characterised by a sequence $s_{0:T}$ of length $T + 1$, where the sequences are constrained to be a non-decreasing set of integers from 0 to T, *i.e.* with $s_0 = 0$, $s_T = T$ and $s_{t+1} \geq s_t$. This sequence determines the set of functions that are used in Eq. (1). In particular, given a sequence $s_{0:T}$, the output of the corresponding network is computed as:

$$\mathcal{F}(\mathbf{H}_0) = f^{s_T}_{s_{T-1}}(\dots(f^{s_2}_{s_1}(f^{s_1}_{s_0}(\mathbf{H}_0)). \tag{3}$$

For $i < j$ the function f^j_i is a convolutional block as described above with its own parameters, while the functions f^i_i are identity functions that leave the input unchanged.

By, imposing $s_{t-1} \in \{t-1, s_t\}$, there is a one-to-one mapping between sequences $s_{0:T}$ and the corresponding CNNs.[1] If multiple networks use the same function f^j_i, these networks share their parameters on this function, which ensures that the total number of parameters of the mixture does not grow exponentially, although there are exponentially many mixture components. For instance, for $T = 4$, the mixture will be composed of eight different networks illustrated in Figure 2 (Left). From the illustration it is easy to see that, in general, the mixture contains 2^{T-1} components with shared parameters.

In order to define the probabilities $p(\mathcal{F})$ for each network in the mixture, we define a distribution over sequences $s_{0:T}$ as a reversed Markov chain:

$$p(s_{0:T}) = p(s_T)\prod_{t=1}^{T} p(s_{t-1}|s_t). \tag{4}$$

To ensure that sequences have positive probability if and only if they are valid, *i.e.* satisfy the constraints defined

[1] In particular, this constraint ensures that, *e.g.*, the network $f^4_1(f^1_0(\mathbf{H}_0))$ is uniquely encoded by the sequence '01444', ruling out the alternative sequences '01144' and '01114'. See Figure 2 (Left).

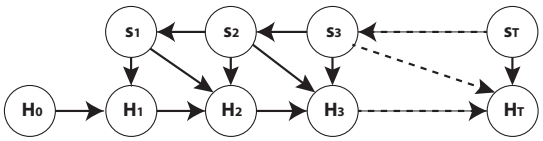

Figure 3. Graphical model representation of the CNMM. The sequence $s_{1:T}$ codes for the CNN architecture. Each $\mathbf{H}_t$ is an intermediate feature map generated by the sampled CNN. It is computed from the previous feature map $\mathbf{H}_{t-1}$ using $f^{s_t}_{s_{t-1}}$.

above, we set $p(s_T = T) = p(s_0 = 0|s_1) = 1$ and define:

$$p(s_{t-1}|s_t) = \begin{cases} \pi^{s_t}_{t-1} & \text{if} \quad s_{t-1} = (t-1), \\ 1 - \pi^{s_t}_{t-1} & \text{if} \quad s_{t-1} = s_t, \\ 0 & \text{otherwise.} \end{cases} \tag{5}$$

As illustrated in Figure 2 (Right), these constraints generate a binary tree generating valid non-decreasing sequences. The conditional probabilities $p(s_{t-1}|s_t)$ are modelled by a Bernoulli distribution with probability $\pi^{s_t}_{s_{t-1}}$, indicating whether the previous number in the sequence is s_t or $t-1$.

3.2. Sampling outputs from CNMMs

The graphical model defined in Figure 3 shows that we can sample from the output distribution $p(\mathbf{H}_T|\mathbf{H}_0)$ in Eq. (2) by first generating a sequence from $p(s_{0:T})$ and then evaluating the associated network with Eq. (3). In the following, we formulate an alternative strategy to sample from the model. This formulation offers two advantages. (i) It is amenable to continuous relaxation, which facilitates learning. (ii) It suggests an iterative algorithm to compute feature map expectations, which can be used instead of sampling for efficient inference.

The conditional $p(\mathbf{H}_t|s_t = l, \mathbf{H}_0)$ gives the distribution over $\mathbf{H}_t$ across the networks with $s_t = l$. For example, $p(\mathbf{H}_2|s_2 = 4, \mathbf{H}_0)$ consists of two weighted delta peaks, located at $f^4_0(\mathbf{H}_0)$ and $f^1_0(f^4_1(\mathbf{H}_0))$, respectively. See Figure 2 (Left). These conditional distributions can be expressed as the forwards recurrence:

$$p(\mathbf{H}_t|s_t, \mathbf{H}_0) = \sum_{\mathbf{H}_{t-1}, s_{t-1}} \Big[p(\mathbf{H}_t|\mathbf{H}_{t-1}, s_t, s_{t-1}) p(s_{t-1}|s_t) \times \underbrace{p(\mathbf{H}_{t-1}|s_{t-1}, \mathbf{H}_0)}_{\text{Recurrent term}} \Big], \tag{6}$$

where $p(\mathbf{H}_t|\mathbf{H}_{t-1}, s_t, s_{t-1})$ is a delta function centered on $f^{s_t}_{s_{t-1}}(\mathbf{H}_{t-1})$. Therefore, unbiased samples $\tilde{\mathbf{h}}^{s_t}_t$ from $p(\mathbf{H}_t|s_t, \mathbf{H}_0)$ can be obtained through sample propagation. Recall from Eq. (5) that, given s_t, there are only two possible values of s_{t-1} that remain, namely s_t and $t-1$. As a consequence, the sum over $\mathbf{s}_{t-1}$ in Eq. (6) only consists of two terms. Given this observation, samples $\tilde{\mathbf{h}}^{s_t}_t \sim p(\mathbf{H}_t|s_t, \mathbf{H}_0)$ can be obtained from samples

