Distillation-Based Training for Multi-Exit Architectures

Mary Phuong, Christoph H. Lampert
IST Austria
Am Campus 1, Klosterneuburg, Austria
{bphuong, chl}@ist.ac.at

Abstract

Multi-exit architectures, *in which a stack of processing layers is interleaved with early output layers, allow the processing of a test example to stop early and thus save computation time and/or energy. In this work, we propose a new training procedure for multi-exit architectures based on the principle of* knowledge distillation. *The method encourages early exits to mimic later, more accurate exits, by matching their output probabilities.*

Experiments on CIFAR100 and ImageNet show that distillation-based training significantly improves the accuracy of early exits while maintaining state-of-the-art accuracy for late ones. The method is particularly beneficial when training data is limited and it allows a straightforward extension to semi-supervised learning, i.e. *making use of unlabeled data at training time. Moreover, it takes only a few lines to implement and incurs almost no computational overhead at training time, and none at all at test time.*

1. Introduction

Over the last years, convolutional networks for image classification have become progressively better, yet also bigger and slower. Today's models that achieve state-of-the-art performance on benchmarks, such as ImageNet, have many tens or even hundreds of layers and require billions floating point operations to classify a single image. Classifying a single image can take several seconds, unless fast hardware is available.

For many practical problems, however, execution speed is as important as classification accuracy. For example, on mobile devices, execution speed directly influences battery life and heat release. For robotics applications, such as self-driving cars, low latency is crucial for operating under real-time constraints. Therefore, there have recently also been improved efforts to create convolutional networks that are optimized for fast evaluation and energy efficiency even at the cost of sacrificing classification accuracy.

This work was in parts funded by the European Research Council under the European Unions Seventh Framework Programme (FP7/2007-2013)/ERC grant agreement no. 308036.

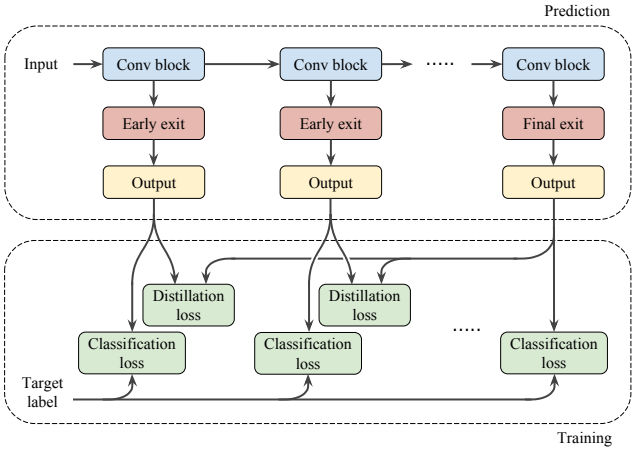

Figure 1: Illustration of the proposed method: distillation-based training (bottom) for a multi-exit a architecture (top).

What both trends have in common, though, is that once the models have been designed and trained, their accuracy and the amount of computation required for inference are fixed. Hence, these approaches are only applicable when the test-time inference budget is constant and known in advance, because only then can an appropriate architecture be chosen. When the time budget is unknown or varies over time, *e.g.* due to concurrently running jobs or a dynamic change of processor speed, any fixed model architecture is suboptimal: a model that is fast enough to run under all conditions yields suboptimal accuracy in situations where the available computational budget is actually higher than the worst case. A more accurate, but slower, model might fail to provide decisions at prediction time when the available budget falls below what the network needs to finish its computation.

In this work, we adopt a paradigm that overcomes the limitations described above: *anytime prediction*, that is, the

ability to trade off accuracy and computation at test time with a single model and on a per-example basis. A typical anytime prediction system quickly produces a crude initial prediction and then gradually improves it. At any time, a valid prediction for the given input is available to be used in case the time budget for the classification process runs out. Hence, anytime systems are more robust and flexible under uncertain or changing conditions and thereby overall more resource-efficient.

For the task of image classification with convolutional networks, anytime prediction can be realized via *multi-exit architectures*, in which a sequence of feature layers (*e.g.* convolutional) is augmented with early *exits* at different depths. These are standard classification layers acting on the feature representation that the network had computed up to this stage. The exits form a sequence of increasingly complex classifiers (see Figure 1, "Prediction" box), in which later layers reuse the representations, and thereby computations, of the earlier layers. To make a prediction, an input image is propagated through the stack of layers (left to right in Figure 1). When the process is interrupted, the model outputs either one of the already evaluated exits, or an ensemble of all of them.

Multi-exit architectures are typically trained with a *multi-task objective*: one attaches a loss function to each exit, for example cross-entropy, and minimizes the sum of exit-wise losses, as if each exit formed a separate classification task. On the one hand, this is a canonical choice: not knowing which exits will be used at prediction time, we want all of them to perform well, so we should train all of them for best classification quality. On the other hand, however, this choice ignores a lot of the prior knowledge we have about the anytime learning problem in general, and about the structure of the multi-exit architecture in particular. For example, while in multi-task learning, different classifiers could have different label sets and be trained on different data, in the multi-exit situation, all classifiers share the same label set and training data. Also, multi-task learning is known to work best if all classifiers are of comparable complexity and quality, such that none of the loss term dominates the others. In contrast, in the multi-exit case we know a priori that the classifiers from later exits have more capacity and should be more accurate than the ones from early exits, as this is in fact the main motivation for the anytime architecture.

Our main contribution in this paper is *a new objective for training multi-exit architectures based on knowledge distillation.* The resulting training method

1) leads to substantially improved classification accuracy, especially for early exits,
2) requires no change to the underlying architecture,
3) is conceptually simple and easy to implement,
4) opens up a natural way to make use also of unlabeled data when training multi-exit architectures.

The main working principle of *distillation-based training* for multi-exit architectures is the sharing of information between exits. Specifically, it does so in an asymmetric way: information flows from exits with high classification accuracy to those with lower accuracy.

Several aspects of the classification task are potentially beneficial to transfer. Take, for example, the information which training examples are easy, difficult, or even outliers. A classifier can benefit from this information, because if it tries too hard to correctly classify difficult examples, the result will be an inefficient assignment of its model capacity and potentially overfitting. Another aspect is the question of which classes are semantically similar to each other. Knowing this will allow a classifier to model them with similar feature representations, thereby resulting in a smoother decision function and, again, a lower risk of overfitting.

Neither which examples are easy or hard examples, nor which class are similar to each other can be extracted simply from the ground truth labels. Therefore, this information is not immediately available to the exits when trained using the multi-task objective. The information can be extracted, however, from the probabilistic outputs of a well-trained classifier: hard examples typically have a lower confidence score for the selected label than easy ones, and similar classes tend to be more co-activated in the outputs than dissimilar ones. We can transfer this information from late to early exits simply by encouraging early exits to mimic the probabilistic outputs of later exits. In practice, this is achieved by minimizing the cross-entropy between the outputs, with an additional temperature-scaling step that we detail in Section 3.5.

