Adaptative Inference Cost With Convolutional Neural Mixture Models
Adrià Ruiz, Jakob Verbeek

To cite this version:

HAL Id: hal-02267564
https://hal.science/hal-02267564
Submitted on 19 Aug 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
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 suboptimal 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].

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
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.
3. Convolutional Neural Mixture Models

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

$$F(H_0) = f^T_0 \cdots f^1_0(H_0)), \quad (1)$$

where $H_t = f^t_{t-1}(H_{t-1})$ is computed from the previous feature map $H_{t-1}$. We assume that the functions $f^t_{i-1}$ 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 $f^t_{i-1}$, is at most $T$.

The output tensor $H_T$ of the CNN is used to make predictions for a specific task. For example, in image classification, a linear classifier over $H_T$ 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 $H_T$.

Given these definitions, a convolutional neural mixture model (CNMM) defines a distribution over output $H_T$ as:

$$p(H_T|H_0) = \sum_{F \in \mathcal{F}} p(F)p(H_T|H_0, F), \quad (2)$$

where $\mathcal{F} = \{F_1, F_2, \ldots, F_K\}$ is a finite set of CNNs, $p(H_T|H_0, F)$ is a delta function centered on the output $F_k(H_0)$ of each network, and $p(F)$ defines the mixing weights over the CNNs in $\mathcal{F}$.

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:

$$F(H_0) = f^{s_T}_{s_T-1} \cdots (f^{s_1}_{0}(f^0_0(H_0))), \quad (3)$$

For $i < j$ the function $f^i_j$ is a convolutional block as described above with its own parameters, while the functions $f^0_i$ 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.\footnote{In particular, this constraint ensures that, e.g., the network $f^2_1(f^1_0(H_0))$ is uniquely encoded by the sequence ‘01444’, ruling out the alternative sequences ‘01144’ and ‘01114’. See Figure 2 (Left).} 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 $p(F)$ for each network in the mixture, we define a distribution over sequences $s_{0:T}$ as a reversed Markov chain:

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

To ensure that sequences have positive probability if and only if they are valid, i.e. satisfy the constraints defined
above, we set \( p(s_T = T) = p(s_0 = 0 | s_1) = 1 \) and define:

\[
p(s_{t-1} | s_t) = \begin{cases} 
\pi_{s_{t-1}}^{s_t} & \text{if } s_{t-1} = (t - 1), \\
1 - \pi_{s_{t-1}}^{s_t} & \text{if } s_{t-1} = s_t, \\
0 & \text{otherwise.}
\end{cases} 
\]

(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 \( \pi_{s_{t-1}}^{s_t} \), 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 \( p(H_T | H_0) \) 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 \( p(H_t | s_t = l, H_0) \) gives the distribution over \( H_t \) across the networks with \( s_t = l \). For example, \( p(H_2 | s_2 = 4, H_0) \) consists of two weighted delta peaks, located at \( f_0^4(H_0) \) and \( f_0^0(f_0^4(H_0)) \), respectively. See Figure 2 (Left). These conditional distributions can be expressed as the forwards recurrence:

\[
p(H_t | s_t, H_0) = \sum_{H_{t-1}, s_{t-1}} p(H_t | H_{t-1}, s_t, s_{t-1}) p(s_{t-1} | s_t) 
\times p(H_{t-1} | s_{t-1}, H_0),
\]

(6)

where \( p(H_t | H_{t-1}, s_t, s_{t-1}) \) is a delta function centered on \( f_{s_{t-1}}^{s_t}(H_{t-1}) \). Therefore, unbiased samples \( \hat{h}_{t}^{s_t} \) from \( p(H_t | s_t, H_0) \) 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 \( s_{t-1} \) in Eq. (6) only consists of two terms. Given this observation, samples \( \hat{h}_{t}^{s_t} \sim p(H_t | s_t, H_0) \) can be obtained from samples \( \hat{h}_{t-1}^{s_t} \sim p(H_{t-1} | s_{t-1}, H_0) \) using Eq. (7) for \( t = 1, \ldots, T \).

\[
\hat{h}_{t}^{s_t} \sim p(H_{t-1} | s_{t-1}, H_0) \]

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 \( \tilde{\pi}_{s_{t-1}}^{s_t} = [s_{t-1} = t - 1] \), which signals whether the resulting \( \hat{h}_{t}^{s_t} \) is equal to \( \hat{h}_{t-1}^{s_t} \) or \( f_{s_{t-1}}^{s_t}(\hat{h}_{t-1}^{s_t}) \).

Using Eq. (7) we iterative sample from distributions \( p(H_t | s_t, H_0) \) 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 \( f_t^{s_t} \).

