Adaptive Deep Neural Network Inference Optimization with EENet

Fatih Ilhan*, Ka-Ho Chow*, Sihao Hu*, Tiansheng Huang*, Selim Tekin*, Wenqi Wei*, Yanzhao Wu*,
Myungjin Lee†, Ramana Kompella†, Hugo Latapie†, Gaowen Liu†, Ling Liu*
*Georgia Institute of Technology, Atlanta, GA, USA †CISCO Research, San Jose, CA, USA
*{filhan, khchow, sihaohu, thuang, stekin6, wenqiwei, yanzhaowu, ling.liu}@gatech.edu,
†{myungjle, rkompell, hlatapie, gaoliu}@cisco.com

Abstract

Well-trained deep neural networks (DNNs) treat all test samples equally during prediction. Adaptive DNN inference with early exiting leverages the observation that some test examples can be easier to predict than others. This paper presents EENet, a novel early-exiting scheduling framework for multi-exit DNN models. Instead of having every sample go through all DNN layers during prediction, EENet learns an early exit scheduler, which can intelligently terminate the inference earlier for certain predictions, which the model has high confidence of early exit. As opposed to previous early-exiting solutions with heuristics-based methods, our EENet framework optimizes an early-exiting policy to maximize model accuracy while satisfying the given per-sample average inference budget. Extensive experiments are conducted on four computer vision datasets (CIFAR-10, CIFAR-100, ImageNet, Cityscapes) and two NLP datasets (SST-2, AgNews). The results demonstrate that the adaptive inference by EENet can outperform the representative existing early exit techniques. We also perform a detailed visualization analysis of the comparison results to interpret the benefits of EENet.

1. Introduction

Deep neural networks (DNNs) have shown unprecedented success in various fields such as computer vision and natural language processing, thanks to the advances in computation technologies (GPUs, TPUs) and the increasing amount of available data to train very large and deep neural networks. However, these models usually have very high computational costs, which leads to many practical challenges in deployment and inference on edge computing applications, especially for edge clients with limited resources such as smartphones, IoT devices, and embedded devices [10, 17, 24]. Significant research has been dedicated to improving the computational efficiency of DNN models through training phase optimizations, such as model quantization and deep compression [8], neural network pruning [7, 9], knowledge distillation [12], and multi-exit DNNs [13, 25]. Among these, multi-exit DNNs stand out as a promising technique since they enable adaptive inference with early exiting to reduce inference latency based on the available budget of the deployed device [17].

Early exiting employs the idea of injecting early exit classifiers into certain intermediate layers of a deep learning model and with that multi-exit DNN, gaining the capability to adaptively stop inference at one of these early exits in runtime [17]. At the model training phase, we train the DNN model with the additional multiple exit branches through joint loss optimization. During adaptive inference, the multi-exit DNN model can elastically adjust how much time to spend on each sample based on an early exit scheduling policy to maximize the overall performance metric under the given total latency budget. Even though there is a significant line of research on improving the performance of multi-exit DNNs through designing specialized architectures [6, 13, 26, 28] and training algorithms [18, 22], work on optimizing early exit policies is very limited. In the literature, most methods still consider hand-tuned confidence measures for sample scoring such as maximum score [13], entropy [25], voting [30] etc., and heuristics-based threshold computation approaches.

We argue that through optimizing an early exit scheduling policy, the potential of early exiting can be maximized to efficiently utilize the provided inference resources on heterogeneous edge devices. To this end, we present EENet (Early-Exiting Network), an early exit scheduling framework with two novel functionalities. First, EENet introduces the concept of exit scores by jointly evaluating and combining two complimentary statistics: (i) multiple confidence measures and (ii) class-wise prediction scores. This enables EENet to calibrate output scores for handling statistical differences among prediction scores for different classes. Second, EENet optimizes the distribution of samples to different exits and computes the exit thresholds given the inference budget. We note that the optimization of

EENet is performed on the validation dataset in a decoupled manner with the multi-exit DNN training phase. This enables EENet to quickly adapt to varying budget scenarios by only training the schedulers and eliminates the requirement of re-training the full DNN model each time the inference budget changes as opposed to other studies that integrate two optimization phases [1, 3]. Lastly, EENet is model-agnostic and applicable to all pre-trained multi-exit models. In addition, flexible deployment is applicable for edge clients with heterogeneous computational resources by running only a partial model split until a certain early exit that matches the resource constraint.

We conduct extensive experiments with multiple DNN architectures (ResNet [11], DenseNet [14], MSDNet [13], HRNet [27], BERT [5]) on three image recognition benchmarks (CIFAR10, CIFAR100, ImageNet), one image segmentation benchmark (Cityscapes) and two sentiment analysis benchmarks (SST-2, AgNews). We demonstrate the improvements of EENet compared to existing representative approaches, such as BranchyNet [25], MSDNet [13], PABEE [30] and MAML-stop [1]. Lastly, we provide an ablation study of EENet design components and visual analysis to interpret the behavior of our approach.

2. Related Work

BranchyNet [25] is the first to explore the idea of early exits for adaptive inference with DNNs. It considers the entropy of prediction scores as the measure of confidence and sets the early exit thresholds heuristically. Certain studies consider maximum prediction score instead of entropy as the exit confidence measure [13] on computer vision tasks. PABEE [30] proposes stopping the inference when the number of predictions on the same output reaches a certain patience threshold. However, this method depends on having a high number of early exits to produce meaningful scores so that the samples can be separated with a higher resolution for the exit decision. All these methods introduce heuristics-based rules, which do not optimize the early exit policy in terms of scoring and threshold computation.

Some recent efforts propose task-dependent confidence measures [19, 20] or modifying the training objective to include exit policy learning during the training of the multi-exit DNN [1, 3]. EPNet [3] proposes using Markov decision processes however, they add an early exit classifier at each exit to increase the number of states, which is computationally unfeasible since each early exit introduces an additional computational cost during both training and inference. MAML-stop [1] proposes a variational Bayesian approach to learning when to stop predicting during training. The major drawback of these approaches is the requirement of training the full model for every different budget value since the optimization of early exiting behavior is integrated into the DNN training phase.