The temperature-scaled cross-entropy has been used in *knowledge distillation* to transfer information from one network to another. Therefore, we refer to this loss term as *distillation loss*. The overall system is illustrated in Figure 1 ("Training" box).

It has been observed [15] that distillation acts similarly to regularization, *i.e.* it is particularly useful when training large networks from relatively little labeled training data. The same is the case for us: we expect distillation-based training to be the most useful when the amount of labeled training data is limited, as is often the case in practice. As we will show in Section 4, our experiments confirm this expectation.

Distillation-based training has another advantage: the distillation loss can be computed even for data that has no ground truth labels available. Consequently, we obtain a straightforward way of training multi-exit architectures in a *semi-supervised* regime. Note that conventional multi-task based training does not have this ability, because all its loss terms need data with ground truth labels.

2. Related work

Anytime prediction. The roots of anytime computation go back to the work of [6, 16]. In [6], *anytime algorithms* are defined for the first time, and they become widely popular in planning and control [8, 29, 49].

In the context of statistical learning, anytime classifiers were preceded by *cascades* [37, 42, 43]. These are models with variable, instance-dependent runtime; however, they cannot be stopped exogenously. Early examples of truly anytime classifiers were based on streaming nearest neighbors [41] or on classifier ensembles such as decision trees [9], random forests [10], or boosting [13]. A parallel line of work aimed at developing techniques for adapting an arbitrary ensemble to the anytime setting in a learner-agnostic way [3, 11, 40]. These methods usually execute individual classifiers in a dynamically determined, input-dependent order.

More recently, in the context of convolutional networks, two broad approaches to anytime prediction have gained prominence: a) networks whose parts are selectively executed or skipped at test time [23, 30, 44, 45], or b) networks with additional exits [19, 20, 39], from which an appropriate one is chosen at test time. We only discuss multi-exit architectures in detail, as this is the class of models to which our proposed training technique applies.

Multi-exit architectures. The first work to propose attaching early exits to a deep network was [39], where standard image classification architectures such as LeNet, AlexNet and ResNet, were augmented by early exits. Huang *et al*. [19] were the first to propose a custom multi-exit architecture, the Multi-Scale DenseNet, motivated by the observation that early exits interfere with the role of early layers as feature extractors for later use. The MSDNet was the state-of-the-art multi-exit architecture until very recently (see [48]) and it is the one we use in our experiments. Kim et al [20] propose a doubly nested architecture suitable for memory-constrained settings. Finally, there is recent work on discovering multi-exit architectures by neural architecture search (NAS) [48].

While the main contributions of these works are architectural, our focus is on training. In all of these works except [48] (which employs NAS-specific training), multi-exit networks are trained by minimizing the sum of exit-wise losses. We propose a novel training procedure that is applicable to any of these multi-exit architectures.

Orthogonally to the subject of this work, networks with early exits have also been proposed for other purposes, such as providing stronger gradient signals for training [24], or multi-resolution image processing [46].

Distillation. *Distillation* is a popular technique for knowledge transfer between two models. Although similar techniques have been widely considered already in [2, 4, 5, 25, 28], distillation in its most well-known form was introduced only relatively recently in [15].

Since then, the technique has been found to benefit a wide array of applications, including transferring from one architecture to another [12], compression [17, 32], integration with first-order logic [18] or other prior knowledge [47], learning from noisy labels [26], defending against adversarial attacks [31], training stabilization [35, 38], few-shot learning [21], or policy distillation [36]. In the vast majority of applications, including the work cited above, the teacher network is fixed.

Notable exceptions are [1], where distillation is used in a distributed optimization setting to ensure consistency between different copies of a model, [33], where a model is self-trained by distillation from its previous version's predictions, and [27, 34], where distillation between stored predictions from the past and current predictions is used as a regularizer during incremental training. In contrast, we propose distillation from one part of a model to another part. To our knowledge, no previous work has addressed this setting.

3. Distillation Training of Multi-Exit Networks

In this section, we introduce the mathematical notation and the necessary background for the discussion of the proposed *distillation-based training of multi-exit architectures*. Throughout the paper, we consider the case of multi-class classification with an input set $\mathcal{X}$, *e.g*. images, and an output set $\mathcal{Y} = \{1, \ldots, K\}$, where K is the number of classes.

3.1. Multi-Exit Architectures

Multi-exit architectures are layered classification architectures with exits at different depths, see Figure 1 for an illustration. For a system with M exits, this results in a sequence $(\mathbf{p}_1, \ldots, \mathbf{p}_M)$ of probabilistic classifiers $\mathbf{p}_m : \mathcal{X} \to \Delta^K$, each of which maps its input to the probability simplex Δ^K, *i.e*. the set of probability distributions over the K classes. We think of $\mathbf{p}_1, \ldots, \mathbf{p}_M$ as being ordered from least to most expressive (and computationally expensive). In principle, the classifiers may or may not share weights and computation, but in the most interesting and practically useful case, they do share both.

3.2. Prediction

At prediction time, the multi-exit system can operate in two different modes, depending on whether the computational budget to classify an example is known or not.

Budget-mode. If the computational budget is known, the system can directly identify a suitable exit, $\mathbf{p}_{M'}(\mathbf{x})$, to evaluate. This way, it only has to evaluate the shared parts of the architecture and can save the computation of having to evaluate also the earlier exits. How exactly the specific exit is chosen is model-dependent. In this paper, we first determine

the runtime and quality of any single exit on a validation set. Then, for any target runtime, we output the decision of the exit with highest validation accuracy that runs within the available budget.

Anytime-mode. If the computational budget is unknown, *i.e.* for anytime prediction, after receiving a test input $\mathbf{x}$, the system starts evaluating the classifiers $\mathbf{p}_1, \mathbf{p}_2, \dots$ in turn, reusing computation where possible. It continues doing this until it receives a signal to stop – say this happens after the M'-th exit – at which point it returns the predictions of the ensemble created from the evaluated exits, $\frac{1}{M'}\sum_{m=1}^{M'} \mathbf{p}_m(\mathbf{x})$.

3.3. Distillation Training Objective

Our main contribution is a new training objective for multi-exit architectures. Given a training set, $\{(\mathbf{x}_n, y_n)\}_{n=1}^N$, we propose to train the multi-exit architecture by a combination of a *classification loss*, $\mathcal{L}_{\text{cls}}$ and a *distillation loss*, $\mathcal{L}_{\text{dist}}$,

$$\frac{1}{N}\sum_{n=1}^{N}\Big[\mathcal{L}_{\text{cls}}(x_n, y_n) + \mathcal{L}_{\text{dist}}(x_n)\Big]. \tag{1}$$

Distillation loss. The second term, $\mathcal{L}_{\text{dist}}$, is where our main contribution lies, as it introduces the possibility for different exits to learn from each other.