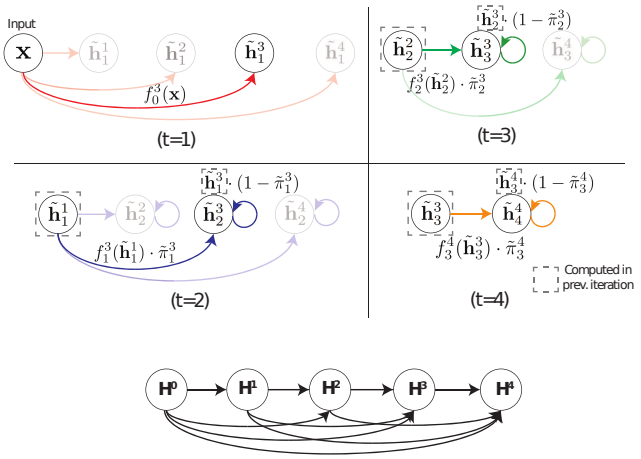

Figure 4. Top: Illustration of the algorithm used to sample intermediate feature maps from the mixture distribution. At each iteration t, we generate $\tilde{\mathbf{h}}^{s_t}_t \sim p(\mathbf{H}_t|s_t, \mathbf{H}_0)$ by using: (i) samples $\tilde{\mathbf{h}}^{t-1}_{t-1}$ obtained in the previous iteration, (ii) the corresponding functions $f^{s_t}_{t-1}$ and (iii) samples $\tilde{\pi}^{s_t}_{t-1}$ from $p(s_{t-1}|s_t)$. Bottom: Network with dense connectivity implementing the sampling algorithm.

$\tilde{\mathbf{h}}^{s_{t-1}}_{t-1} \sim p(\mathbf{H}_{t-1}|s_{t-1}, \mathbf{H}_0)$ as:

$$\tilde{\mathbf{h}}^{s_t}_t = \tilde{\pi}^{s_t}_{t-1} f^{s_t}_{t-1}(\tilde{\mathbf{h}}^{t-1}_{t-1}) + (1 - \tilde{\pi}^{s_t}_{t-1}) \tilde{\mathbf{h}}^{s_t}_{t-1}, \tag{7}$$

where for a given value of s_t we sample s_{t-1} from $p(s_{t-1}|s_t)$ to compute a binary indicator $\tilde{\pi}^{s_t}_{t-1} = [\![s_{t-1} = t-1]\!]$, which signals whether the resulting $\tilde{\mathbf{h}}^{s_t}_t$ is equal to $\tilde{\mathbf{h}}^{s_t}_{t-1}$ or $f^{s_t}_{t-1}(\tilde{\mathbf{h}}^{t-1}_{t-1})$.

Using Eq. (7) we iterative sample from distributions $p(\mathbf{H}_t|s_t, \mathbf{H}_0)$ for $t = 1, \ldots, T$, and for each t we compute samples for $s_t = t, \ldots, T$. An illustration of the algorithm is shown in Figure 4. The computational complexity of a complete pass in this iterative process is $O(T(T+1)/2)$, since for each $t = 1, \ldots, T$, we compute $T - t + 1$ samples, each of which is computed in $O(1)$ from the samples already computed for $t-1$. This is roughly equivalent to the cost of evaluating a single network with dense layer connectivity of depth T [20], which has a total of $T(T-1)/2$ connections implemented by the functions f^j_i.

Sampling outputs from networks of bounded depth. Using the described algorithm, $\tilde{\mathbf{h}}^T_T \sim p(\mathbf{H}_T|s_T = T, \mathbf{H}_0)$ correspond to output tensors $\mathbf{H}_L$ sampled from the mixture defined in Eq. (2). Moreover, for any t, samples from $p(\mathbf{H}_t|s_t = T, \mathbf{H}_0)$ are output feature maps generated by networks with depth bounded by t. For instance, in Figure 2, samples $\tilde{\mathbf{h}}^T_2$ are generated with one of the networks coded by the sequences 01444 and 04444.