3. Methodology

Overview. Figure 1 provides an architectural overview of our system. Given a pre-trained DNN model, we first inject early exits and finetune the multi-exit model on the training dataset. During this training phase, we jointly optimize the losses from each exit. During multi-exit model training to enhance the performance of earlier exits, we perform self-distillation between early exits and the final exit by minimizing the KL divergence between outputs. Next, we initiate the early exit scheduling policy optimization phase on the validation dataset. The goal of this step is to optimize the exit scoring functions and exit thresholds such that the provided inference budget is satisfied and the performance metric is maximized. We first obtain the validation predictions of the trained multi-exit model and construct the training dataset for scheduling optimization. We iteratively optimize the exit scoring functions (g_k) and exit assignment functions (h_k). Finally, we compute the early exit triggering thresholds, one for each exit, and complete the early exit scheduling policy optimization phase. The trained multi-exit model, with the exit scoring functions and exit thresholds, is then utilized during the test to determine which exit each sample should take. In the rest of this section, we provide the multi-exit model training in Section 3.1. Then, we describe the adaptive inference with early exiting and scheduling optimization methodology in Section 3.2.

3.1. Multi-exit Model Training with Self-distillation

To enable a given pre-trained DNN classifier to perform early exiting, we first inject early exit classifiers into the model at certain intermediate layers. If the configuration of per-client computational resource constraints, which specifies the computational budget B_m of a client $m \in \{1, ..., M\}$, is available, the number of exits K can be determined heuristically with clustering analysis of the computational requirements of potential users $\{B_m | 1 \leq m \leq M\}$. In this work, we set K based on the number of resource categories of heterogeneous edge clients. Then, we set the exit locations l_k for each exit k based on the lowest resource capacity from the group of edge clients in the kth resource category by finding the largest submodel f_k while not exceeding the constraints satisfying that $\texttt{cost}(f_k) \leq \min\{B_m | m \in \mathcal{S}_k\}$, where $\mathcal{S}_k$ is the set of clients at the kth resource category and $\texttt{cost}$ is the cost function (#PARAMS, #FLOPs, latency etc.).

Let f denote a multi-exit classification model capable of outputting multiple predictions after the injection of early exit subnetworks f_k^e at each exit k. After preparing the multi-exit model with K exits, we start finetuning with joint loss optimization across all K exit classifiers. Let us denote the set of output probability scores of f for one sample as $\{\hat{\boldsymbol{y}}_k\}_{k=1}^K$ and the label as $y \in \mathcal{C}$, where K is the number of exits and $\mathcal{C} = \{1, 2, \ldots C\}$ is the set of classes.

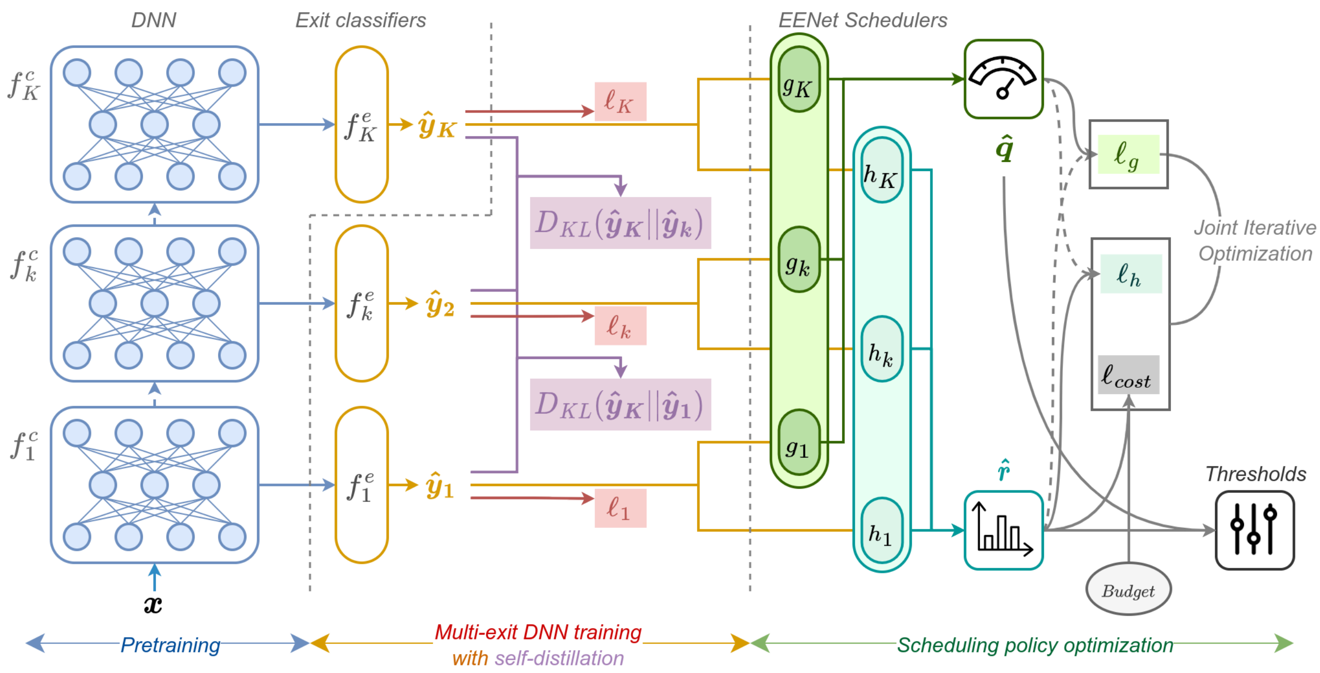

Figure 1. Architectural overview of EENet. The training phase contains the steps necessary to obtain and train the multi-exit DNN model. The scheduling optimization phase involves optimizing the early exit schedulers and computing exit thresholds given the inference budget. Each scheduler at exit k encapsulates an exit scoring function g_k, which outputs an exit score $\hat{q}_k$ representing the confidence in the correctness of the model prediction, and an exit assignment function h_k, which is used to estimate the posterior distribution over the exit assignment $\hat{\boldsymbol{r}}$.

Here, at each exit k, $\hat{\boldsymbol{y}}_k = f_k(\boldsymbol{x}) = [\ldots \hat{y}_{k,c} \ldots] \in \mathbb{R}^C$ is the vector of prediction scores for each class c, where $f_k \triangleq f_k^e \circ f_k^c \circ \ldots f_1^c$ with f_k^c the kth core subnetwork and f_k^e the kth early exit subnetwork of the multi-exit model f.

