ReX: An Efficient Approach to Reducing Memory Cost in Image Classification

Xuwei Qian, Renlong Hang[*], Qingshan Liu

Nanjing University of Information Science and Technology, Nanjing, China
20191223049@nuist.edu.cn, renlong_hang@163.com, qsliu@nuist.edu.cn

Abstract

Exiting simple samples in adaptive multi-exit networks through early modules is an effective way to achieve high computational efficiency. One can observe that deployments of multi-exit architectures on resource-constrained devices are easily limited by high memory footprint of early modules. In this paper, we propose a novel approach named recurrent aggregation operator (ReX), which uses recurrent neural networks (RNNs) to effectively aggregate intra-patch features within a large receptive field to get delicate local representations, while bypassing large early activations. The resulting model, named ReXNet, can be easily extended to dynamic inference by introducing a novel consistency-based early exit criteria, which is based on the consistency of classification decisions over several modules, rather than the entropy of the prediction distribution. Extensive experiments on two benchmark datasets, i.e., Visual Wake Words, ImageNet-1k, demonstrate that our method consistently reduces the peak RAM and average latency of a wide variety of adaptive models on low-power devices.

Introduction

The inference-time computational demands of deep neural networks (DNNs) are increasing, owing to the "going deeper" (Szegedy et al. 2015) strategy for improving accuracy. This strategy has enabled breakthroughs in many tasks, such as image classification (Krizhevsky, Sutskever, and Hinton 2012) or speech recognition (Hinton et al. 2012), at the price of costly inferences. The increased computational complexity makes various very deep models inappropriate for resource-constrained processors, which are common in devices such as smartphones, wearables, and drones.

To address this issue, several recent works have shown that some samples, which are already correctly classified by shallow networks, do not necessitate the extra complexity (Bolukbasi et al. 2017; Li et al. 2019; Hu et al. 2019; Graves 2016). This observation has stimulated the study of input-adaptive mechanisms, in particular, multi-exit architectures (Teerapittayanon, McDanel, and Kung 2016; Huang et al. 2018; Kaya, Hong, and Dumitras 2019; Hu et al. 2020; Yang et al. 2020). Exiting easy samples in adaptive multi-exit networks through early modules is an effective way to achieve high computational efficiency. However, the early modules tend to have large intermediate activations and thus require large working memory for inference (as shown in Figure 1 (left)). Since memory takes up a large portion of the device chip and has high continuous power requirements, it tends to be the most constrained resource on low-power devices (Jouppi et al. 2017). Most low-power ARM Cortex-M* microcontrollers have less than 256KB of RAM.

To reduce working memory, a straightforward solution is to use pooling or stride convolution with large stride length to reduce the size of early activations, but this can result in poor performance of subsequent classifiers as features important to the task are washed out in early modules. Therefore, their use is limited to small receptive fields, usually no larger than 3 × 3, and activation maps cannot be actively reduced by aggregating larger receptive fields.

In this paper, we are interested in whether this peak RAM can be effectively reduced without sacrificing accuracy. We develop a memory-efficient recurrent aggregation operator (ReX), aiming at bypassing large early activations while using RNNs to effectively aggregate intra-patch features within a large receptive field to obtain delicate local representations. Specifically, our method first takes a quick traverse along each row and column with the first RNN to get a cheap summary of the row or column. Then we fuse them through a linear projection to produce coarse global information. Finally, the second RNN summarizes the outputs of the first RNN bi-directionally. A ReX layer consists of a single recurrent aggregation operator with stride s and allows rapid down-sampling of images and activation maps. Additionally, a single layer of ReX is most efficient when used to replace multiple memory-intensive blocks in the early modules of the network where the activation map sizes are large and thus require the most memory and computation. The resulting model, named ReXNet, can be trained efficiently on GPUs and has high inference speed on mobile devices. An illustration of ReXNet is shown in Figure 1 (right).