3.3. Training and inference

We use ψ to collectively denote the parameters of the convolutional blocks f^j_i and the parameters $\pi^{s_t}_{s_{t-1}}$ defining the mixing weights via Eq. (5). Moreover, the parameters of the classifier that predict the image label(s) from the

output tensor $\mathbf{H}_T$ are denoted as θ. Given a training set $\mathcal{D} = \{(\mathbf{X}_1, y_1), \ldots, (\mathbf{X}_N, y_N)\}$ composed of images $\mathbf{X}_n$ and labels y_n, we optimize the parameters by minimizing

$$\mathcal{L}_{\text{single}}(\psi, \theta) = \sum_{n=1}^{N} \mathbb{E}_{p(\mathbf{H}_T|\mathbf{H}_0=\mathbf{X_n};\psi)} \Big[\mathcal{L}\left(y_n, \mathbf{H}_T, \theta\right) \Big], \quad (8)$$

where $\mathcal{L}\left(y_n, \mathbf{H}_T, \theta\right)$ is the cross-entropy loss comparing the label y_n with the class probabilities computed from $\mathbf{H}_T$. In practice, we replace the expectation over $\mathbf{H}_T$ in each training iteration with samples from $p(\mathbf{H}_T|\mathbf{H}_0 = \mathbf{X_n}; \psi)$.

Learning from subsets of networks. As discussed in Section 3.2, samples from the distribution $p(\mathbf{H}_t|s_t = T, \mathbf{H}_0)$ correspond to outputs of CNNs in the mixture with depth at most t. In order to improve performance of models with reduced inference time, we explicitly emphasize the loss for such efficient relatively shallow networks. Therefore, we sum the above loss function over the outputs sampled from networks of increasing depth:

$$\mathcal{L}(\theta, \psi) = \sum_{n=1}^{N} \sum_{t=1}^{T} \mathbb{E}_{p(\mathbf{H}_t|s_t=T,\mathbf{X_n};\psi)} \Big[\mathcal{L}\big(y_n, \mathbf{H}_t, \theta_t\big) \Big], \quad (9)$$

where we use a separate classifier for each t. In practice, we balance each loss with a weight increasing linearly with t.

Relaxed binary variables with concrete distributions. The recurrence in Eq. (7) requires sampling from $p(s_{t-1}|s_t)$, defined in Eq. (5). The sampling renders the parameters $\pi_{t-1}^{s_t}$ non-differentiable, which prevents gradient-based optimization for them. To address this limitation, we use a continuous relaxation by modelling $p(s_{t-1}|s_t)$ as a binary "concrete" distribution [42]. In this manner, we can use the re-parametrization trick [27, 48] to back-propagate gradients w.r.t. samples $\tilde{\pi}_{t-1}^{s_t}$ in Eq. (7) and, thus to compute gradients for the parameters $\pi_{t-1}^{s_t}$.

Efficient inference by expectation propagation. Once the CNMM is trained, the predictive distribution on y is given by $p(y|\mathbf{X}; \theta) = \mathbb{E}_{p(\mathbf{H}_T|\mathbf{X})}[p(y|\mathbf{H}_T; \theta)]$. The expectation is intractable to compute exactly, contrary to our goal of efficient inference. A naive Monte-Carlo sample approximation is still requires multiple evaluations of the full CNMM. Instead, we propose an alternative approximation by propagating expectations instead of samples in Eq. (7), *i.e.* using the approximation $\bar{\mathbf{H}}_T \approx p(\mathbf{H}_T|\mathbf{X})$, where $\bar{\mathbf{H}}_T$ is obtained by running the sampling algorithm replacing the samples $\tilde{\pi}_{t-1}^{s_t}$ with their expectations $\pi_{t-1}^{s_t}$.

3.4. Accelerating CNNMs

CNMMs offer two complementary mechanisms in order to accelerate inference. We describe both in the following.

Evaluating intermediate classifiers. The different classifiers θ_t learned by minimizing Eq. (9) operate over the outputs of a mixture of networks with maximum depth t. Therefore, at each iteration t of the inference algorithm in Eq. (7) we can already output predictions based on classifier θ_t. This strategy is related with the one employed in multi-scale dense networks (MSDNets) [17], where "early-exit" classifiers are used to provide predictions at various points in time during the inference process.

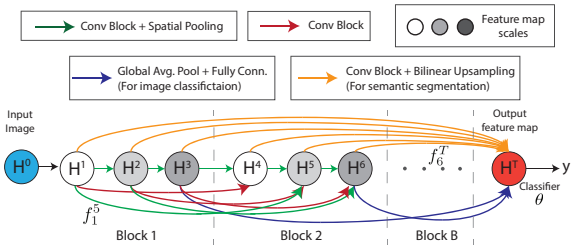

Figure 5. Dense network with sparse connectivity implementing the inference algorithm of our CNMMs. See text for details.