During the training/finetuning of these K-exit models, we minimize the weighted average of cross-entropy losses from each exit combined with the KL divergence among exit predictions for self-distillation: $\mathcal{L}_{train} = \sum_{k=1}^{K} \gamma_k \ell_k + \alpha_{KL} \sum_{k=1}^{K-1} D_{KL}(\hat{\boldsymbol{y}}_K||\hat{\boldsymbol{y}}_k)$, where ℓ_k is the cross-entropy loss at the kth exit, $\gamma_k = \frac{k}{K(K+1)}$ is the loss weight of the kth exit, and $D_{KL}(\hat{\boldsymbol{y}}_K||\hat{\boldsymbol{y}}_k) = \texttt{sum}(\sigma(\hat{\boldsymbol{y}}_K/\tau) \log \frac{\sigma(\hat{\boldsymbol{y}}_K/\tau)}{\sigma(\hat{\boldsymbol{y}}_k/\tau)})\tau^2$ is the forward KL divergence from $\hat{\boldsymbol{y}}_k$ to $\hat{\boldsymbol{y}}_K$. Here, α_{KL} and τ are the hyperparameters that control the effect of self-distillation.

3.2. Adaptive Early-Exit Inference Optimization

After training the multi-exit classification model f with K exits, we store the validation predictions of the multi-exit model at each exit to use during scheduler optimization. Then, we move forward to generating an early exit policy under given budget constraints.

Problem Definition. We are given an average per-sample inference budget B (in terms of latency, #FLOPs etc.), and the vector of computation costs $\boldsymbol{c} \in \mathbb{R}^K$ of the model f until each exit. On a dataset with N examples, $\mathcal{D} = \{(\{\hat{\boldsymbol{y}}_{n,k}\}_{k=1}^K, y_n)\}_{n=1}^N$ containing model prediction scores and labels on the validation set, the ultimate goal is to find the exit scoring functions ($\{g_k\}_{k=1}^K$) and the thresholds $\boldsymbol{t} \in \mathbb{R}^K$ that maximizes the accuracy such that:

$$\boldsymbol{t}, \{g_k\}_{k=1}^K = \underset{\boldsymbol{t} \in \mathbb{R}^K, \{g_k : \mathbb{R}^D \to \mathbb{R}\}_{k=1}^K}{\arg\max} \frac{1}{N} \sum_{n=1}^{N} \mathbf{1}_{\hat{y}_{n,k_n} = y_n} \quad (1)$$

while satisfying the given average per-sample inference budget B such that $\frac{1}{N}\sum_{n=1}^{N} c_{k_n} \leq B$.

Here, $k_n = min\{k | g_k(\hat{\boldsymbol{y}}_{n,k}) \geq t_k\}$ denotes the minimum exit index where the computed exit score is greater or equal to the threshold of that exit, i.e. the assigned exit for the nth sample. Exit scoring functions g_k take the corresponding prediction score vector at exit-k as input and return the exit score for that sample, which represents the likelihood of a correct prediction, i.e. $g_k(\hat{\boldsymbol{y}}) \triangleq P(y = \hat{y} | \hat{\boldsymbol{y}}, k)$. The pair of exit scoring functions $(g_1, g_2 \ldots g_K)$ and thresholds $(t_1, t_2, \ldots t_K)$ that maximize the validation accuracy while satisfying the given average budget are then used for early-exit enabled inference as will be explained at Section 3.2.1 and illustrated in Figure 2.

3.2.1 EENet Scheduler Architecture

We develop a multi-objective optimization approach with three goals: (i) estimating whether a given prediction is correct or not, (ii) estimating the exit assignment for a given sample, and (iii) satisfying the given average per sample inference budget over the validation dataset. To this end, first, we define a target variable $q_k = \mathbf{1}_{\hat{y}_k=y}$, representing the correctness of a prediction, where $\hat{y}_k \triangleq \arg\max_{c\in\mathcal{C}} \hat{y}_{k,c}$ is the predicted class. Here, q_k is equal to one if the prediction at exit k is correct and zero otherwise. In addition, we define a confidence score vector $\boldsymbol{a}_k$ containing three different confidence measures: (i) maximum prediction score, which measures the confidence based on the highest probability score among predictions for each class (ii) entropy, which measures the confidence based on the entropy of probability scores (i.e., if entropy is high, there is high uncertainty therefore, confidence is low) and (iii) voting, which measures the confidence based on the number of agreements among different exit classifiers. These values can be formulated as below:

$$a_k^{(max)} = \max_{c\in\mathcal{C}} \hat{y}_{k,c}, \tag{2}$$

$$a_k^{(entropy)} = 1 + \frac{\sum_{c'=1}^{C} \hat{y}_{k,c'} \log \hat{y}_{k,c'}}{\log C}, \tag{3}$$

$$a_k^{(vote)} = \frac{1}{k} \max_{c\in\mathcal{C}} \sum_{k'=1}^{k} \mathbf{1}_{\hat{y}_{k'}=c}. \tag{4}$$

At each exit k, using the prediction score vector $\hat{\boldsymbol{y}}_k$ and the confidence score vector $\boldsymbol{a}_k$, EENet calibrates the predictions and computes the exit score $\hat{q}_k$ as a linear combination to estimate the correctness of the prediction: $\hat{q}_k = g_k(\hat{\boldsymbol{y}}_k, \boldsymbol{a}_{\boldsymbol{k}}, \boldsymbol{b}_k) = \texttt{clamp}(\boldsymbol{\psi}^T[\hat{\boldsymbol{y}}_k, \boldsymbol{a}_{\boldsymbol{k}}, \boldsymbol{b}_k], 0, 1) \triangleq P(y = \hat{y}|\hat{\boldsymbol{y}}, k)$, where $\boldsymbol{b}_k = [\hat{q}_1, \ldots \hat{q}_{k-1}]$ for $k > 1$ is the collection of previous exit scores.

Next, we consider the assigned exit k as latent variable and define the exit assignment functions h_k to estimate the posterior distribution over exits $\hat{r}_k \triangleq p(k|\hat{\boldsymbol{y}})$ as follows:

$$\tilde{r}_k = h_k(\hat{\boldsymbol{y}}_k, \boldsymbol{a}_k, \boldsymbol{b}_k), \quad \hat{r}_k = \frac{e^{\tilde{r}_k}}{\sum_{k'=1}^{K} e^{\tilde{r}_{k'}}}, \tag{5}$$

where h_k is a two-layer MLP with ReLU activation and D_h hidden dimensions. Outputs over all exits are normalized using the softmax function.