Next, we further extend ReXNet so that it can stop inference dynamically based on the input sample. To implement this idea, a natural solution is to calculate softmax prediction scores as confidence for the internal classifiers and then set a threshold to make the classifier exit early. However, we should note that prediction probabilities as confidence metrics, may be unreliable and usually lead to significant

[*] Corresponding author

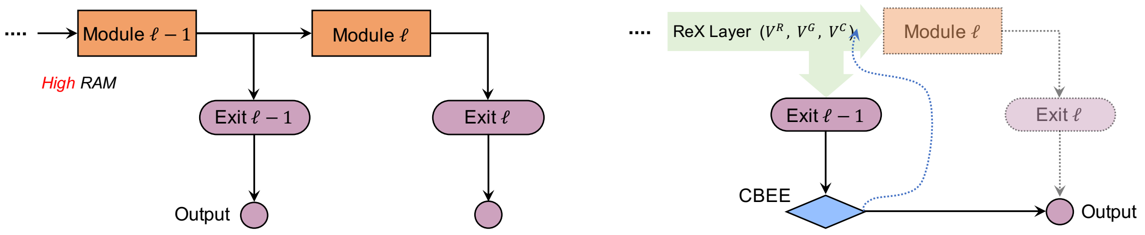


Figure 1: Standard adaptive multi-exit network *(left)* and our proposed ReXNet *(right)*. Visual inference on low-power devices is easily limited by the high peak memory usage of early modules, while ReX aims to bypass intermediate large spatial resolution activations. The dashed lines and shaded modules are not executed, conditioned on the decisions made by the CBEE.

performance degradation. Therefore, we propose a novel Consistency-Based Early Exit (CBEE) algorithm. Its stopping criteria are based on the consistency of classification decisions over several modules, rather than the entropy of the prediction distribution. Specifically, we make the model stop inference when the intermediate predictions of the internal classifier remain unchanged for k times consecutively, where k is a predefined count. We employ this algorithm (CBEE) to reason about ReXNet and show a better accuracy-speed trade-off than existing methods.

We evaluate the performance of our method on Visual Wake Words (Chowdhery et al. 2019) and ImageNet with various adaptive networks. We also benchmark the average latency of ReXNet on an ARM processor and an iPhone. Experimental results show that ReXNet by itself consistently outperforms all the baselines by large margins, while CBEE further improves the efficiency. For example, when the MobileNetV2 (6 exits) is used as the base network, ReXNet has up to $11.5\times$ and $2\times$ less peak RAM and average latency than the original models when achieving the same level of accuracy, respectively.

Related Work

Input-Adaptive Network. Adjusting the network architecture to the corresponding input has been recently studied in the computer vision domain. There are two types of input adaptive DNNs: layer-skipping networks (Wang et al. 2018; Figurnov et al. 2016) and multi-exit structures. During the inference, the former dynamically skip a certain part of the model to reduce the number of computations. This mechanism can be used only for ResNet-based architectures as they facilitate skipping within a network. On the other hand, an early-exiting strategy is adopted in multi-exit architectures (Huang et al. 2018; Kaya, Hong, and Dumitras 2019) for resource-efficient object recognition, which classifies easier inputs and gives output in earlier modules. However, we should note that this inference process may be limited by the peak RAM of early modules. In other words, optimizing computation/memory efficiency is a key to the deployment of adaptive networks on devices powered by tiny microcontrollers, and it is the primary focus of this paper.

Memory Optimization. Previous works on memory optimized inference manipulate existing convolution operators by reordering computations (Cho and d 2017; Lai, Suda, and Chandra 2018) or performing them in place (Gural and Murmann 2019). However, most of these methods provide relatively small memory savings and are validated on low-resolution images like CIFAR-10 (Krizhevsky and Hinton 2009). Max pooling, average pooling, or stride convolution (LeCun, Bengio, and Hinton 2015) is the most direct method, but large-stride operations often result in severe performance losses. Channel pruning (He, Zhang, and Sun 2017) is an approach that tries to reduce memory footprint by pruning out multiple convolution kernels in every layer. The proposed ReX method differentiates itself from these approaches in that we focus on replacing the expensive early modules in adaptive architectures. Hence, our method is compatible with them.