Network pruning. A complementary strategy to accelerate CNMMs is to remove networks from the mixture. The computational cost of the inference process is dominated by the evaluation of the CNN blocks $f_{t-1}^{s_t}(\tilde{\mathbf{h}}_{t-1}^{t-1})$ in Eq. (7). However, these function does not need to be computed when the variable $\tilde{\pi}_{t-1}^{s_t} = 0$. Therefore, a natural approach to prune CNMMs is to set certain $\pi_{t-1}^{s_t}$ to zero, removing all the CNNs from the mixture that use $f_{t-1}^{s_t}$. We use the learned distribution $p(s_{0:T})$ in order to remove networks with a low probability. Note that for a given value of s_t, the pairwise marginal $p(s_t, s_{t-1} = t-1)$ is exactly the sum of probabilities of all the networks involving the function $f_{t-1}^{s_t}$. Using this observation, we use an iterative pruning algorithm where, at each step, we compute all pairwise marginals $p(s_t, s_{t-1} = t-1)$ for all possible values of s_t and t. We then set $\pi_{t^\star-1}^{s_t^\star} = 0$ where $(s_t^\star, t^\star) = \arg\min_{(s_t,t)} p(s_t, s_{t-1} = t-1)$. Finally, the marginals are updated, and we iterate.

In this manner, we achieve different pruning levels by progressively removing convolutional blocks that will not be evaluated during inference. This process does not require any re-training of the model, allowing to dynamically set different pruning ratios. Note that this process is complementary to the use of intermediate classifiers, as discussed above. The reason for this is that our pruning strategy may be used to remove functions f_i^j for any "early" prediction step $t < T$. Finally, it is interesting observe that the proposed pruning mechanism can be regarded as a form of neural architecture search [37, 63], where the optimal network connectivity for a given pruning ratio is automatically discovered by taking into account the learned probabilities $p(s_{t-1} = t-1|s_t)$.

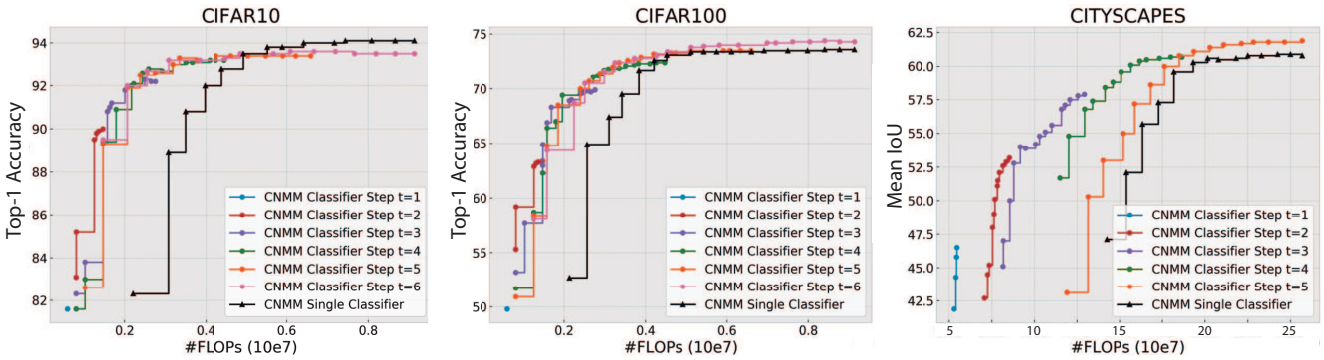

Figure 6. Prediction accuracy *vs*. FLOPs for accelerated CNMMs. Black curves depict the performance of a CNMM learned using a single final classifier. Colored curves correspond to intermediate classifiers at different steps of the inference algorithm. Points on one curve are obtained by progressively pruning convolutional layers.

4. Experiments

We perform experiments over two different tasks: image classification and semantic segmentation. Following previous work, we measure the computational cost in terms of the number of floating point multiply and addition operations (FLOPs) required for inference. The number of FLOPs provides a metric that correlates very well with the actual inference wall-time, while being independent of implementation and hardware used for evaluation.

4.1. Datasets and experimental setup

CIFAR-10/100 datasets. These datasets [29] are composed of 50k train and 10k test images with a resolution of 32×32 pixels. The goal is to classify each image across 10 or 100 classes, respectively. Images are normalized using the means and standard deviations of the RGB channels. We apply standard data augmentation operations: (i) a 4-pixel zeros padding followed by 32×32 cropping. (ii) Random horizontal flipping with probability 0.5. Performance is evaluated in terms of the mean accuracy across classes.

CityScapes dataset. This dataset [6] contains 1024×2048 pixel images of urban scenes with pixel-level labels across 19 classes. The dataset is split into training, validation and test sets with 2,975, 500 and 1,525 samples each. The ground-truth annotations for the test set are not public, and we use the validation set instead for evaluation. To assess performance we use the standard mean intersection-over-union (mIoU) metric. We follow the setup of [45], and down-sample the images by a factor two before processing them. As a data augmentation strategy during training, we apply random horizontal flipping and resizing by using a scaling factor between 0.75 and 1.1. Finally, we use random crops of 384×768 pixels from the down-sampled images.

Base architecture. As discussed in Section 3.2, the learning and inference algorithms for CNMMs can be implemented using a network with dense layer connectivity [20]. Based on this observation, we use an architecture similar to MSDNets [17]. Specifically. we define a set of B blocks, each composed of a set of S feature maps $\mathbf{H}_t$. See Fig. (5).