3.2.2 Optimization

We provide the pseudocode for the optimization of the exit scoring functions $\{g_k\}_{k=1}^{K}$ and thresholds $\boldsymbol{t}$ in Algorithm 1. Here in line 4-6, given the dataset containing model predictions and target labels on the validation data, budget and inference costs, we iteratively optimize the exit scoring and assignment functions by minimizing the losses observed by these functions.

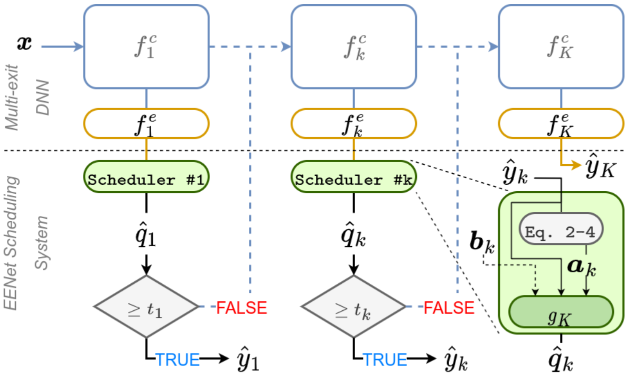

Figure 2. Adaptive inference with early exit scheduling. Early exiting at exit k is performed if the computed exit score $\hat{q}_k$ (estimated probability of a correct prediction) is higher than the threshold t_k. Execution continues until the condition is met.

For the optimization objective of exit scoring functions, we define the loss $\mathcal{L}_g$ as follows:

$$\mathcal{L}_g = \frac{1}{K} \sum_{n=1}^{N} \sum_{k=1}^{K} w_{n,k} \ell_g(\hat{q}_{n,k}, q_{n,k}), \tag{6}$$

where $\ell_g(\hat{q}_k, q_k) = q_k \log(\hat{q}_k) + (1 - q_k) \log(1 - \hat{q}_k)$ is the

Algorithm 1 `EENet Scheduling Optimization`

```
 1: Inputs: $\mathcal{D} = \{(\{\hat{\boldsymbol{y}}_{n,k}\}_{k=1}^{K}, y_n)\}_{n=1}^{N}$, $B$, $\boldsymbol{c} \in \mathbb{R}^K$
 2: Outputs: $\{g_k\}_{k=1}^{K}$, $\boldsymbol{t} \in \mathbb{R}^K$
 3: Initialize: $\boldsymbol{h} \leftarrow \texttt{zeros}(N)$, $\boldsymbol{t} \leftarrow \texttt{ones}(k)$
 4: for iter $i = 1$ to $N_{iter}$ do
 5:     Optimize $\{g_k\}_{k=1}^{K}$ by minimizing $\mathcal{L}_g$ on $\mathcal{D}$.
 6:     Optimize $\{h_k\}_{k=1}^{K}$ by minimizing $\mathcal{L}_h$ on $\mathcal{D}$.
 7: Compute exit scores using $\{g_k\}_{k=1}^{K}$: $\hat{\mathbf{Q}} \triangleq (\hat{q}_{n,k}) \in \mathbb{R}^{N\times K}$
 8: $\mathbf{S} = (s_{n,k}) \in \mathbb{N}^{N\times K} \leftarrow \texttt{argsort}(\hat{\mathbf{Q}}, 1)$
 9: for exit index $k = 1$ to $K - 1$ do
10:     $c \leftarrow 0$
11:     Estimate exit distribution using $\{h_k\}_{k=1}^{K}$: $p_k \leftarrow \frac{1}{N}\sum_{n=1}^{N} \hat{r}_{n,k}$
12:     for sample index $n = 1$ to $N$ do
13:         if $h_{s_{n,k}} = 0$ then
14:             $c \leftarrow c + 1$
15:             $h_{s_{n,k}} \leftarrow 1$
16:             if $c = \texttt{round}(Np_k)$ then
17:                 $t_k \leftarrow \hat{q}_{s_{n,k},k}$
18:                 break
19: $t_K \leftarrow 0$
20: return $\{g_k\}_{k=1}^{K}$, $\boldsymbol{t}$
```

binary CE loss. N is the number of validation data samples and $w_{n,k} = \frac{\hat{r}_{n,k}}{\sum_{n'=1}^{N} \hat{r}_{n',k}}$ is the loss weight for the nth sample at the kth exit. This weighting scheme encourages exit scoring functions to specialize in their respective subset of data. For instance, for a given sample, if the exit assignment score at a particular exit is high, the exit scoring function at that exit will be optimized with that sample more compared to other samples.

For the optimization objective of exit assignment functions, we ideally would try to achieve a distribution that maximizes the following objective:

$$\arg\max_{p(k|\cdot)} \mathbb{E}_{k\sim p(k|\cdot)} \log P(y = \hat{y}|\hat{\boldsymbol{y}}, k) + \beta_h H(p). \tag{7}$$

In other words, the expected log-probability of a correct prediction should be maximized in addition to an entropy regularization term, $H(p) \triangleq -\sum_k p(k) \log p(k)$, whose effect is controlled with $\beta_h > 0$. Taking the derivative with respect to p yields the following target distribution:

$$p^*(k|y, \hat{\boldsymbol{y}}_k) = \frac{P(y = \hat{y}|\hat{\boldsymbol{y}}_k, k)^{1/\beta_h}}{\sum_{k'} P(y = \hat{y}|\hat{\boldsymbol{y}}_{k'}, k')^{1/\beta_h}} = \frac{\hat{q}_k^{\,1/\beta_h}}{\sum_{k'} \hat{q}_{k'}^{\,1/\beta_h}} \tag{8}$$