Early Exiting Mechanism. Ideally, a multi-exit network stops when it reaches a correct prediction at an exit point. However, for unseen samples, this is impractical as the ground-truth labels are unknown. Previous works suggested that the module exits inference when the confidence of the intermediate prediction is greater than a given threshold. BranchyNet (Teerapittayanon, McDanel, and Kung 2016) calculated the entropy of the prediction probability distribution to enable early exit. Shallow-Deep Nets (Kaya, Hong, and Dumitras 2019) and MSDNet (Huang et al. 2018) leveraged the softmax scores of classifiers as a proxy for the confidence. Recently, Zhou et al. (Zhou et al. 2020) adapted this approach in adversarial training to improve the adversarial robustness of DNNs. Unlike confidence-based methods, our approach stops inference when the intermediate predictions of the internal classifier remain unchanged over a predefined count k.

RNN-based Approaches. Many existing works integrate the RNN into image processing systems, especially in sequence-related tasks. For example, ReNet (Visin et al. 2015) uses a RNN based layer as a replacement for a convolution layer. ReSeg (Visin et al. 2016) extends the ReNet architecture to deal with the task of semantic segmentation. ION (Bell et al. 2016) uses a ReNet based layer for extracting context information outside the region of interest. The PiCANet (Liu, Han, and Yang 2018) uses it as a global attention network for salient detection. Some other works also use RNNs in their architectures but only to model certain sequences in the respective tasks rather than performing input-adaptive inference (Acuna et al. 2018; Saha et al. 2020; Wang et al. 2016). Details of the differences between ReNet and our ReX will be discussed in the next section.

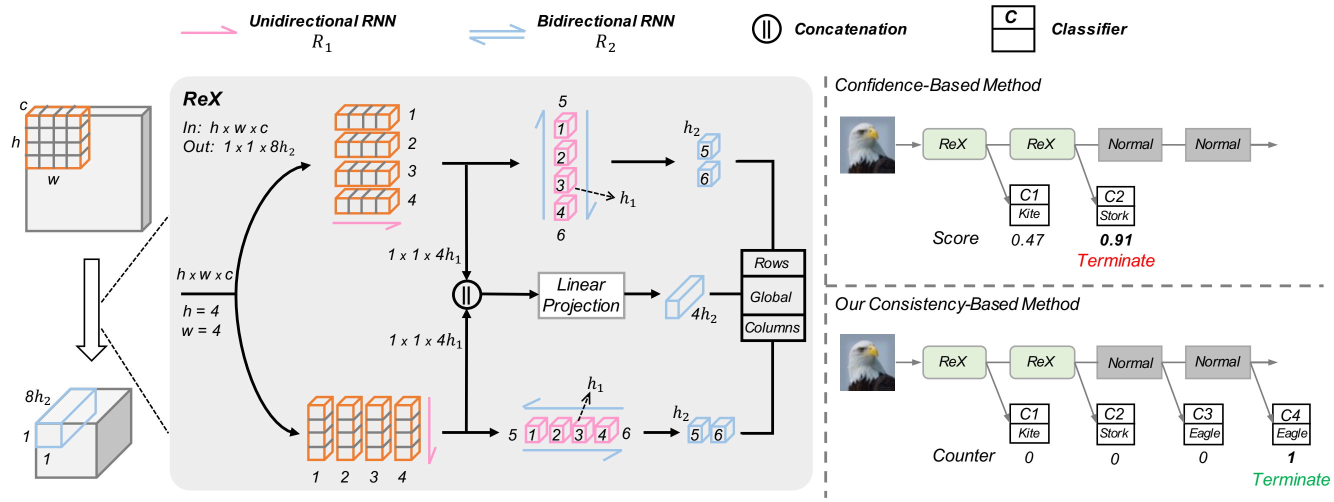


Figure 2: Left and right panels show our ReX and CBEE method, respectively. Left shows the overview of ReX, which applied to patches of size $h \times w$ with stride s. It summarizes the patch into a vector of size $8h_2$ through two RNNs and a linear projection. Right shows the comparison between a prediction score-based early exit (threshold is set to 0.9), and our consistency-based early exit (count $k = 1$). Adaptive network incorrectly exits based on the prediction score. CBEE considers multiple classifiers and exits with a correct prediction.