To introduce it, we first remind the reader of the classical distillation framework as introduced in [15]: assume we want a probabilistic classifier $\mathbf{s}$ (called the student) to learn from another classifier $\mathbf{t}$ (called the teacher). This can be achieved by minimizing the (temperature-scaled) cross-entropy between their output distibutions,

$$\ell^\tau(\mathbf{t}, \mathbf{s}) = -\tau^2 \sum_{k=1}^{K} [\mathbf{t}^{1/\tau}(\mathbf{x})]_k \log[\mathbf{s}^{1/\tau}(\mathbf{x})]_k, \tag{2}$$

for some temperature $\tau \in \mathbb{R}_+$, with respect to the parameters of $\mathbf{s}$, where

$$[\mathbf{s}^{1/\tau}(\mathbf{x})]_k = \frac{s_k(\mathbf{x})^{1/\tau}}{\sum_{l=1}^{K} s_l(\mathbf{x})^{1/\tau}} \tag{3}$$

is the distribution obtained from the distribution $\mathbf{s}(\mathbf{x})$ by temperature-scaling, and $[\mathbf{t}^{1/\tau}(\mathbf{x})]_k$ is defined analogously. Note that for typical network architectures, for which the outputs are the result of a softmax operation over logits, temperature scaling can be done efficiently by simply dividing all logits by τ.

The temperature parameter allows controlling the softness of the teachers' predictions: the higher the temperature, the more suppressed is the difference between the largest and the smallest value of the probability vector. This allows compensating for the fact that network outputs are often overconfident, *i.e.* they put too much probability mass on the top predicted class, and too little on the others. The factor τ^2 in Equation (2) ensures that the temperature scaling does not negatively affect the gradient magnitude.

Algorithm 1: Distillation-Based Training

given: $\mathcal{T}, \mu, \tau_*$

$\mathcal{T}(:) \leftarrow \mathcal{T}(1) \cup \dots \cup \mathcal{T}(M)$
$\tau \leftarrow 1$
for $(\mathbf{x}, y)$ **in** data **do**
  $\mathbf{x}' \leftarrow$ shared_part_of_model($\mathbf{x}$)
  for m **in** $1, \dots, M$ **do**
    $\mathbf{p}_m \leftarrow$ softmax(exit$_m(\mathbf{x}')$)
    $\mathbf{t}_m \leftarrow$ detach(softmax(exit$_m(\mathbf{x}')/\tau$))
  end
  loss $\leftarrow \frac{1}{M}\sum_{m=1}^{M} \ell(y, \mathbf{p}_m)$
    $+\frac{1}{M}\sum_{m=1}^{M} \frac{1}{|\mathcal{T}(m)|}\sum_{t\in\mathcal{T}(m)} \ell^\tau(\mathbf{t}_t, \mathbf{p}_m)$
  loss.gradient_update()
  if $\max\left(\frac{1}{|\mathcal{T}(:)|}\sum_{m\in\mathcal{T}(:)} \mathbf{t}_m\right) > \mu$ **then** $\tau \leftarrow \tau\tau_*$
end

Returning to the multi-exit architecture, we follow the same strategy as classical distillation, but use different exits of the multi-exit classifiers both as students and teachers. For any exit m, let $\mathcal{T}(m) \subset \{1, \dots, M\}$ (which could be empty) be the set of teacher exits it is meant to learn from. Then we define the overall *distillation loss* as

$$\mathcal{L}_{\text{dist}}(\mathbf{x}_n) = \frac{1}{M}\sum_{m=1}^{M} \frac{1}{|\mathcal{T}(m)|}\sum_{t\in\mathcal{T}(m)} \ell^\tau(\mathbf{p}_t(\mathbf{x}_n), \mathbf{p}_m(\mathbf{x}_n)), \tag{4}$$

In practice, there are different ways how to choose the set of teachers. The simplest choice, where all exists learn only from the last one, *i.e.* $\mathcal{T}(m) = \{M\}$ for $m < M$ and $\mathcal{T}(M) = \emptyset$, has worked well for us, so we propose it as the default option. However, we also study other setups, for example each exit distilling from all later exits; see Section 4.

Classification loss. The first term in (1) is a standard multi-class classification loss that acts separately on each exit,

$$\mathcal{L}_{\text{cls}}(\mathbf{x}_n, y_n) = \frac{1}{M}\sum_{m=1}^{M} \ell(y_n, \mathbf{p}_m(\mathbf{x}_n)) \tag{5}$$

where $\ell(y, \mathbf{p}) = -\log p_y(\mathbf{x})$ is the cross-entropy loss.

3.4. Optimization

We minimize the training objective (1) using standard gradient-based methods with mini-batches. In particular, all exits are trained at the same time and on the same data. We provide the pseudo-code (for single sample batches) in Algorithm 1. It consists largely of standard gradient-based op-

timization. However, two aspects are specific to distillation-based training: *partial detaching* (detach; line 7), and *temperature annealing* (line 11).

Partial detaching. When minimizing the loss, we have to make sure that indeed only the student learns from the teacher and not vice versa. We achieve this by treating the teachers' predictions, $\mathbf{p}_t(\mathbf{x}_n)$, as constant for gradient calculation of the distillation term (4).

Temperature annealing. Over the course of training, networks tend to grow more confident. In the multi-exit setting, this also applies to exits that serve as teachers (see Figure 6). Therefore, increasingly higher temperatures are needed to "soften" their outputs, and we achieve this by increasing the temperature during training. For this, we introduce an adaptive annealing scheme that aims at keeping the teachers' outputs roughly constant: we define the *confidence* of a classifier to be the maximum of the output vector of class probabilities, averaged over a set of examples. Let μ be an upper bound for the desired teacher confidence. We initialise the temperature at $\tau_0 = 1$ and multiply it by a constant $\tau_* > 1$ whenever the teachers' average temperature-adjusted confidence for a training batch exceeds μ.

3.5. Semi-supervised training

A characteristic property of the distilliaton loss (4) is that it does not depend on the labels of the training data. This means it can also be computed from unlabeled training data, providing us with a straightforward way of training multi-exit architectures in a *semi-supervised* way.