**Sampling outputs from networks of bounded depth.** Using the described algorithm, \( \hat{h}_{T}^{s_t} \sim p(H_T | s_T = T, H_0) \) correspond to output tensors \( H_L \) sampled from the mixture defined in Eq. (2). Moreover, for any \( t \), samples from \( p(H_t | s_T = T, H_0) \) are output feature maps generated by networks with depth bounded by \( t \). For instance, in Figure 2, samples \( \hat{h}_{2}^{s_t} \) are generated with one of the networks coded by the sequences 01444 and 04444.

3.3. Training and inference

We use \( \psi \) to collectively denote the parameters of the convolutional blocks \( f_t^{s_t} \) and the parameters \( \pi_{s_{t-1}}^{s_t} \) defining the mixing weights via Eq. (5). Moreover, the parameters of the classifier that predict the image label(s) from the
output tensor $H_T$ are denoted as $\theta$. Given a training set $D = \{ (X_1, y_1), \ldots, (X_N, y_N) \}$ composed of images $X_n$ and labels $y_n$, we optimize the parameters by minimizing

$$
\mathcal{L}_{\text{single}}(\psi, \theta) = \sum_{n=1}^{N} \mathbb{E}_{p(H_T|H_0=X_n; \psi)} \left[ \mathcal{L}(y_n, H_T, \theta) \right],
$$

(8)

where $\mathcal{L}(y_n, H_T, \theta)$ is the cross-entropy loss comparing the label $y_n$ with the class probabilities computed from $H_T$. In practice, we replace the expectation over $H_T$ in each training iteration with samples from $p(H_T|H_0=X_n; \psi)$.

**Learning from subsets of networks.** As discussed in Section 3.2, samples from the distribution $p(H_t|s_t = T, H_0)$ 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:

$$
\mathcal{L}(\theta, \psi) = \sum_{n=1}^{N} \sum_{t=1}^{T} \mathbb{E}_{p(H_t|s_t=T, X_n; \psi)} \left[ \mathcal{L}(y_n, H_t, \theta) \right],
$$

(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 $\pi_{t-1}^{s_t}$ 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 $p(y|X, \theta) = \mathbb{E}_{p(H_T|X)}[p(y|H_T; \theta)]$. The expectation is intractable to compute exactly, contrary to our goal of efficient inference. A naive Monte-Carlo sample approximation is still required 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 $H_T \approx p(H_T|X)$, where $H_T$ is obtained by running the sampling algorithm replacing the samples $\pi_{t-1}^{s_t}$ with their expectations $\tilde{\pi}_{t-1}^{s_t}$.

### 3.4. Accelerating CNMMs

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

**Evaluating intermediate classifiers.** The different classifiers $\theta_i$ 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 $\theta_i$. 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.

**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 $f_{t+1}^{s_t}(H_{t-1}^{s_t})$ 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 $p(s_t|T)$ 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 $\pi_{t-1}^{s_t} = 0$ where $(s_t, T^*) = \arg\min_{s_t, T} p(s_t, s_{t-1} = T - 1)$. 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 $f_t^s$ 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 $p(s_{t-1} = T - 1|s_t)$.
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 test sets with 2,975, 500 and 1,525 samples each. 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 the number of classes is doubled. Feature maps are connected by functions $f_j$ if the output feature map $H_j$ has the same or half the resolution of the input feature map $H_i$. Finally, we consider the output tensor $H_T$ 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 $(3 \times 3 \times C \times \frac{C}{T})$ depth-wise separable convolution [16], and Conv is a $(1 \times 1 \times \frac{C}{T} \times C)$ convolution. In order to reduce computation, for a given tensor $H_j$, the different functions $f_j^i$ 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 $H_j$ is set to $C = 64$. This is achieved by using a $(3 \times 3 \times 3 \times \frac{C}{T})$ convolution over the input image $H_0$ and then apply a $(1 \times 1 \times \frac{C}{T} \times C)$ convolutional block. Finally, all the tensors $H_j$ with the lowest spatial resolution are connected to the output $H_L$. Concretely, $H_T$ 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 $\theta$ maps $H_T$ linearly to a vector of dimension equal to the number of classes. When using Eq. (9) to train the CNMM, we connect a classifier $\theta_t$ 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-
temporal resolution. The input tensor $H_1$ has $C = 48$ channels and a resolution four times lower than the original images. This is achieved by applying a $(3 \times 3 \times 3 \times \frac{C}{s})$ convolution with stride 2 to the input and then using a (BN-ReLU-Conv) block followed by max pooling. The output tensor $H_t$ receives connections from all the previous feature maps and has the same channels and spatial resolution as $H_1$. Given that the input feature maps are at different scales, we apply a (BN-ReLU-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 $\theta$ computing the class probabilities using $H_T$ is defined as blocks (UP-BN-PReLU-Conv-UP-BN-PReLU-DConv), where UP refers to bilinear up-sampling, 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 $\theta_t$ 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 $512 \times 1024$ instead of $384 \times 768$.