Efficient Inference with ReX

Existing adaptive networks rely on exiting simple samples through early modules to achieve high efficiency while neglecting the fact that visual inference on low-power devices is usually limited by the high peak memory of early modules. To this end, we propose a novel recurrent aggregation operator (ReX) to bypass early large spatial resolution activations without sacrificing accuracy.

In this section, we first describe the details of the proposed ReX, and then implement it to build our memory-efficient networks.

Recurrent Aggregation Operator

We first give an overview of ReX (Figure 2 (left)). The whole structure consists of three components: a unidirectional RNN $\boldsymbol{R_1}$ of hidden dimension h_1, a bidirectional RNN $\boldsymbol{R_2}$ with hidden dimension h_2, and a linear projection $\boldsymbol{f}_{\mathrm{L}}$.

Formally, given an activation patch $\mathbf{X}_{\text{in}}$ of size $h \times w \times c$, where c is the number of channels, ReX directly takes it as input and produces a summary $\mathbf{X}_{\text{out}}$ of size $1 \times 1 \times 8h_2$, i.e.,

$$\mathbf{X}_{\text{in}} \leftarrow [\mathbf{x}^{1,1}, \mathbf{x}^{1,2}, \ldots, \mathbf{x}^{h,w}], \quad \mathbf{x}^{i,j} \in \mathcal{R}^c, \tag{1}$$

$$[\boldsymbol{V}^{\mathrm{R}}, \boldsymbol{V}^{\mathrm{G}}, \boldsymbol{V}^{\mathrm{C}}] = \mathbf{ReX}(\mathbf{X}_{\text{in}}), \tag{2}$$

$$\mathbf{X}_{\text{out}} \leftarrow [\boldsymbol{V}^{\mathrm{R}}, \boldsymbol{V}^{\mathrm{G}}, \boldsymbol{V}^{\mathrm{C}}] \tag{3}$$

where i and j refer to the location of the feature. In the following, we describe the generation process of features $\{\boldsymbol{V}^{\mathrm{R}}, \boldsymbol{V}^{\mathrm{G}}, \boldsymbol{V}^{\mathrm{C}}\}$ in detail. $\boldsymbol{R_1}$ first takes a quick traversal along each row and column, and summarizes patches horizontally and vertically to obtain cheap row and column information ($\boldsymbol{e}^h$ and $\boldsymbol{e}^w$). A linear projection $\boldsymbol{f}_{\mathrm{L}}$ directly takes their concatenation as inputs and produces a coarse global feature vector $\boldsymbol{V}^{\mathrm{G}}$:

$$\boldsymbol{e}_i^h \leftarrow \boldsymbol{R_1}(\mathbf{x}^{i,1\le j\le w}), \text{ for } i = 1, 2, \cdots, h, \tag{4}$$

$$\boldsymbol{e}_j^w \leftarrow \boldsymbol{R_1}(\mathbf{x}^{1\le i\le h,j}), \text{ for } j = 1, 2, \cdots, w, \tag{5}$$

$$\boldsymbol{V}^{\mathrm{G}} = \boldsymbol{f}_{\mathrm{L}}([\boldsymbol{e}^h_{1\le i\le h}, \boldsymbol{e}^w_{1\le j\le w}]), \tag{6}$$

where $[\cdot]$ denotes the vector concatenation. The second RNN $\boldsymbol{R_2}$ is leveraged to take full advantage of the cheap summary of each row or column for learning more accurate bidirectional context representations, and hence we traverse the outputs ($\boldsymbol{e}^h$ and $\boldsymbol{e}^w$) of $\boldsymbol{R_1}$ bi-directionally through $\boldsymbol{R_2}$:

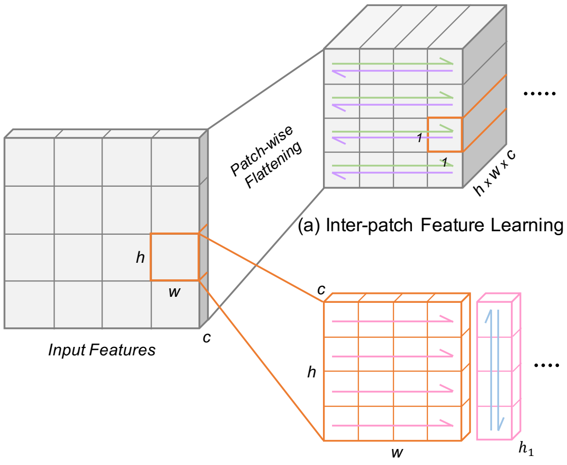
$$\boldsymbol{V}^{\mathrm{R}} \leftarrow \boldsymbol{R_2}(\boldsymbol{e}^h_{1\le i\le h}), \tag{7}$$

$$\boldsymbol{V}^{\mathrm{C}} \leftarrow \boldsymbol{R_2}(\boldsymbol{e}^w_{1\le j\le w}), \tag{8}$$

where $\boldsymbol{V}^{\mathrm{R}}$ and $\boldsymbol{V}^{\mathrm{C}}$ refer to the fine row and column feature vectors, respectively. In our implementation, a ReX layer consists of a single ReX sliding over an input activation map and takes as input another parameter: the stride length. It is noteworthy that there are only two different RNNs ($\boldsymbol{R_1}$ & $\boldsymbol{R_2}$) in a ReX layer, thus it can be trained efficiently on modern GPU devices, and the bulk of computation can be done in parallel.

Formally, we simply use GRU (Cho et al. 2014) for these two RNNs in our method, as done in (Visin et al. 2015). Note that the proposed ReX is independent of the chosen recurrent unit. More complex unit designs have the potential to improve performance but are not the focus of this work.

Learning Delicate Local Representations. ReX is a fine-grained feature learning mechanism using RNNs. Prior proposed ReNet (Visin et al. 2015) captures global context by aggregating inter-patch features. Different from it, our ReX uses overlapping patches and captures fine-grained local representations by aggregating intra-patch features within a large



(a) Inter-patch Feature Learning

(b) Intra-patch Feature Learning

Figure 3: Inter-patch feature learning (a) in ReNet and Intra-patch feature learning (b) in our ReX. The colored arrows represent the sweeping path of RNNs, and arrows of the same color share weights.

receptive field. In other words, ReX and ReNet can be used in combination. Each layer can simultaneously learn local (Intra-patch) and global (Inter-patch) representations.

Figure 3 illustrates the differences and connections between ReNet (top) and our ReX (bottom). Notably, the global and local contexts do not depend on overlapping or non-overlapping patches but are captured by the constituent operations in a layer. ReNet applies each horizontal and vertical RNN to process the entire image in one pass and the resulting hidden states provide the output. In contrast, ReX is applied patch-wise and only the final states from each RNN run are used for output. Moreover, in ReNet, each patch is flattened and passed as a single input to an RNN which leads to a huge matrix multiplication operation.

Architecture Design

Based on the proposed ReX layer, we experiment on two recent techniques: Shallow-Deep Networks (SDNs) (Kaya, Hong, and Dumitras 2019) and MSDNets (Huang et al. 2018). Considering the generic SDN architecture in Figure 1 (left), we can change the early modules with ReX to bypass large early activations. Specifically, here we replace multiple memory-intensive blocks in the early module with a single ReX layer to quickly reduce the activation size. It can achieve a better trade-off between memory cost and accuracy. For example, in MobileNetV2 (6 exits), we can replace 6 bottleneck blocks-spanning 18 layers with a single ReX to reduce the activation size from 112 × 112 × 32 to 28 × 28 × 64 (shown in Table 1). The replacement ReX can be performed patch-by-patch without re-computation, thus reducing the need to store the entire activation map over the image. Due to its general formulation, ReX can replace the feature reduction layer in all classifiers. The resulting model, called ReXNet, has an extremely low memory footprint and latency, and is simple to train.