The initial feature map in each block has C channels and, at each subsequent feature map in the block, the spatial resolution is reduced by a factor two in each dimension, and the number of channels is doubled. Feature maps are connected by functions f_i^j if the output feature map $\mathbf{H}_j$ has the same or half the resolution of the input feature map $\mathbf{H}_i$. Finally, we consider the output tensor $\mathbf{H}_T$ to have different connectivity and spatial resolution depending on the task.

Implementation for image classification. We implement the convolutional layers as the set of operations (BN-ReLU-DConv-BN-ReLU-Conv-BN), where BN refers to batch normalization, DConv is a $(3 \times 3 \times C \times \frac{C}{4})$ depth-wise separable convolution [16], and Conv is a $(1 \times 1 \times \frac{C}{4} \times C)$ convolution. In order to reduce computation, for a given tensor $\mathbf{H}_i$, the different functions f_i^j share the parameters of the initial operations (BN-ReLU-DConv) for all j. Moreover, when the resolution of the feature map is reduced, we use average pooling after these three initial operations. In all our experiments, the number of initial channels in $\mathbf{H}_1$ is set to $C = 64$. This is achieved by using a $(3 \times 3 \times 3 \times \frac{C}{4})$ convolution over the input image $\mathbf{H}_0$ and then apply a $(1 \times 1 \times \frac{C}{4} \times C)$ convolutional block. Finally, all the tensors $\mathbf{H}_l$ with the lowest spatial resolution are connected to the output $\mathbf{H}_L$. Concretely, $\mathbf{H}_T$ is a vector $\mathbb{R}^{512}$ obtained by applying the operations (BN-ReLU-GP-FC-BN) to the input tensors, where GP refers to global average pooling, and FC corresponds to a fully-connected layer. The classifier θ maps $\mathbf{H}_T$ linearly to a vector of dimension equal to the number of classes. When using Eq. (9) to train the CNMM, we connect a classifier θ_t with the end of each block.

Implementation for semantic segmentation. We use the same setup as for image classification, but replace the ReLU activations with parametric ReLUs as in [44]. Moreover, we use max instead of average pooling to reduce the spa-

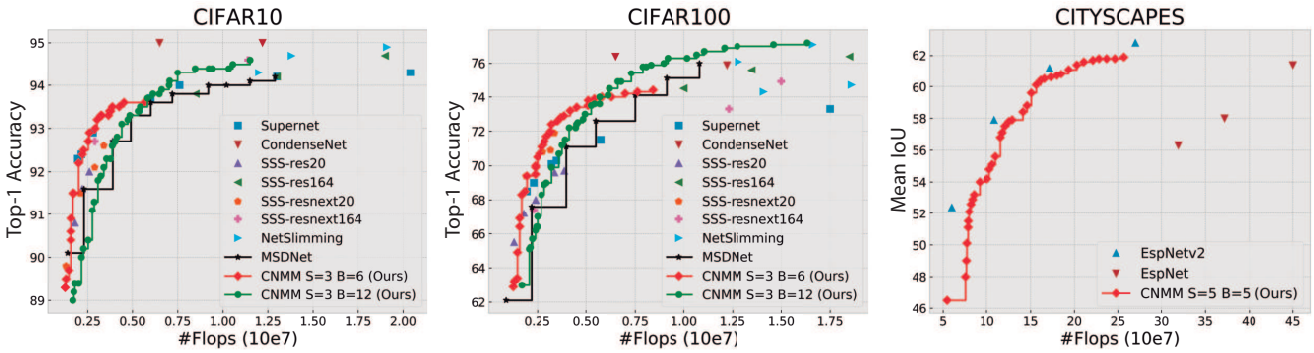

Figure 7. Comparison of the our CNMM with state-of-the-art efficient inference approaches on the CIFAR and CityScapes datasets. Disconnected markers refer to models that are trained independently. Curves correspond to a single model that can operate at different number of FLOPs. CNMM curves are obtained by using the optimal combination of pruning and intermediate classifiers.

tial resolution. The input tensor $\mathbf{H}_1$ has $C = 48$ channels and a resolution four times lower than the original images. This is achieved by applying a $(3 \times 3 \times 3 \times \frac{C}{4})$ convolution with stride 2 to the input and then using a (BN-ReLU-Conv) block followed by max pooling. The output tensor $\mathbf{H}_L$ receives connections from all the previous feature maps and has the same channels and spatial resolution as $\mathbf{H}_1$. Given that the input feature maps are at different scales, we apply a (BN-PReLU-Conv-BN) block over the input tensor and use bi-linear up-sampling with different scaling factors in order to recover the original spatial resolution. The final classifier θ computing the class probabilities using $\mathbf{H}_T$ are defined as blocks (UP-BN-PReLU-Conv-UP-BN-PReLU-DConv), where UP refers to bilinear upsampling, which allows to recover the original image resolution. The first and second convolutions in the block have 12 and K output channels respectively, where K is the number of classes. As in image classification, we use an intermediate classifier θ_t at each step t where a full block of computation is finished.