Assume that, in addition to the labeled training set $\{(\mathbf{x}_n, y_n)\}_{n=1}^{N}$, we have an additional set of unlabeled training examples, $\{\mathbf{x}_n\}_{n=N+1}^{N'}$, potentially with $N' \gg N$. We then define the semi-supervised training objective as

$$\frac{1}{N}\sum_{n=1}^{N}\mathcal{L}_{\text{cls}}(x_n, y_n) + \frac{1}{N'}\sum_{n=1}^{N'}\mathcal{L}_{\text{dist}}(x_n). \tag{6}$$

We can minimize this objective using the same techniques as in the fully-supervised case and, in fact, with only minor modifications to the source code.

4. Experiments

In this section, we report on our experimental results, which, in particular, show that distillation-based training consistently outperforms the standard training procedure for multi-exit architectures on image classification benchmarks: ImageNet (subsets) and CIFAR100 (subsets, as well as the full dataset). We further present experiments showing the tentative benefit of semi-supervised distillation when unlabeled training data is available.

We also report on in-depth experiments that provide insight into the working mechanism of the proposed distillation based training, in particular the temperature annealing scheme, and we discuss the choice of teachers.

4.1. Experimental Setup

Datasets. We report on experiments on two standard datasets. For *CIFAR100* [22], we follow the default split, using 50,000 images (500 for each of 100 classes) for training and model selection, and we report the accuracy on the remaining 10,000 test examples. For *ImageNet (ILSVRC 2012)* [7], we use the 1.2 million *train* images of 1000 classes for training and model selection. We report the accuracy on the 50,000 images of the ILSVRC *val* set. During training, we apply data augmentation as in [14, 19]. For testing, we resize the images to 256×256 pixels and center-crop them to 224×224. For both datasets, we pre-process all images by subtracting the channel mean and dividing by the channel standard deviation.

Because we are particularly interested in the low-data regime, we perform experiments using only subsets of the available data: by ImageNet(X) we denote a dataset with X randomly selected examples from each ImageNet class (which are then split 90%/10% into a training and a model selection part). By CIFAR(X) we denote subsets of CIFAR100 that are constructed analogously to the above, but always with 50 images used for model selection (using 10% would be too few for this dataset), and the remaining $X-50$ for training. As additional unlabeled data for the experiments on semi-supervised learning we use $500 - X$ (in the case of CIFAR100) or $700 - X$ (in the case of ImageNet) images per class, sampled randomly from all remaining images, *i.e.* the ones that were not selected as training or validation data (nor as test data, of course).

Model architecture. We use the Multi-Scale DenseNet [19], a state-of-the-art multi-exit architecture with convolutional blocks arranged in a grid of multiple scales (rows) and multiple layers (columns). We train the MSDNets ourselves, following the original architectures and hyper-parameters.[1] The CIFAR MSDNet has 3×24 blocks and 11 exits, one attached to every second layer, starting from layer 4. The ImageNet MSDNet has 4×38 blocks and 5 exits, one on every seventh layer, starting from layer 10.

Baseline. We compare distillation training to the traditional way of training multi-exit architectures, namely by minimizing the *exit-wise loss* (used *e.g.* in [19, 20, 39]),

$$\frac{1}{NM}\sum_{n=1}^{N}\sum_{m=1}^{M}\ell(y_n, \mathbf{p}_m(\mathbf{x}_n)). \tag{7}$$

Note that this coincides with using just the *classification loss* (5) of our training objective. Because labels for all

[1] Our implementation achieves very similar performance to the original MSDNet, e.g. $\approx 75\%$ accuracy on CIFAR100.

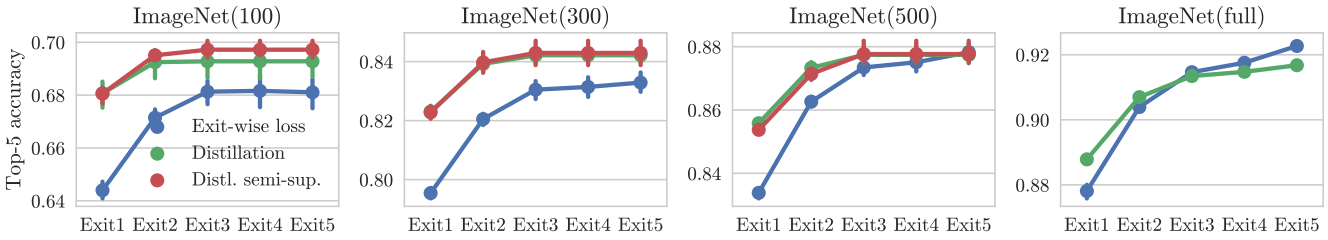

Figure 2: Top-5 accuracy as a function of computational budget (denominated in available exits). MSDNet trained by the exit-wise loss (blue) vs. trained by distillation (green) vs. trained by semi-supervised distillation (red) on ImageNet ILSVRC2012 with 100, 300, 500, and all available (≥ 700) images per class.

	ImageNet(100)			ImageNet(300)			ImageNet(500)			ImageNet(full)	
	Exit-wise loss	Distillation	Distl. semi-sup.	Exit-wise loss	Distillation	Distl. semi-sup.	Exit-wise loss	Distillation	Distl. semi-sup.	Exit-wise loss	Distillation
Exit 1	64.4 ± 0.4	$\mathbf{68.1 \pm 0.5}$	$\mathbf{68.1 \pm 0.4}$	79.5 ± 0.2	$\mathbf{82.3 \pm 0.2}$	$\mathbf{82.3 \pm 0.3}$	83.4 ± 0.2	$\mathbf{85.6 \pm 0.1}$	$\mathbf{85.4 \pm 0.1}$	87.8 ± 0.2	$\mathbf{88.8 \pm 0.1}$
Exit 2	67.1 ± 0.3	$\mathbf{69.2 \pm 0.6}$	$\mathbf{69.5 \pm 0.1}$	82.1 ± 0.2	$\mathbf{83.9 \pm 0.3}$	$\mathbf{84.0 \pm 0.5}$	86.3 ± 0.2	$\mathbf{87.3 \pm 0.2}$	$\mathbf{87.1 \pm 0.3}$	90.4 ± 0.1	$\mathbf{90.7 \pm 0.1}$
Exit 3	68.1 ± 0.5	$\mathbf{69.3 \pm 0.6}$	$\mathbf{69.7 \pm 0.4}$	83.0 ± 0.3	$\mathbf{84.2 \pm 0.3}$	$\mathbf{84.3 \pm 0.6}$	87.3 ± 0.3	87.7 ± 0.3	87.8 ± 0.4	91.5 ± 0.1	91.3 ± 0.2
Exit 4	68.2 ± 0.6	69.3 ± 0.6	$\mathbf{69.7 \pm 0.4}$	83.1 ± 0.3	$\mathbf{84.2 \pm 0.3}$	$\mathbf{84.3 \pm 0.6}$	87.5 ± 0.3	87.7 ± 0.3	87.8 ± 0.4	$\mathbf{91.8 \pm 0.1}$	91.5 ± 0.1
Exit 5	68.1 ± 0.6	69.3 ± 0.6	$\mathbf{69.7 \pm 0.4}$	83.3 ± 0.3	$\mathbf{84.2 \pm 0.3}$	$\mathbf{84.3 \pm 0.6}$	87.8 ± 0.3	87.7 ± 0.3	87.8 ± 0.4	$\mathbf{92.3 \pm 0.1}$	91.7 ± 0.2

Table 1: Top-5 accuracy in % (mean $\pm$ 1.96 stderr) for different computational budgets (denominated in available exits). MSDNet trained by the exit-wise loss vs. trained by distillation vs. trained by semi-supervised distillation on ImageNet with 100, 300, 500 and all available (≥ 700) training images per class. Bold values indicate statistically significant improvements.

training examples are needed to compute the loss, (7) does not have an obvious extension to semi-supervised training.