Input	Operator	t	c	n	s	e
$224^2 \times 3$	conv2d 3×3	1	32	1	2	-
$112^2 \times 32$	ReX Layer	1	64	1	4	-
$28^2 \times 64$	bottleneck	6	64	4	2	2
$14^2 \times 64$	bottleneck	6	96	3	1	1
$14^2 \times 96$	bottleneck	6	160	3	2	1
$7^2 \times 160$	bottleneck	6	320	1	1	1
$7^2 \times 320$	conv2d 1×1	1	1280	1	1	1

Table 1: ReX-MobileNetV2 (6 exits) : ReX Layer with patch-size 6×6 and hidden sizes $h_1 = 16$, $h_2 = 8$ is used. Each line describes a sequence of 1 or more identical layers, repeated n times.The first layer of each sequence has stride s and the rest use stride 1. The number of internal exits in this sequence and the expansion factor are denoted by e and t, respectively.

Learning to Stop Dynamically

In this section, we describe our CBEE to enable ReXNet to stop inference dynamically. CBEE captures the inner agreement between earlier and later internal classifiers and exploits multiple classifiers for inference. In addition, our method supports regression tasks, which were overlooked in prior work.

Inference

The inference process of CBEE is illustrated in Figure 2 (right bottom). We set up the common inference model as a network that is composed of m predictors. These intermediate classifiers/regressors $C_1 \ldots C_m$ are attached at varying depths of the model to predict a class label distribution $\mathbf{p}$ for classification or a value g for regression (we assume the output dimension is 1 for brevity). Given an input image x, the model generates a set of m hidden states as the input of the classifiers/regressors, i.e.,

$$[h_1, \ldots, h_m] = f(x; \theta) = [f_1(x; \theta_1), \ldots, f_m(x; \theta_m)], \quad (9)$$

where f_i and θ_i $(i = 1, \ldots, m)$ represent the transformation learned by the i-th hidden state h_i and its corresponding parameters, respectively. Note that θ_i's have shared parameters here.

Then, we use its internal classifier/regressor to output a distribution or value as an intermediate prediction $\mathbf{p}_i = C_i(h_i)$ or $g_i = C_i(h_i)$. We use a counter N to store the number of times that the predictions remain "unchanged". For classification, this can be represented by:

$$N_i = \begin{cases} N_{i-1} + 1 & \arg\max(\mathbf{p}_i) = \arg\max(\mathbf{p}_{i-1}), \\ 0 & \arg\max(\mathbf{p}_i) \neq \arg\max(\mathbf{p}_{i-1}). \end{cases} \quad (10)$$

While for regression, we have:

$$N_i = \begin{cases} N_{i-1} + 1 & |g_i - g_{i-1}| < \eta, \\ 0 & |g_i - g_{i-1}| \geq \eta. \end{cases} \quad (11)$$

where η is a pre-defined threshold. If $N_i = k$, the sequential process stops, and $\mathbf{p}_i$ or g_i will be outputted as the final prediction. Otherwise, we use the final classifier C_m for prediction.

Model	Method	Standard (Chowdhery et al. 2019)				Memory Optimised	
		Top-1 Accuracy	Parameters	Peak RAM	FLOPs	Peak RAM	FLOPs
ResNet34 (7 exits)	Original	75.0 ± 0.20%	22M	3.1MB	3.8G	0.7MB	28.7G
	ReX Layer	**75.8 ± 0.12%**	**20M**	**0.4MB (↓7.75x)**	**3.0G**	0.3MB	3.4G
MobileNetV2 (6 exits)	Original	71.7 ± 0.24%	3.4M	2.3MB	0.3G	0.4MB	1.1G
	ReX Layer	**72.8 ± 0.11%**	**3.1M**	**245KB (↓11.5x)**	**0.24G**	0.2MB	0.28G
DenseNet121 (21 exits)	Original	75.0 ± 0.05%	8M	3.1MB	2.8G	1.7MB	25.2G
	ReX Layer	**76.3 ± 0.07%**	**7M**	**0.8MB (↓3.88x)**	**1.0G**	0.6MB	1.7G
MSDNet (6 exits)	Original	69.6 ± 0.16%	18M	2.2MB	0.64G	0.9MB	17.5G
	ReX Layer	**71.1 ± 0.03%**	**12.7M**	**0.5MB (↓4.40x)**	**0.28G**	0.4MB	0.39G

Table 2: Performance of different networks with and without a single ReX Layer on ImageNet. For clear comparisons, the accuracy in this table is achieved on the optimal threshold or count k.