Our goal is to optimize a distribution $p(k|\hat{\boldsymbol{y}}_k)$ that does not require observing the label and it will be as close as possible to this target distribution $p^*(k|y, \hat{\boldsymbol{y}}_k)$ while satisfying the budget requirement. To this end, we define the exit assignment loss as a combination of KL-divergence between p and p^*, and a budget loss such that $\mathcal{L}_h = D_{KL}(p^*||p) + \alpha_{cost}\ell_{cost}$, where

$$D_{KL}(p^*||p) = \frac{1}{NK} \sum_{n=1}^{N} \sum_{k=1}^{K} p^*(k|y_n, \hat{\boldsymbol{y}}_{n,k}) \log \frac{p^*(k|y_n, \hat{\boldsymbol{y}}_{n,k})}{p(k|\hat{\boldsymbol{y}}_{n,k})}, \tag{9}$$

$$\ell_{cost} = \frac{1}{B} |B - \frac{1}{N} \sum_{n=1}^{N} \sum_{k=1}^{K} \hat{r}_{n,k} c_k|. \tag{10}$$

$p(k|\hat{\boldsymbol{y}}_{n,k}) = \hat{r}_{n,k}$ is provided in Eq. 5 and $p^*(k|y_n, \hat{\boldsymbol{y}}_{n,k})$ is provided in Eq. 8. Here, ℓ_{cost} is the budget cost and calculates the scaled absolute distance between the inference budget and used resources using the computed exit assignment scores and inference costs until each exit.

After optimizing the exit scoring functions $\{g_k\}_{k=1}^{K}$ and exit assignment functions $\{h_k\}_{k=1}^{K}$, we compute the exit thresholds. We first sort the exit scores for all samples at each exit (line 8). Then at each exit, we let Np_k samples with the highest scores exit, where $p_k = \frac{1}{N} \sum_{n=1}^{N} \hat{r}_{n,k}$ is the mean of exit assignment scores of validation samples. And, we set the threshold to the exit score of the last exited sample with the lowest score (Algorithm 1, lines 9-19).

We provide the formulation of our framework for the traditional classification tasks. For dense prediction tasks such as image segmentation, we consider the mean of the pixel-level scores during exit score and assignment computations. Regression tasks such as object detection/tracking (early exiting for easier frames with fewer objects/less occlusion, etc.) are also applicable with minor modifications: (i) instead of $\hat{\boldsymbol{y}}_k$, we can use the hidden layer output $(f_k^c \circ \ldots f_1^c)(\boldsymbol{x})$ as input to g_k and h_k, (ii) we can treat exit scoring functions as boosters by defining residuals as their targets, and (iii) for MSE loss, we can consider Gaussian distribution to derive the likelihood for Eq. 7.

4. Experiments

We conduct experiments with convolutional and transformer-based networks to evaluate EENet and report the performance improvements obtained for budget-constrained adaptive inference on six image/text benchmarks (CIFAR-10, CIFAR-100, ImageNet, Cityscapes, SST-2 and AgNews). We provide detailed explanations of experiment setups, and further analysis in the supplementary material. Our code is available at https://github.com/git-disl/EENet.

4.1. Datasets and Preprocessing

In image classification experiments, we work on CIFAR-10/100 [16] and ImageNET [4] datasets. CIFAR-10 and CIFAR-100 contain 50000 train and 10000 test images with 32x32 resolution from 10 and 100 classes respectively. ImageNET contains 1.2 million train and 150000 validation images (used for testing) with 224x224 resolution from 1000 classes. We hold out randomly selected 5000 images from CIFAR-10/100 train set and 25000 images from the ImageNET train set for validation. We follow the data augmentation techniques applied in [11], zero padding, center cropping and random horizontal flip with 0.5 probability. For image segmentation, we use the Cityscapes [2] dataset, which contains 5000 (train: 2975, val: 500, test: 1525) images with size 1024x2048, finely labeled in pixel-level for 19 classes. In text classification experiments, we consider SST-2 [23] and AGNews [29] datasets. SST-2 contains 67349 train, 872 validation and 1821 samples with positive or negative labels. AGNews contains 120000 train and 7600 samples from four classes. We hold randomly selected 5000 sentences for validation.

4.2. Validation of EENet with Comparison

We compare our method with various representative early exiting methodologies such as BranchyNet [25], PABEE [30], MSDNet [13] and MAML-stop [1] in terms of the accuracy obtained under different latency budget constraints. We consider average latency per sample as the budget definition throughout the experiments. For MSDNet [13], BranchyNet [25] and PABEE [30], we use maximum score, entropy-based and agreement-based scores as

Dataset, Model and	Base Model	Average Latency	Speed Gain	Metric (%)				
Base Performance	Latency	Budget per sample		**BranchyNet**	**MSDNet**	**PABEE**	**MAML-stop**	**EENet**
CIFAR-10		3.50 ms	1.34x	93.76	93.81	93.69	-	**93.84** ±0.05
ResNet56 w/ 3 exits	4.70 ms	3.00 ms	1.56x	92.57	92.79	91.85	-	**92.90** ±0.13
93.90% - *accuracy*		2.50 ms	1.88x	87.55	88.76	84.39	88.67	**88.90** ±0.19
CIFAR-100		7.50 ms	1.36x	73.96	74.01	73.68	-	**74.08** ±0.13
DenseNet121 w/ 4 exits	10.20 ms	6.75 ms	1.51x	71.65	71.99	68.10	-	**72.12** ±0.20
75.08% - *accuracy*		6.00 ms	1.70x	68.13	68.70	61.13	69.00	**69.57** ±0.22
ImageNet		125.00 ms	1.56x	74.10	74.13	74.05	-	**74.18** ±0.13
MSDNet35 w/ 5 exits	195.14 ms	100.00 ms	1.95x	72.44	72.70	72.40	-	**72.75** ±0.11
74.60% - *accuracy*		75.00 ms	2.60x	69.32	69.76	68.13	69.55	**69.88** ±0.15
Cityscapes		100.00 ms	1.32x	79.22	78.75	72.20	-	**80.03** ±0.21
HRNET-W48 w/ 3 exits	131.60 ms	50.00 ms	2.63x	75.54	75.50	70.09	-	**76.90** ±0.17
80.90% - *mIoU*		15.00 ms	8.77x	68.12	68.35	63.95	65.76	**70.30** ±0.35

Table 1. Image classification/segmentation experiment results in terms of accuracy for image classification on CIFAR-10, CIFAR-100, ImageNet and mean IoU for image segmentation on Cityscapes dataset at different average latency budget values.