Optimization and hyper-parameters. We train all models from a random initialization by SGD with Nesterov momentum, an initial learning rate of 0.5, a momentum weight of 0.9, and a weight decay of 10^{-4}. For CIFAR100, we set the batch size to 64 and train for 300 epochs. The learning rate is divided by 10 after epochs 150 and 225. For ImageNet, we set the batch size to 256 and train for 90 epochs. The learning rate is divided by 10 after epochs 30 and 60.

For the temperature annealing we use a confidence limit of $\mu = 0.5$ for CIFAR100 and $\mu = 0.1$ for ImageNet, and $\tau_* = 1.05$ as the temperature multiplier.

Repeated runs. We repeat each experiment 5 or 10 times, each time with a different random subset of the training data and different weight initialization. We report the average performance across the repeated runs as well as its 95% confidence interval (*i.e.* 1.96 times the standard error).

4.2. Main results: budget-mode accuracy

We first report results from experiments when operating the model in *budget-mode*, *i.e.* with a known time budget at test time. As described in Section 3.2, for any value of the budget, we identify the best exit (according to the validation set) that can be computed within the budget, and evaluate its decision. This is often the latest exit among the available ones, but not always: for example, in the low-data regime, the additional capacity of a late exit may make it more likely to overfit, and an intermediate exit might perform better.

Figures 2 and 3 show the results for ImageNet and CIFAR100, respectively. Numeric results can be found in Tables 1 and 2. For each training procedure, we report the resulting model's accuracy for different values of the budget: when only *Exit1* is available, when *Exit1* and *Exit2* are available, and so on.

ImageNet results. Figure 2 and Table 1 compare distillation training and exit-wise training. Distillation training consistently outperforms exit-wise training, in many settings substantially. For most computational budgets, the distillation-trained model has a higher or comparable accuracy, with accuracy gains of up to 3.8%. Conversely, to achieve any given accuracy, the distillation-trained model typically requires far less computation, especially in the data-constrained regime. For example, in the case of ImageNet(100), already *Exit1* suffices to match the accuracy of the exit-wise trained model at *any* budget. Similarly, in the case of ImageNet(300), already at the time *Exit2* becomes available, the distillation-trained model dominates the exit-wise trained model at any budget.

Overall, two main factors seem to affect the performance gap: a) The amount of training data: comparing the results for ImageNet(100) to those for ImageNet(300), ImageNet(500) and ImageNet(full), we see that the smaller the training set, the bigger the benefit from distillation. In the regime of very large data (full ImageNet), distillation seems to trade off the accuracy of early and late exits, instead of providing a uniform improvement. This agrees with earlier studies, *e.g.* [15], that found distillation to have a regular-

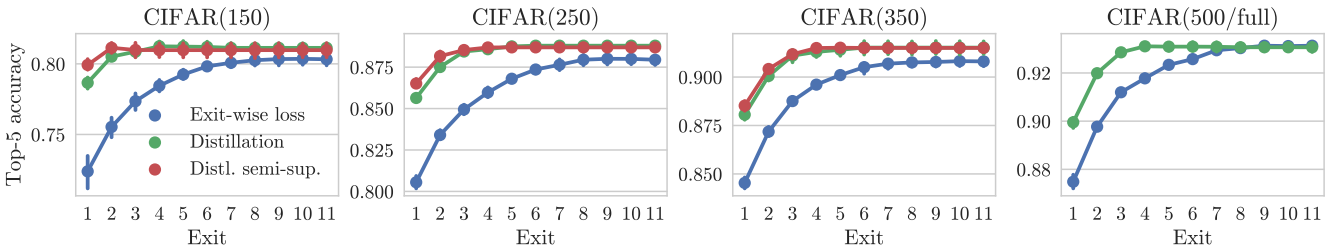

Figure 3: Top-5 accuracy as a function of computational budget (denominated in available exits). MSDNet trained by the exit-wise loss (blue) vs. trained by distillation (green) vs. trained by semi-supervised distillation (red) on CIFAR100 with 150, 250, 350, and 500 images per class.