In Figure 2 (right top), prediction confidence-based early exit relies on the softmax score, and the second classifier outputs a high confidence score and incorrectly terminates inference. Prior work has shown that predicted probability distribution—*softmax scores*—makes an unreliable metric, as it is over-confident towards a single class (Szegedy et al. 2014; Jiang et al. 2018). In this regard, CBEE comprehensively considers results from multiple classifiers, so it can prevent the effect of errors from one single classifier.

Training

A straightforward way to train an adaptive network is to train the submodules sequentially. However, this method is far from optimal due to the conflict between two optimization goals: to learn discriminative features for the current predictor, and to maintain necessary information for generating high-quality features for later predictors (Huang et al. 2018). A more effective training strategy is to jointly optimize all the submodules. So, we train the model to minimize a weighted cumulative loss function $\mathcal{L}$:

$$\mathcal{L} = \frac{\sum_{j=1}^{m} j \cdot \ell_j}{\sum_{j=1}^{m} j} \tag{12}$$

Here, the weighted average can correspond to the relative inference cost of each internal predictor, and ℓ_j denotes the loss function for the j-th classifier/regressor. For classification, the loss function ℓ_i for classifier C_i is computed by:

$$\ell_i = \frac{1}{|\mathcal{D}_{\text{train}}|} \sum_{(\boldsymbol{x},y)\in\mathcal{D}_{\text{train}}} L_{\text{CE}}(\mathbf{p}_i, y) \tag{13}$$

where $\mathcal{D}_{\text{train}}$ is the training set, y denotes the label corresponding to $\boldsymbol{x}$. We use the standard cross-entropy loss function $L_{\text{CE}}(\cdot)$ during training. For regression, given a ground truth $\hat{g}_i$, the loss is instead calculated by a mean squared error:

$$\ell_i = \frac{1}{|\mathcal{D}_{\text{train}}|} \sum_{(\boldsymbol{x},\hat{g}_i)\in\mathcal{D}_{\text{train}}} (g_i - \hat{g}_i)^2 \tag{14}$$

Experiments

In this section, we empirically evaluate the effectiveness of the proposed ReX and CBEE on various devices, and give ablation studies.

Dataset. Our experiments are based on two widely-used visual datasets: (1) Visual Wake Words is a binary classification dataset proposed by (Chowdhery et al. 2019). The dataset contains a total of 115K training images and 8K validation images. We study this dataset because it presents a relevant use case for computer vision on realistic tiny devices; (2) ImageNet (Deng et al. 2009) is a 1000-class dataset from ILSVRC2012, with 1.2 million images for training and 50000 images for validation.

Architectures and Hyper-parameters. To demonstrate the effectiveness of our approach, we experiment on two recent techniques: Shallow-Deep Networks (SDNs) (Kaya, Hong, and Dumitras 2019) and MSDNets (Huang et al. 2018). These architectures were designed for different purposes: SDNs are generic and can convert any DNN into a multi-exit model, and MSDNets are custom-designed for efficiency. After every two blocks, an internal classifier is added. We evaluate an MSDNet architecture (6 exits) and three SDN architectures, based on ResNet-34 (He et al. 2016) (7 exits), MobileNetV2 (Sandler et al. 2018) (6 exits), and DenseNet121 (Huang et al. 2017) (21 exits). To make a fair comparison, the internal classifiers and their insertions are the same in both baselines and our approach.

Metrics. For clarity, we use average FLOPs on the whole test dataset as the metric to measure the computational cost of a network. As for the classification performance, we report the cumulative Top-1 accuracy on all classifiers. In other words, we consider a sample to be correctly classified if there is at least one correct prediction in all classifiers. In addition, we report the peak RAM usage of various networks during the inference.