Dataset, Model and	Base Model	Average Latency	Speed Gain	Metric (%)				
Base Performance	Latency	Budget per sample		**BranchyNet**	**MSDNet**	**PABEE**	**MAML-stop**	**EENet**
SST-2		150.00 ms	1.27x	92.14	92.17	92.05	-	**92.25** ±0.03
BERT-base w/ 4 exits	189.93 ms	125.00 ms	1.52x	90.86	91.00	90.75	-	**92.09** ±0.05
92.36% - *accuracy*		100.00 ms	1.89x	87.66	87.71	86.99	88.15	**91.58** ±0.60
AgNews		150.00 ms	1.27x	93.20	93.17	93.15	-	**93.85** ±0.01
BERT-base w/ 4 exits	189.93 ms	125.00 ms	1.52x	92.95	92.98	92.57	-	**93.75** ±0.11
93.98% - *accuracy*		100.00 ms	1.89x	85.58	84.93	85.22	93.00	**93.45** ±0.09

Table 2. Sentiment analysis experiment results in terms of accuracy obtained at different average latency budget values.

provided in Eq. 4. To compute the thresholds for these methods, we follow the approach in [13] and assume that the exit assignment of samples will follow a geometric distribution since BranchyNet and PABEE do not provide any guidance on how to set the exit thresholds.

We provide Table 1 for the results on image classification and Table 2 for sentiment analysis tasks. Since MAML-stop requires training the full DNN model for each different budget setting, we obtain the result for one budget value in each experiment. Please note that we refer to the maximum prediction score-based early exiting approach in [13] as MSDNet and the multi-exit CNN architecture with 35 layers as MSDNet35. As demonstrated in Table 1, our system consistently performs better compared to other early exit approaches for image tasks. In addition, we observe that our approach achieves greater performance gains as the budget tightens with improvements ranging from 0.23% to 1.95% compared to the closest competitor method. We observe that MAML-stop performs closest in most scenarios but requires finetuning the full multi-exit model separately for each budget setting, which is not practical in most applications. PABEE only considers the agreement among different early exit classifiers and completely ignores the scores, which causes it to perform worse in most settings. Similar observations are also made for the sentiment analysis task, showing that EENet offers consistent performance improvement for all six benchmarks across different budget settings thanks to optimizing exit scoring and distribution as opposed to manual rule-based approaches. For instance, our approach achieves 93.45% accuracy on AgNews at 100ms/sample while BranchyNet can only achieve 93.25% even at 150ms/sample.

4.3. Analysis of EENet Scheduler Behavior

Analysis of Exit Assignments. Figure 3 provides a comparison of our method with MSDNet on CIFAR-100, with four exits of the respective early exit models. We randomly display samples and the exit location that they were assigned by our system (right) and by MSDNet (left). Images with green/red borders are predicted correctly/incorrectly at the corresponding exit. Our approach obtains the performance gain over MSDNet by allowing more correct predictions to exit earlier at the second exit. In this figure, Group A is correctly predicted by both at the same exit. Group B samples are correctly predicted by both but EENet exits earlier and gains throughput. Likewise, Group D samples are incorrectly predicted by both but EENet exits earlier. EENet exits later for group C samples to achieve the correct prediction while the other method fails. Group E samples are incorrectly predicted by both at the same exit.

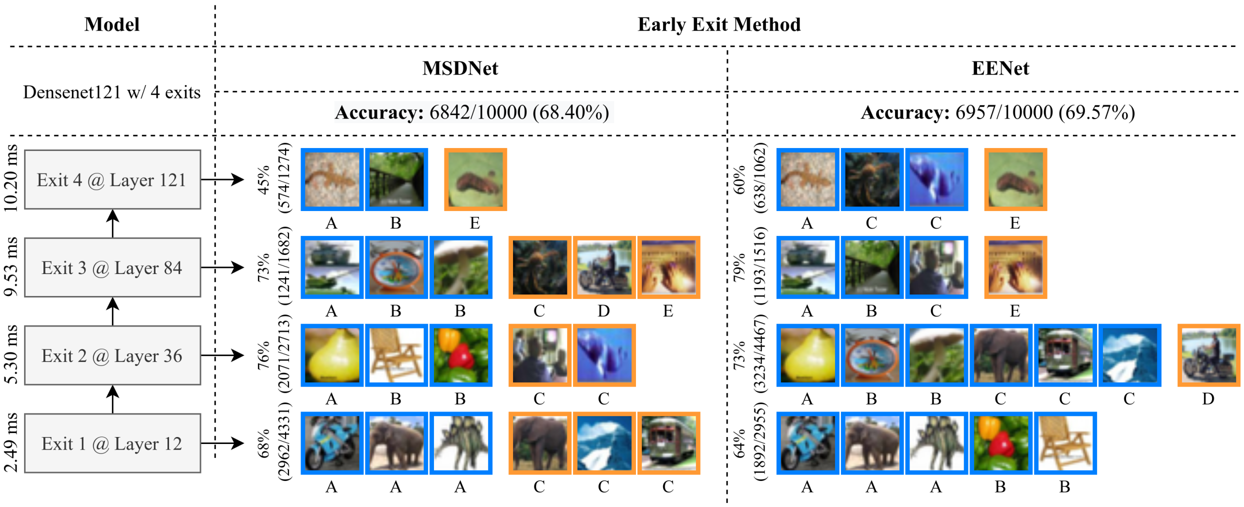

Figure 3. Visual comparison of MSDNet and EENet results on CIFAR-100 for the average latency budget of 6 ms. We illustrate randomly selected twenty samples and the exit location that they were assigned. Images with blue/orange borders are predicted correctly/incorrectly at the corresponding exit. We also report the number of correct predictions and exited samples at each exit.