	CIFAR(150)			CIFAR(250)			CIFAR(350)			CIFAR(500)	
	Exit-wise loss	Distillation	Distl. semi-sup.	Exit-wise loss	Distillation	Distl. semi-sup.	Exit-wise loss	Distillation	Distl. semi-sup.	Exit-wise loss	Distillation
Exit 1	72.4 ± 1.3	$\mathbf{78.7 \pm 0.4}$	$\mathbf{79.9 \pm 0.4}$	80.6 ± 0.4	$\mathbf{85.6 \pm 0.2}$	$\mathbf{86.5 \pm 0.3}$	84.5 ± 0.3	$\mathbf{88.1 \pm 0.3}$	$\mathbf{88.5 \pm 0.3}$	87.5 ± 0.3	$\mathbf{90.0 \pm 0.2}$
Exit 2	75.5 ± 0.7	$\mathbf{80.5 \pm 0.3}$	$\mathbf{81.1 \pm 0.4}$	83.4 ± 0.3	$\mathbf{87.5 \pm 0.2}$	$\mathbf{88.2 \pm 0.3}$	87.2 ± 0.3	$\mathbf{90.1 \pm 0.2}$	$\mathbf{90.4 \pm 0.2}$	89.8 ± 0.1	$\mathbf{92.0 \pm 0.2}$
Exit 3	77.4 ± 0.6	$\mathbf{80.9 \pm 0.4}$	$\mathbf{81.0 \pm 0.5}$	84.9 ± 0.3	$\mathbf{88.4 \pm 0.2}$	$\mathbf{88.5 \pm 0.2}$	88.8 ± 0.2	$\mathbf{91.1 \pm 0.3}$	$\mathbf{91.2 \pm 0.2}$	91.2 ± 0.1	$\mathbf{92.9 \pm 0.2}$
Exit 4	78.4 ± 0.4	$\mathbf{81.2 \pm 0.4}$	$\mathbf{81.0 \pm 0.5}$	86.0 ± 0.3	$\mathbf{88.6 \pm 0.2}$	$\mathbf{88.7 \pm 0.2}$	89.6 ± 0.2	$\mathbf{91.3 \pm 0.3}$	$\mathbf{91.5 \pm 0.2}$	91.8 ± 0.2	$\mathbf{93.1 \pm 0.1}$
Exit 5	79.2 ± 0.3	$\mathbf{81.2 \pm 0.4}$	$\mathbf{81.0 \pm 0.5}$	86.8 ± 0.2	$\mathbf{88.8 \pm 0.3}$	$\mathbf{88.7 \pm 0.2}$	90.1 ± 0.2	$\mathbf{91.4 \pm 0.3}$	$\mathbf{91.5 \pm 0.1}$	92.3 ± 0.1	$\mathbf{93.1 \pm 0.2}$
Exit 6	79.8 ± 0.2	$\mathbf{81.2 \pm 0.4}$	$\mathbf{81.0 \pm 0.5}$	87.4 ± 0.2	$\mathbf{88.8 \pm 0.3}$	$\mathbf{88.7 \pm 0.2}$	90.5 ± 0.3	$\mathbf{91.5 \pm 0.4}$	$\mathbf{91.5 \pm 0.2}$	92.6 ± 0.1	$\mathbf{93.1 \pm 0.2}$
Exit 7	80.1 ± 0.4	$\mathbf{81.1 \pm 0.3}$	$\mathbf{81.0 \pm 0.5}$	87.6 ± 0.3	$\mathbf{88.8 \pm 0.3}$	$\mathbf{88.7 \pm 0.2}$	90.7 ± 0.2	$\mathbf{91.5 \pm 0.4}$	$\mathbf{91.5 \pm 0.2}$	92.9 ± 0.1	93.1 ± 0.2
Exit 8	80.3 ± 0.4	$\mathbf{81.1 \pm 0.3}$	81.0 ± 0.5	87.9 ± 0.3	$\mathbf{88.8 \pm 0.3}$	$\mathbf{88.7 \pm 0.2}$	90.8 ± 0.2	$\mathbf{91.5 \pm 0.4}$	$\mathbf{91.5 \pm 0.2}$	93.0 ± 0.1	93.1 ± 0.2
Exit 9	80.3 ± 0.5	81.1 ± 0.3	81.0 ± 0.5	88.0 ± 0.3	$\mathbf{88.8 \pm 0.3}$	$\mathbf{88.7 \pm 0.2}$	90.8 ± 0.2	$\mathbf{91.5 \pm 0.4}$	$\mathbf{91.5 \pm 0.2}$	93.1 ± 0.1	93.1 ± 0.2
Exit 10	80.4 ± 0.5	81.1 ± 0.3	81.0 ± 0.5	88.0 ± 0.3	$\mathbf{88.8 \pm 0.3}$	$\mathbf{88.7 \pm 0.2}$	90.8 ± 0.3	$\mathbf{91.5 \pm 0.4}$	$\mathbf{91.5 \pm 0.2}$	93.1 ± 0.2	93.1 ± 0.2
Exit 11	80.3 ± 0.5	81.1 ± 0.3	81.0 ± 0.5	87.9 ± 0.3	$\mathbf{88.8 \pm 0.3}$	$\mathbf{88.7 \pm 0.2}$	90.8 ± 0.2	$\mathbf{91.5 \pm 0.4}$	$\mathbf{91.5 \pm 0.2}$	93.1 ± 0.2	93.1 ± 0.2

Table 2: Top-5 accuracy in % (mean $\pm$ 1.96 stderr) for different computational budgets (denominated in available exits). MSDNet trained by the exit-wise loss vs. trained by distillation vs. trained by semi-supervised distillation on CIFAR100 with 150, 250, 350, and 500 images per class. Bold values indicate statistically significant improvements.

izing effect, *i.e.* it helps prevent overfitting in the low-data regime. b) The inference budget: within each subplot, the largest gains are realised for the smallest inference budgets. Intuitively, this makes sense, as the earliest exits can benefit the most from a teacher during learning. In combination, the results suggest that distillation training can provide a large accuracy boost, especially when the amount of training data and/or the computational resources at test time are limited.

Figure 2 and Table 1 also show results for the semi-supervised variant of distillation-based training, as described in Section 3.5. We observe an additional small improvement over the fully-supervised variant, especially when labeled data is limited and unlabeled data plentiful.

CIFAR100 results. We present analogous results for CIFAR100 in Figure 3 and Table 2. We observe similar trends as for ImageNet, though in this case distillation training uniformly outperforms exit-wise training and yields an up to 6.3% improvement in accuracy for a fixed budget. Conversely, distillation training enables the resulting model to stop already after *Exit2* or *Exit3* with comparable accuracy as the conventionally trained model when executed in full. As previously for ImageNet, here, too, we observe that the gains from distillation are largest when training data is limited, and when the inference budget is low.

The semi-supervised variant provides an additional small but consistent improvement. For example, for *Exit1*, the additional unlabeled data translates into $1.2\%, 0.9\%$, and 0.4% increase in accuracy for CIFAR(150), CIFAR(250), and CIFAR(350) respectively.

4.3. Main results: anytime-mode accuracy

In a second set of experiments, we operate the multi-exit model in anytime-mode, *i.e.* the model evaluates all its exits in turn until the (unknown) computational budget is spent, at which point it returns the ensembled prediction of all completed exits. As before, we report multiclass accuracy for different computational budgets, this time denominated in the number of completed exits, or the size of the ensemble.

The results for ImageNet and CIFAR100 are shown in Figures 4 and 5 respectively. Due to lack of space, the corresponding tables with numeric results are deferred to the supplemental material. The results are similar to those for budget-mode evaluation. Across datasets, dataset sizes (except for the very large-scale regime), and computation budgets, the models trained with distillation clearly outperform the model trained without it. The results for semi-supervised learning are less clear: for early exits, the unlabeled data often helps, but we observe a small drop of accuracy of the late exits for CIFAR(150). Still, the proposed method outperforms the exit-wise trained model.