Optimization details. In our experiments, we use SGD with momentum by setting the initial learning rate to 0.1 and weight decay to 10^{-4}. In CIFAR, we use a cosine annealing schedule to accelerate convergence. On the other hand, in Cityscapes we employ a cyclic schedule with warm restarts as in [45]. The temperature of the concrete distributions modelling $p(s_{t-1}|s_t)$ is set to 2. We train our model by using 300 and 200 epochs, and batch size of 64 and 6, for respectively CIFAR-10/100 and Cityscapes. For the cyclic scheduler, the learning rate is divided by two at epochs $\{30, 60, 90, 120, 150, 170, 190\}$. Additionally, the models trained in Cityscapes are fine-tuned during 10 epochs by using random crops of size 512×1024 instead of 384×768.

4.2. Pruning and intermediate classifiers

We evaluate the proposed pruning and intermediate classifiers strategies to reduce the inference time of trained CNMMs. For CIFAR-10/100 we learn a CNMM with $B = 6$ blocks, using $S = 3$ scales each. For Cityscapes we use $B = 5$ blocks and $S = 5$ scales. For each dataset, we train one model that uses a single classifier θ, optimized using Eq. (8). In addition, we train a second model with intermediate classifiers θ_t, minimizing the loss function in Eq. (9). In the following, we will refer to the first and second variant as CNMM-single and CNMM respectively.

In Figure 6 we report prediction accuracy *vs*. FLOPs for inference. Each model is represented as a curve, traced by pruning the model to various degrees. Across the three datasets, the CNMM model with intermediate classifiers achieves higher accuracy in fast inference settings than the CNMM-single model. Recall that all the operation point across the different CNMM curves are obtained from a single trained model. Therefore, this single model can realize the upper-envelope of the performance curves. As expected, the maximum performance of the intermediate classifiers increases with the step number. The accuracy of CNMM at the final step is comparable to the level obtained by the CNMM-single model: slightly worse on CIFAR-10, and slightly better at CIFAR-100 and CityScapes. This is because the minimized intermediate losses provide additional supervisory signal which is particularly useful to encourage accurate prediction for shallow, but fast, CNNs. In conclusion, the CNMM model with intemediate classifiers is to be preferred, since it provides a better trade-off between accuracy and computation at a wider range of FLOP counts.

By analysing the operating points along each curve, we can observe the effectiveness of the proposed pruning algorithm. For the CIFAR datasets we can reduce the FLOP count by a factor two without significant loss in accuracy. For CityScapes, about 25% pruning can be achieved without a significant loss. In general, if several exit points can achieve the same FLOP count by applying varying amounts of pruning, best performance is obtained pruning less for an earlier classifier, rather than pruning more for a later exit.

4.3. Comparison with the state of the art

Image classification. We compare our model with dif-

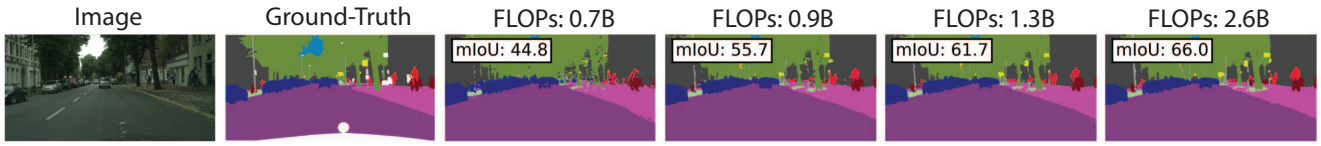

Figure 8. Pixel-level predictions for a single CNMM operating under different computational constraints. As discussed, our model allows to dynamically dynamically set the trade-off between accuracy and inference time with no additional cost.

ferent state-of-the-art CNN acceleration strategies [17, 19, 22, 38, 56]. We consider methods applying pruning at different levels, such as independent filters (Network slimming [38]), groups of weights (CondenseNet) [19], connections in multi-branch architectures (SuperNet) [56], or a combination of them (SSS) [22]. We also compare our method with any-time models based on early-exit classifiers (MSDNet) [17]. Among other previous state-of-the-art methods, the compared approaches have shown the best performance among efficient inference methods with $\leq$200 million FLOPs. We compare to CNMMs using 6 and 12 blocks, using three scales is both cases.

The results in Figure 7 (left, center) show that CNMMs achieve similar or better accuracy-compute trade-off across a broad range of FLOP counts than all the compared methods in the CIFAR datasets. Only CondenseNets shows somewhat better performance for medium FLOP counts. Moreover, note that the different operating points shown for the compared methods (except for MSDNets) are obtained by using different models trained independently, *e.g.* by different settings of a hyper-parameter controlling the pruning ratio. In contrast, CNMM embeds a large number operating points in a single model. This feature is interesting when the available computational budget can change dynamically, based on concurrent processes, or when the model is deployed across a wide range of devices. In these scenarios, a single CNMM can be accelerated on-the-fly depending on the available resources. Note that a single MSDNet is also able to provide early-predictions by using intermediate classifiers. However, our CNMM provides better performance for a given FLOP count and allows for a finer granularity to control the computational cost.