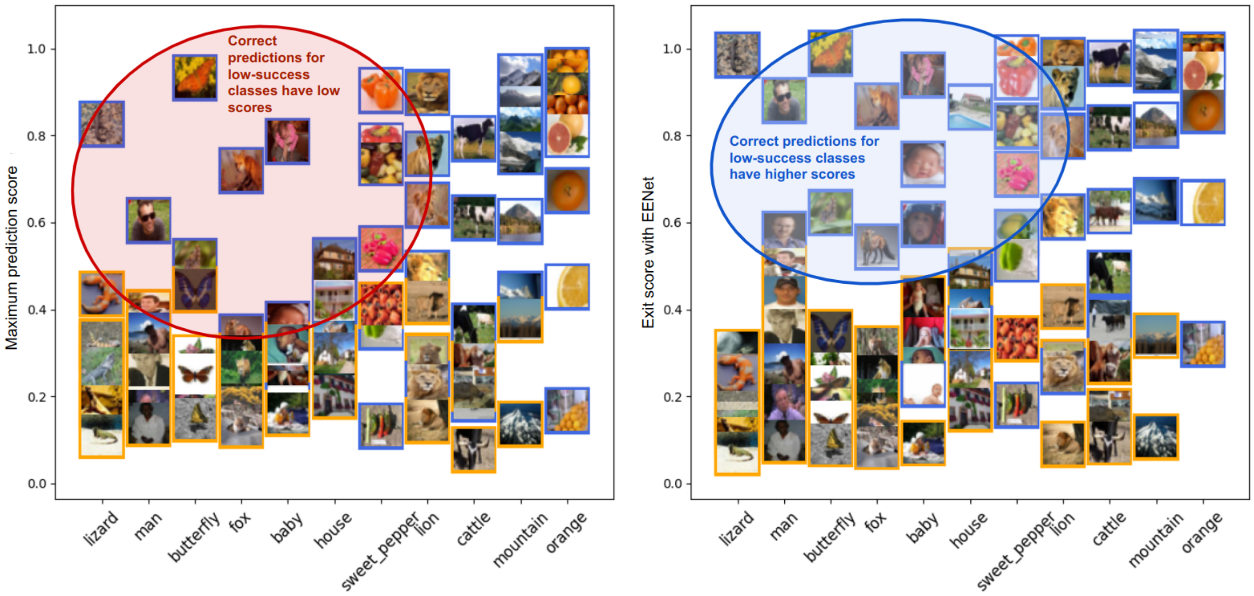

Figure 4. Images with blue/orange borders are predicted correctly/incorrectly at the first exit of DenseNet121 on CIFAR-100 (left: MSDNet, right: EENet). Our approach provides a clearer separation of true and false predictions for all classes, compared to maximum prediction score-based confidence, which is popularly used in the literature. Correct predictions for low-success classes have lower maximum prediction scores whereas their exit scores computed by EENet are higher as desired.

Analysis of Exit Scores. In Figure 4, we analyze examples to show why the exit scores produced by our approach are more effective by comparing with the maximum prediction scoring method for early exit used in MSDNet and others in the literature. Randomly selected ten classes are listed on the x-axis sorted by the accuracy achieved using the full model on the corresponding class. From the left figure, using maximum prediction scores to determine exiting may lead to missing some good early exit opportunities. For example, consider those classes that the DNN model produces relatively low maximum prediction scores (less than 0.7), such as lizard, man, butterfly and fox. Even though the predictor can predict them correctly, the relative confidence is not very high. In comparison, the exit scores obtained by our system reflect the correctness of test predictions more accurately. For example, those classes that have

Model	Exit-1		Exit-2		Exit-3		Exit-4		Exit-5		Base Model	
	#PRMs	Latency	#PRMs	Latency	#PRMs	Latency	#PRMs	Latency	#PRMs	Latency	#PRMs	Latency
ResNet56	0.06M	2.31ms	0.28M	4.15ms	0.96M	4.93ms	-	-	-	-	0.86M	4.77ms
(w/ EENet)	(<1K)	(+0.01ms)	(<1K)	(+0.01ms)	(<1K)	(+0.01ms)	-	-	-	-	-	-
DenseNet121	0.06M	2.49ms	0.25M	5.30ms	0.86M	9.53ms	1.17M	10.20ms	-	-	1.04M	10.03ms
(w/ EENet)	(+5.25K)	(+0.08ms)	(+5.36K)	(0.08ms)	(+5.47K)	(+0.08ms)	(+5.57K)	(+0.08ms)	-	-	-	-
MSDNet35	8.76M	58.95 ms	20.15M	122.99 ms	31.73M	155. 49 ms	41.86M	177.69 ms	62.31M	194.31 ms	58.70M	188.49 ms
(w/ EENet)	(+0.25M)	(+0.72ms)	(+0.25M)	(+0.72ms)	(+0.25M)	(+0.72ms)	(+0.25M)	(+0.72ms)	(+0.25M)	(+0.72ms)	-	-
HRNet-W48	1.59M	4.80 ms	17.14M	19.17 ms	65.96M	138.05 ms	-	-	-	-	63.60M	131.60 ms
(w/ EENet)	(<1K)	(+1.21ms)	(<1K)	(+1.21ms)	(<1K)	(+1.24ms)	-	-	-	-	-	-
BERT-base	45.69M	51.04 ms	67.55M	91.35 ms	89.40M	148.13 ms	111.26M	188.90 ms	-	-	109.90M	183.45 ms
(w/ EENet)	(<200)	(<0.01ms)	(<200)	(<0.01ms)	(<200)	(<0.01ms)	(<200)	(<0.01ms)	-	-	-	-

Table 3. In each row, the top values are the model size (#PRMs) and average inference latency (ms) measured when running the multi-exit model at inference time. The bottom values in parentheses are the additional cost of model size and latency measured when running the EENet adaptive scheduler to predict the exit scores for each test example. The last column is the model size without early exits.

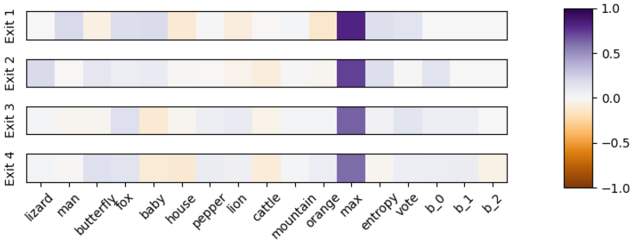

Figure 5. Scheduler weights (for randomly selected ten classes and other inputs) of exit scoring functions at each exit at 6.5 ms/sample budget setting for CIFAR-100.

lower maximum prediction scores on the true predictions, such as lizard, man, butterfly and fox, will have high exit scores in EENet as shown in the right figure highlighted in the blue oval. To interpret what contributes most during exit score computation, we also analyze the scheduler weights at each exit for CIFAR-100 experiments in Figure 5. The results show that the maximum score contributes the most and higher weights for low-success classes also support the observations.