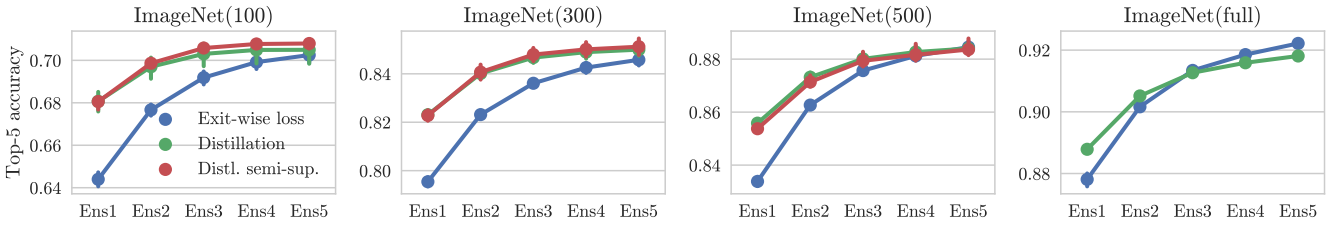

Figure 4: Top-5 accuracy of first-m-exits ensembles ($m = 1, \ldots, 5$) trained by the exit-wise loss (blue) vs. trained by distillation (green) vs. trained by semi-supervised distillation (red) on ImageNet ILSVRC2012 with 100, 300, 500 or all available (≥ 700) training images per class.

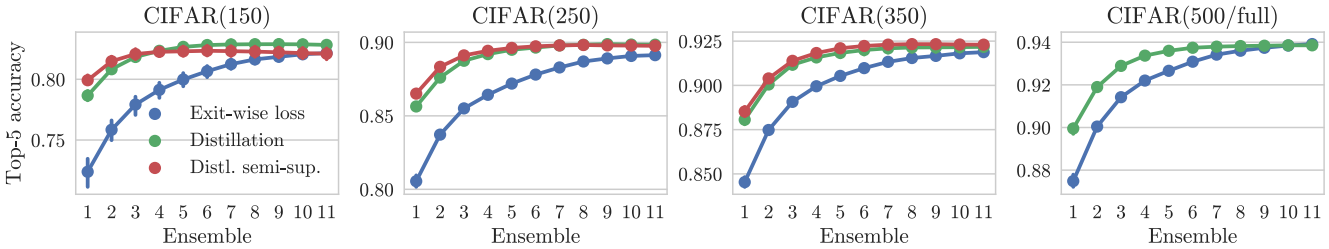

Figure 5: Top-5 accuracy of first-m-exits ensembles ($m = 1, \ldots, 11$) trained by the exit-wise loss (blue) vs. trained by distillation (green) vs. trained by semi-supervised distillation (red) on CIFAR100 with 150, 250, 350 or 500 images per class.

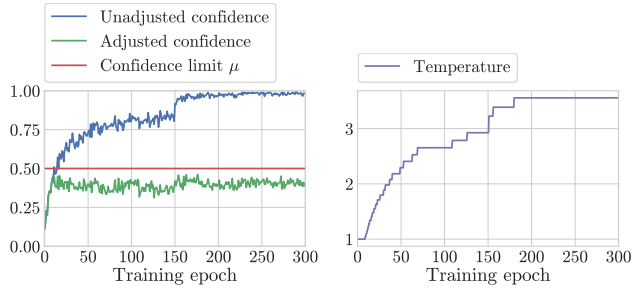

Figure 6: Confidence of MSDNet's last exit with and without temperature annealing throughout training on CIFAR(150). At epochs 150 and 225, the learning rate drops.

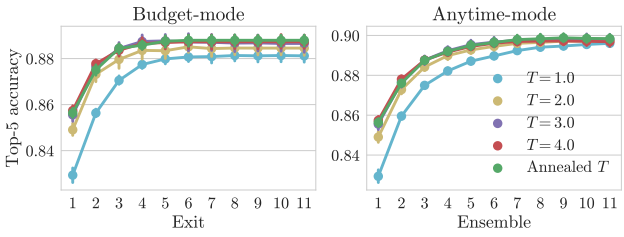

Figure 7: Top-5 accuracy of five models trained by distillation, each with a different temperature setting, on CIFAR(250). Results for different computational budgets in both the budget-mode (left) and the anytime-mode (right).

4.4. Additional experiments

Temperature annealing. In this section, we provide further insight and justification for the proposed temperature annealing scheme. Figure 6 shows the teacher's confidence (blue) during training. One can see that it changes markedly and generally increases. The proposed temperature-scaling procedure reacts to this by raising the temperature over time (purple). The result is that the temperature-adjusted confidence (green) remains roughly constant, and slightly below the confidence limit μ (red). We find that temperature-annealing performs as well as the best fixed temperature (see Figure 7), while being easier to tune.

Choice of teachers. For all experiments reported so far, we used the last exit as the teacher for all other exits. We also performed exploratory studies on how the choice of teacher affects the overall performance, but found the effect to be minor. Details can be found in the supplemental material.

5. Conclusion

In this work, we propose *distillation-based training* for multi-exit image classification architectures. The method is conceptually simple, architecture-agnostic, and as our experiments show, it provides large and robust improvements over the state-of-the-art training procedure, especially in data- or computation-constrained settings. It also naturally supports learning from additional unlabeled training data.

References

[1] Rohan Anil, Gabriel Pereyra, Alexandre Passos, Robert Ormandi, George E. Dahl, and Geoffrey E. Hinton. Large scale distributed neural network training through online distillation. In *International Conference on Learning Representations (ICLR)*, 2018. 3

[2] Jimmy Ba and Rich Caruana. Do deep nets really need to be deep? In *Conference on Neural Information Processing Systems (NIPS)*, 2014. 3

[3] Djalel Benbouzid, Róbert Busa-Fekete, and Balázs Kégl. Fast classification using sparse decision DAGs. In *International Conference on Machine Learing (ICML)*, 2012. 3

[4] Cristian Bucilu, Rich Caruana, and Alexandru Niculescu-Mizil. Model compression. In *Conference on Knowledge Discovery and Data Mining (KDD)*, 2006. 3

[5] Mark Craven and Jude W. Shavlik. Extracting tree-structured representations of trained networks. In *Conference on Neural Information Processing Systems (NIPS)*, 1996. 3

[6] Thomas L. Dean and Mark S. Boddy. An analysis of time-dependent planning. In *AAAI Conference on Artificial Intelligence*, 1988. 3

[7] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009. 5

[8] Mark Drummond and John Bresina. Anytime synthetic projection: Maximizing the probability of goal satisfaction. In *AAAI Conference on Artificial Intelligence*, 1990. 3

[9] Saher Esmeir and Shaul Markovitch. Anytime learning of anycost classifiers. *Machine Learning*, 82:445–473, 2010. 3

[10] Björn Fröhlich, Erik Rodner, and Joachim Denzler. As time goes by – anytime semantic segmentation with iterative context forests. In *German Conference on Pattern Recognition (GCPR)*, 2012. 3

[11] Tianshi Gao and Daphne Koller. Active classification based on value of classifier. In *Conference on Neural Information Processing Systems (NIPS)*, 2011. 3