Semantic segmentation. State-of-the-art methods for real-time semantic segmentation have mainly focused on the manual-design of efficient network architectures. By employing highly optimized convolutional modules, ESPNet [44] and ESPNetv2 [45] have achieved impressive accuracy-computation trade-offs. Other methods, such as [5, 59], offer higher accuracy but at several orders of magnitude higher inference cost, limiting their application in resource constrained scenarios.

In Figure 7 (right) we compare our CNMM results to these two approaches. Note that the original results reported in [45] are obtained by using a model pre-trained in ImageNet. For a fair comparison with our CNMMs, we have trained EspNetv2 from scratch by using the code provided by the authors [2]. As can be observed, CNMM provides a better trade-off compared to ESPNet. In particular, a full CNMM without pruning obtains an improvement of 0.5 points of mIoU, while reducing the FLOP count by 45%. Moreover, an accelerated CNMM achieves a similar performance compared to the most efficient ESPNet that needs more than two times more FLOPs. On the other hand, ESPNetv2 gives slightly better trade-offs compared to our CNMMs. However, this model relies on an efficient inception-like module [53] that also includes group point-wise and dilated convolutions. These are orthogonal design choices that can be integrated in our model as well, and we expect that to further improve our results. Additionally, the different operating points in ESPNet and ESPNetv2 are achieved using different models trained independently. Therefore, unlike our approach, these methods do not allow for a fine-grained control over the accuracy-computation trade-off, and multiple models need to be trained. Figure 8 shows qualitative results using different operating points from a single CNMM.

5. Conclusions

We proposed to address model pruning by using Convolutional Neural Mixture Models (CNMMs), a novel probabilistic framework that embeds a mixture of an exponential number of CNNs. In order to make training and inference tractable, we rely on massive parameter sharing across the models, and use concrete distributions to differentiate across the discrete sampling of mixture components. To achieve efficient inference in CNMM we use an early-exit mechanism that allows prediction after evaluating only a subset of the networks. In addition, we use a pruning algorithm to remove CNNs that have low mixing probabilities. Our experiments on image classification and semantic segmentation tasks show that CNMMs achieve excellent trade-offs between prediction accuracy and computational cost. Unlike most of previous works, a single CNMM model allows for a large number and wide range of accuracy-compute trade-offs, without any re-training.

Acknowledgements

This work is supported by ANR grants ANR-16-CE23-0006 and ANR-11-LABX-0025-01.

[2] https://github.com/sacmehta/EdgeNets

References

[1] Karim Ahmed and Lorenzo Torresani. Maskconnect: Connectivity learning by gradient descent. In *ECCV*, 2018.

[2] Pierre Baldi and Peter J Sadowski. Understanding dropout. In *NeurIPS*, 2013.

[3] Leo Breiman. Bagging predictors. *Machine learning*, 1996.

[4] Guobin Chen, Wongun Choi, Xiang Yu, Tony Han, and Manmohan Chandraker. Learning efficient object detection models with knowledge distillation. In *NeurIPS*, 2017.

[5] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *PAMI*, 2018.

[6] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *CVPR*, 2016.

[7] Simon S Du, Xiyu Zhai, Barnabas Poczos, and Aarti Singh. Gradient descent provably optimizes over-parameterized neural networks. *ICLR*, 2019.

[8] Anuvabh Dutt, Denis Pellerin, and Georges Quénot. Coupled ensembles of neural networks. In *2018 International Conference on Content-Based Multimedia Indexing (CBMI)*, 2018.

[9] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Training pruned neural networks. *ICLR*, 2018.

[10] Yarin Gal, Jiri Hron, and Alex Kendall. Concrete dropout. In *NeurIPS*, 2017.

[11] Soumya Ghosh, Jiayu Yao, and Finale Doshi-Velez. Structured variational learning of bayesian neural networks with horseshoe priors. *ICML*, 2018.

[12] Bohyung Han, Jack Sim, and Hartwig Adam. Branchout: Regularization for online ensemble tracking with convolutional neural networks. In *CVPR*, 2017.

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *ECCV*, 2016.

[14] Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. AMC: AutoML for model compression and acceleration on mobile devices. In *ECCV*, 2018.

[15] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *NIPS Deep Learning and Representation Learning Workshop*, 2015.

[16] A. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[17] G. Huang, D. Chen, T. Li, F. Wu, L. van der Maaten, and K. Weinberger. Multi-scale dense convolutional networks for efficient prediction. *ICLR*, 2018.

[18] Gao Huang, Yixuan Li, Geoff Pleiss, Zhuang Liu, John E Hopcroft, and Kilian Q Weinberger. Snapshot ensembles: Train 1, get m for free. *ICLR*, 2017.

[19] G. Huang, S. Liu, L. van der Maaten, and K. Weinberger. Condensenet: An efficient densenet using learned group convolutions. In *CVPR*, 2018.

[20] G. Huang, Z. Liu, L. van der Maaten, and K. Weinberger. Densely connected convolutional networks. In *CVPR*, 2017.

[21] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q Weinberger. Deep networks with stochastic depth. In *ECCV*, 2016.

[22] Zehao Huang and Naiyan Wang. Data-driven sparse structure selection for deep neural networks. In *ECCV*, 2018.