4.4. Computational Cost Analysis

Table 3 reports the number of parameters (#PRMs) and latency of the models used in the experiments until each exit for four multi-exit DNNs. For each model, we also provide the additional computational cost of EENet in employing the budgeted adaptive early exit policy. The increase of latency caused is negligible ($< 0.5\%$) compared to the cost of the forward pass of the original pre-trained DNN model. The scheduler training cost is also very low thanks to lightweight scheduler architectures. For instance, optimizing the schedulers takes less than five minutes in CIFAR experiments on an RTX3060 GPU.

Partial deployment on edge device hierarchies. For the edge application scenarios with strict constraints (storage/RAM limitations), the partial model split until a certain exit can be deployed, with the partial model size meeting the edge deployment constraints. With flexible deployment on heterogeneous edge clients, those test samples with exit scores below the learned exit threshold can be passed to the next-level edge server in the hierarchical edge computing infrastructure, which has a higher computational capacity to continue inference.

5. Conclusion

We present EENet, an early exit scheduling optimization method for adaptive DNN inference. This paper makes two novel contributions. EENet optimizes both exit scoring functions and exit thresholds such that the given inference budget is satisfied while the performance metric is maximized. As opposed to previous manually defined heuristics-based early exit techniques, including task-specific architectural optimization techniques, our approach is model-agnostic and can easily be used in different deep learning tasks with multi-exit neural networks, ranging from computer vision to NLP applications. In addition, the efficient structure of schedulers provides flexibility compared to the methods that require full model finetuning for varying budget settings. Extensive experiments on six benchmarks demonstrate that EENet offers significant performance gains compared to existing methods, especially under tighter budget regimes and large DNN models.

Acknowledgements

This research is partially sponsored by a grant from the CISCO Edge AI program, and the NSF CISE grants 2302720, 2312758, and 2038029.

References

[1] Xinshi Chen, Hanjun Dai, Yu Li, Xin Gao, and Le Song. Learning to stop while learning to predict. In *ICML*, pages 1520–1530, 2020. 2, 5

[2] 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 *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 5

[3] Xin Dai, Xiangnan Kong, and Tian Guo. Epnet: Learning to exit with flexible multi-branch network. In *Proceedings of the 29th ACM International Conference on Information amp; Knowledge Management*, CIKM '20, page 235–244, New York, NY, USA, 2020. Association for Computing Machinery. 2

[4] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009. 5

[5] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. 2

[6] Maha Elbayad, Jiatao Gu, Edouard Grave, and Michael Auli. Depth-adaptive transformer. In *International Conference on Learning Representations*, 2020. 1

[7] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations*, 2019. 1

[8] Asghar Gholami, Sehoon Kim, Dong Zhen, Zhewei Yao, Michael Mahoney, and Kurt Keutzer. *A Survey of Quantization Methods for Efficient Neural Network Inference*, pages 291–326. 01 2022. 1

[9] Sayan Ghosh, Karthik Prasad, Xiaoliang Dai, Peizhao Zhang, Bichen Wu, Graham Cormode, and Peter Vajda. Pruning compact convnets for efficient inference, 2022. 1

[10] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT Press, 2016. 1

[11] Kaiming He, X. Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016. 2, 5, 11

[12] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network, 2015. 1

[13] Gao Huang, Danlu Chen, Tianhong Li, Felix Wu, Laurens van der Maaten, and Kilian Weinberger. Multi-scale dense networks for resource efficient image classification. In *International Conference on Learning Representations*, 2018. 1, 2, 5, 6, 11

[14] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2261–2269, 2017. 2, 11

[15] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. 11

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

[17] Stefanos Laskaridis, Alexandros Kouris, and Nicholas D. Lane. Adaptive inference through early-exit networks: Design, challenges and directions. In *Proceedings of the 5th International Workshop on Embedded and Mobile Deep Learning*, EMDL'21, page 1–6, New York, NY, USA, 2021. Association for Computing Machinery. 1

[18] Hao Li, Hong Zhang, Xiaojuan Qi, Ruigang Yang, and Gao Huang. Improved techniques for training adaptive deep networks. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 1891–1900, 2019. 1

[19] Shuang Li, Jinming Zhang, Wenxuan Ma, Chi Harold Liu, and Wei Li. Dynamic domain adaptation for efficient inference. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2021. 2

[20] Ziqian Lin, Sreya Dutta Roy, and Yixuan Li. Mood: Multi-level out-of-distribution detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021. 2

[21] Zhuang Liu, Zhiqiu Xu, Hung-Ju Wang, Trevor Darrell, and Evan Shelhamer. Anytime dense prediction with confidence adaptivity. In *International Conference on Learning Representations*, 2022. 11

[22] Mary Phuong and Christoph Lampert. Distillation-based training for multi-exit architectures. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 1355–1364, 2019. 1

[23] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642, Seattle, Washington, USA, Oct. 2013. Association for Computational Linguistics. 5

[24] Surat Teerapittayanon, Bradley McDanel, and H.T. Kung. Distributed deep neural networks over the cloud, the edge and end devices. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pages 328–339, 2017. 1

[25] Surat Teerapittayanon, Bradley McDanel, and H. T. Kung. Branchynet: Fast inference via early exiting from deep neural networks. *2016 23rd International Conference on Pattern Recognition (ICPR)*, pages 2464–2469, 2016. 1, 2, 5

[26] T. Veniat and L. Denoyer. Learning time/memory-efficient deep architectures with budgeted super networks. In *2018 IEEE/CVF Conference on Computer Vision and Pattern*

Recognition (CVPR), pages 3492–3500, Los Alamitos, CA, USA, jun 2018. IEEE Computer Society. 1

[27] J. Wang, K. Sun, T. Cheng, B. Jiang, C. Deng, Y. Zhao, D. Liu, Y. Mu, M. Tan, X. Wang, W. Liu, and B. Xiao. Deep high-resolution representation learning for visual recognition. *IEEE Transactions on Pattern Analysis amp; Machine Intelligence*, 43(10):3349–3364, oct 2021. 2

[28] Le Yang, Yizeng Han, Xi Chen, Shiji Song, Jifeng Dai, and Gao Huang. Resolution adaptive networks for efficient inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2020. 1

[29] Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015. 5

[30] Wangchunshu Zhou, Canwen Xu, Tao Ge, Julian McAuley, Ke Xu, and Furu Wei. Bert loses patience: Fast and robust inference with early exit. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 18330–18341. Curran Associates, Inc., 2020. 1, 2, 5