[12] Krzysztof J. Geras, Abdel-Rahman Mohamed, Rich Caruana, Gregor Urban, Shengjie Wang, Ozlem Aslan, Matthai Philipose, Matthew Richardson, and Charles Sutton. Blending LSTMs into CNNs. In *International Conference on Learning Representations (ICLR) Workshop*, 2016. 3

[13] Alex Grubb and Drew Bagnell. Speedboost: Anytime prediction with uniform near-optimality. In *Conference on Uncertainty in Artificial Intelligence (AISTATS)*, 2012. 3

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 5

[15] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. In *Conference on Neural Information Processing Systems (NIPS) Workshop*, 2014. 2, 3, 4, 6

[16] Eric J. Horvitz. Reasoning about beliefs and actions under computational resource constraints. In *Uncertainty in Artificial Intelligence (UAI)*, 1987. 3

[17] Andrew G. 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. In *arXiv:1704.04861*, 2017. 3

[18] Zhiting Hu, Xuezhe Ma, Zhengzhong Liu, Eduard Hovy, and Eric Xing. Harnessing deep neural networks with logic rules. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2016. 3

[19] Gao Huang, Danlu Chen, Tianhong Li, Felix Wu, Laurens van der Maaten, and Kilian Q Weinberger. Multi-scale dense networks for resource efficient image classification. In *International Conference on Learning Representations (ICLR)*, 2018. 3, 5

[20] Jaehong Kim, Sungeun Hong, Yongseok Choi, and Jiwon Kim. Doubly nested network for resource-efficient inference. In *arXiv:1806.07568*, 2018. 3, 5

[21] Akisato Kimura, Zoubin Ghahramani, Koh Takeuchi, Tomoharu Iwata, and Naonori Ueda. Few-shot learning of neural networks from scratch by pseudo example optimization. In *British Machine Vision Conference (BMVC)*, 2018. 3

[22] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009. 5

[23] Gustav Larsson, Michael Maire, and Gregory Shakhnarovich. FractalNet: Ultra-deep neural networks without residuals. In *International Conference on Learning Representations (ICLR)*, 2017. 3

[24] Chen-Yu Lee, Saining Xie, Patrick Gallagher, Zhengyou Zhang, and Zhuowen Tu. Deeply-supervised nets. In *Conference on Uncertainty in Artificial Intelligence (AISTATS)*, 2015. 3

[25] Jinyu Li, Rui Zhao, Jui-Ting Huang, and Yifan Gong. Learning small-size DNN with output-distribution-based criteria. In *Conference of the International Speech Communication Association (Interspeech)*, 2014. 3

[26] Yuncheng Li, Jianchao Yang, Yale Song, Liangliang Cao, Jiebo Luo, and Li-Jia Li. Learning from noisy labels with distillation. In *International Conference on Computer Vision (ICCV)*, 2017. 3

[27] Zhizhong Li and Derek Hoiem. Learning without forgetting. In *European Conference on Computer Vision (ECCV)*, 2016. 3

[28] Percy Liang, Hal Daumé III, and Dan Klein. Structure compilation: trading structure for features. In *International Conference on Machine Learing (ICML)*, 2008. 3

[29] Maxim Likhachev, Geoffrey J. Gordon, and Sebastian Thrun. ARA*: Anytime A* with provable bounds on sub-optimality. In *Conference on Neural Information Processing Systems (NIPS)*, 2004. 3

[30] Ji Lin, Yongming Rao, Jiwen Lu, and Jie Zhou. Runtime neural pruning. In *Conference on Neural Information Processing Systems (NIPS)*, 2017. 3

[31] Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. Distillation as a defense to adversarial perturbations against deep neural networks. In *IEEE Symposium on Security and Privacy (S&P)*, 2016. 3

[32] Antonio Polino, Razvan Pascanu, and Dan Alistarh. Model compression via distillation and quantization. In *International Conference on Learning Representations (ICLR)*, 2018. 3

[33] Ilija Radosavovic, Piotr Dollár, Ross Girshick, Georgia Gkioxari, and Kaiming He. Data distillation: Towards omni-supervised learning. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. 3

[34] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H. Lampert. iCaRL: Incremental classifier and representation learning. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 3

[35] Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. Fitnets: Hints for thin deep nets. In *International Conference on Learning Representations (ICLR)*, 2015. 3

[36] Andrei A. Rusu, Sergio Gomez Colmenarejo, Caglar Gulcehre, Guillaume Desjardins, James Kirkpatrick, Razvan Pascanu, Volodymyr Mnih, Koray Kavukcuoglu, and Raia Hadsell. Policy distillation. In *International Conference on Learning Representations (ICLR)*, 2016. 3

[37] Jan Šochman and Jiří Matas. Waldboost – learning for time constrained sequential detection. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2005. 3

[38] Zhiyuan Tang, Dong Wang, and Zhiyong Zhang. Recurrent neural network training with dark knowledge transfer. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2016. 3

[39] Surat Teerapittayanon, Bradley McDanel, and H.T. Kung. BranchyNet: Fast inference via early exiting from deep neural networks. In *International Conference on Pattern Recognition (ICPR)*, 2016. 3, 5

[40] Kirill Trapeznikov and Venkatesh Saligrama. Supervised sequential classification under budget constraints. In *Conference on Uncertainty in Artificial Intelligence (AISTATS)*, 2013. 3

[41] Ken Ueno, Xiaopeng Xi, Eamonn Keogh, and Dah-Jye Lee. Anytime classification using the nearest neighbor algorithm with applications to stream mining. In *IEEE International Conference on Data Mining (ICDM)*, 2006. 3

[42] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2001. 3

[43] Paul Viola and Michael Jones. Robust real-time face detection. *International Journal of Computer Vision (IJCV)*, 57(2):137–154, 2004. 3

[44] Xin Wang, Fisher Yu, Zi-Yi Dou, and Joseph E. Gonzalez. SkipNet: Learning dynamic routing in convolutional networks. In *European Conference on Computer Vision (ECCV)*, 2018. 3

[45] Zuxuan Wu, Tushar Nagarajan, Abhishek Kumar, Steven Rennie, Larry S. Davis, Kristen Grauman, and Rogerio Feris. BlockDrop: Dynamic inference paths in residual networks. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. 3

[46] Saining Xie and Zhuowen Tu. Holistically-nested edge detection. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015. 3

[47] Ruichi Yu, Ang Li, Vlad I. Morariu, and Larry S. Davis. Visual relationship detection with internal and external linguistic knowledge distillation. In *International Conference on Computer Vision (ICCV)*, 2017. 3

[48] Chris Zhang, Mengye Ren, and Raquel Urtasun. Graph HyperNetworks for neural architecture search. In *International Conference on Learning Representations (ICLR)*, 2019. 3

[49] Shlomo Zilberstein. Using anytime algorithms in intelligent systems. *AI magazine*, 17(3):73, 1996. 3