[23] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

[24] Eddy Ilg, Ozgun Cicek, Silvio Galesso, Aaron Klein, Osama Makansi, Frank Hutter, and Thomas Brox. Uncertainty estimates and multi-hypotheses networks for optical flow. In *ECCV*, 2018.

[25] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *CVPR*, 2018.

[26] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. Speeding up convolutional neural networks with low rank expansions. *BMVC*, 2014.

[27] D. Kingma and M. Welling. Auto-encoding variational Bayes. In *ICLR*, 2014.

[28] Durk P Kingma, Tim Salimans, and Max Welling. Variational dropout and the local reparameterization trick. In *NeurIPS*, 2015.

[29] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

[30] Anders Krogh and Jesper Vedelsby. Neural network ensembles, cross validation, and active learning. In *NeurIPS*, 1995.

[31] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In *NeurIPS*, 2017.

[32] Gustav Larsson, Michael Maire, and Gregory Shakhnarovich. Fractalnet: Ultra-deep neural networks without residuals. *ICLR*, 2017.

[33] Y. LeCun, J. Denker, and S. Solla. Optimal brain damage. In *NeurIPS*, 1990.

[34] Stefan Lee, Senthil Purushwalkam, Michael Cogswell, David Crandall, and Dhruv Batra. Why m heads are better than one: Training a diverse ensemble of deep networks. *arXiv preprint arXiv:1511.06314*, 2015.

[35] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *ICLR*, 2017.

[36] Darryl Lin, Sachin Talathi, and Sreekanth Annapureddy. Fixed point quantization of deep convolutional networks. In *ICML*, 2016.

[37] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *ICLR*, 2019.

[38] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In *ICCV*, 2017.

[39] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *ICLR*, 2017.

[40] Christos Louizos and Max Welling. Multiplicative normalizing flows for variational bayesian neural networks. In *ICML*, 2017.

[41] Christos Louizos, Max Welling, and Diederik P Kingma. Learning sparse neural networks through l_0 regularization. *ICLR*, 2018.

[42] Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. *ICLR*, 2017.

[43] Marc Masana, Joost van de Weijer, Luis Herranz, Andrew D Bagdanov, and Jose M Alvarez. Domain-adaptive deep network compression. In *ICCV*, 2017.

[44] Sachin Mehta, Mohammad Rastegari, Anat Caspi, Linda Shapiro, and Hannaneh Hajishirzi. Espnet: Efficient spatial pyramid of dilated convolutions for semantic segmentation. In *ECCV*, 2018.

[45] Sachin Mehta, Mohammad Rastegari, Linda Shapiro, and Hannaneh Hajishirzi. Espnetv2: A light-weight, power efficient, and general purpose convolutional neural network. *CVPR*, 2019.

[46] Ury Naftaly, Nathan Intrator, and David Horn. Optimal ensemble averaging of neural networks. *Network: Computation in Neural Systems*, 1997.

[47] Kirill Neklyudov, Dmitry Molchanov, Arsenii Ashukha, and Dmitry P Vetrov. Structured bayesian pruning via log-normal multiplicative noise. In *NeurIPS*, 2017.

[48] D. Rezende, S. Mohamed, and D. Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *ICML*, 2014.

[49] Robert E Schapire. The boosting approach to machine learning: An overview. In *Nonlinear estimation and classification*. 2003.

[50] Saurabh Singh, Derek Hoiem, and David Forsyth. Swapout: Learning an ensemble of deep architectures. In *NeurIPS*, 2016.

[51] Suraj Srinivas and R Venkatesh Babu. Data-free parameter pruning for deep neural networks. *BMVC*, 2015.

[52] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *JMLR*, 2014.

[53] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *CVPR*, 2016.

[54] Cheng Tai, Tong Xiao, Yi Zhang, Xiaogang Wang, et al. Convolutional neural networks with low-rank regularization. *ICLR*, 2016.

[55] Andreas Veit, Michael J Wilber, and Serge Belongie. Residual networks behave like ensembles of relatively shallow networks. In *NeurIPS*, 2016.

[56] Tom Véniat and Ludovic Denoyer. Learning time/memory-efficient deep architectures with budgeted super networks. In *CVPR*, 2018.

[57] Jiahui Yu, Linjie Yang, Ning Xu, Jianchao Yang, and Thomas Huang. Slimmable neural networks. *ICLR*, 2019.

[58] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *CVPR*, 2018.

[59] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. Pyramid scene parsing network. In *CVPR*, 2017.

[60] Kaikai Zhao, Tetsu Matsukawa, and Einoshin Suzuki. Retraining: A simple way to improve the ensemble accuracy of deep neural networks for image classification. In *ICPR*. IEEE, 2018.

[61] Zhi-Hua Zhou, Jianxin Wu, and Wei Tang. Ensembling neural networks: many could be better than all. *Artificial intelligence*, 2002.

[62] Shilin Zhu, Xin Dong, and Hao Su. Binary ensemble neural network: More bits per network or more networks per bit? *CVPR*, 2019.

[63] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *CVPR*, 2018.