### 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 $\theta$, optimized using Eq. (8). In addition, we train a second model with intermediate classifiers $\theta_t$, 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 intermediate 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-
different state-of-the-art CNN acceleration strategies \cite{17, 19, 22, 38, 56}. We consider methods applying pruning at different levels, such as independent filters (Network slimming \cite{38}), groups of weights (CondenseNet) \cite{19}, connections in multi-branch architectures (SuperNet) \cite{56}, or a combination of them (SSS) \cite{22}. We also compare our method with any-time models based on early-exit classifiers (MSDNets) \cite{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 in 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 \cite{44} and ESPNetv2 \cite{45} have achieved impressive accuracy-computation trade-offs. Other methods, such as \cite{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 \cite{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 \cite{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

\(^2\)https://github.com/sacmehta/EdgeNets
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.

References


A. Supplementary material

We provide further results on CIFAR100 in order to show the importance of all components of our proposed CNMMs. Moreover, we provide additional qualitative results of semantic segmentation on the CityScapes dataset.

A.1. Ablative study of CNMM

Using sampling during training. During learning, CNMMs generate a set of samples \( \tilde{\pi}_{s_t-1} \) using Eq. (7). In contrast, during inference we use the expectations \( \pi_{s_t-1} \) instead. In order to evaluate the importance of sampling during learning, we have optimized a CNMM by using the aforementioned expectations instead of samples. Figure 9 shows the results obtained by the model using this approach, denoted as “Training with expectations”. We observe that, compared to the CNMM using sampling, the accuracy decreases faster when different pruning ratios are applied. We attribute this to the fact that our sampling procedure can be regarded as a continuous-relaxation of dropout, where a subset of functions \( f_{s_t-1}^s(h_{t-1}^s) \) are randomly removed when computing the output tensor \( h_{s_t-1}^s \). As a consequence, the learned model is more robust to the pruning process where some of the convolutional blocks are removed during inference. This is not the case when deterministic expectations are used in Eq. (7) rather than samples.

Comparison with a deterministic model. We compare the performance of our CNMM with a deterministic variant using the same architecture. Concretely, in Eq. (7) we ignore \( \tilde{\pi}_{s_t-1} \) and simply sum the feature maps \( \tilde{h}_{s_t-1}^s \) and \( \tilde{h}_{t-1}^s \). Note that the resulting model is analogous to a MSDNet [17] using early-exit classifiers. We report the results in Figure 9, denoted as “Deterministic with early-exits”. We observe that our CNMM model obtains better performance than its deterministic counterpart. Moreover, same as MSDNets, accelerating the deterministic model is only possible by using the early-exits. In contrast, the complementary pruning algorithm available in CNMM allows for a finer granularity to control the computational cost.

Expectation approximation during inference. In order to validate our approximation of \( p(y|X; \theta) = E_{p(H_T|X)}[p(y|H_T; \theta)] \) during inference, we evaluate the performance obtained by using a Monte-Carlo procedure for the same purpose. In particular, we generate \( N \) samples from the output distribution \( p(H_T|H_0) \). Then, we compute the class probabilities \( p(y|H_T; \theta) \) for each sample and average them. Table 1 shows the results obtained by varying the number of samples. We observe that our approach offers a similar performance as the Monte-Carlo approximation using \( N = 5 \). For a higher number of samples, we observe slight improvements in the results. However, note that a Monte-Carlo approximation is very inefficient since it requires \( N \) independent evaluations of the model.

<table>
<thead>
<tr>
<th>Approximation</th>
<th>FLOPs</th>
<th>Top-1 Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expectation (used)</td>
<td>93M</td>
<td>74.4</td>
</tr>
<tr>
<td>Sampling N=1</td>
<td>93M</td>
<td>71.2</td>
</tr>
<tr>
<td>Sampling N=5</td>
<td>463M</td>
<td>74.4</td>
</tr>
<tr>
<td>Sampling N=15</td>
<td>1390M</td>
<td>74.5</td>
</tr>
<tr>
<td>Sampling N=30</td>
<td>2780M</td>
<td>74.6</td>
</tr>
</tbody>
</table>

Table 1. Comparison of the results obtained in CIFAR100 by approximating the CNMM output using our approach or a Monte-Carlo procedure with different number of samples \( N \).

A.2. Additional Qualitative Results

In Figure 10 we provide additional qualitative results for semantic segmentation obtained by a single trained CNMM model, using various operating points with different number of FLOPs during inference.

In particular, the last row in Table 1 is 30 times more costly to obtain than the two first rows. The minimal gain obtained with more samples could probably be more efficiently obtained by using a larger model.
Figure 10. Pixel-level predictions for a single CNMM adapting the number of FLOPs required during inference.