Main Results

Effectiveness of the ReX layer. We first test the compatibility of ReX with different networks. Table 2 shows that ReX based models achieve slightly better performance while significantly reducing the required computation and peak RAM. Next, we study the memory-optimized method (Gural and Murmann 2019). DNNs have a high computational cost and such re-compute intensive optimizations make the network infeasible even for large devices, e.g., ResNet34 (7 exits)

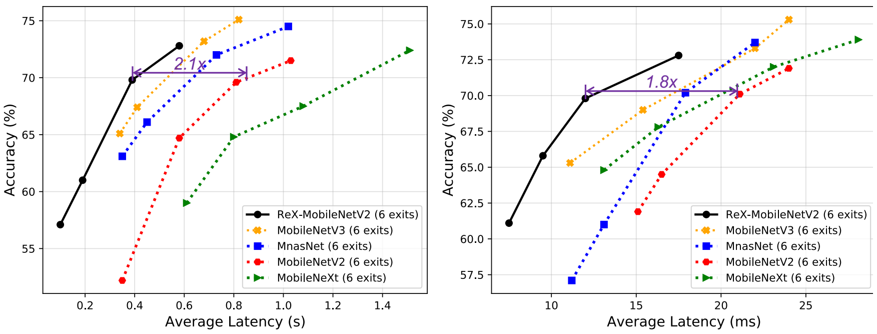


Figure 4: Top-1 accuracy v.s. inference latency on ImageNet. The inference speed is measured on an ARM processor *(left)* and an iPhone 12 *(right)*. Our method is implemented with the predefined count $k \in \{1, 2, 3, 4\}$.

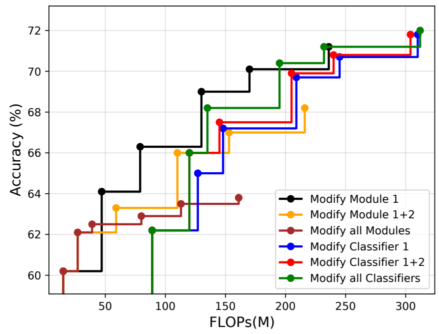


Figure 5: Performance of modifying varying modules and different classifiers with a single ReX layer.

Method	FLOPs	Params	VWW	ImageNet
Confidence-based (2018; 2019)				
Original	301M	3.4M	90.3%	71.7%
MixedPool (2016)	**213M**	**3.1M**	86.7%	65.4%
GFGP (2019)	239M	3.2M	87.4%	66.2%
LiftPool (2021)	254M	3.3M	88.3%	68.3%
Stride Conv	267M	3.3M	88.0%	68.9%
ReNet	295M	3.5M	87.8%	66.8%
ReX Layer	246M	**3.1M**	89.7%	71.1%
Entropy-based (2016)				
ReNet	295M	3.5M	87.6%	66.7%
ReX Layer	246M	**3.1M**	89.4%	69.9%
Consistency-based (Ours)				
ReNet	295M	3.5M	89.3%	68.5%
ReX Layer	246M	**3.1M**	91.0%	72.8%
ReNet + ReX	327M	3.5M	**91.4%**	**73.5%**

Table 3: Comparison of Top-1 accuracy with different downsampling operators and exit criteria on Visual Wake Words (VWW) and ImageNet. Here we use MobileNetV2 (6 exits) as the base network.

requires 28.7 GFLOPs in this scheme (Table 2).

Semantically, ReX is a generalized pooling operator, so we compare ReX with other downsampling methods. We used them to modify the early module of a network to reduce the size of image by a factor of 4×4. In addition, we also compare ReX and ReNet (Visin et al. 2015) as a pooling layer. The results are shown in Table 3. It can be observed that our method outperforms the alternative baselines by large margins in terms of efficiency. For example, on ImageNet, ReX achieves 2.8% higher mAP (71.1% v.s.68.3%) than the strongest baseline, LiftPool (Zhao and Snoek 2021), with the same memory usage.

Improvements from Consistency-based Early Exit. In Table 3, we also compare the results of CBEE with the confidence-based method (Kaya, Hong, and Dumitras 2019; Huang et al. 2018) and entropy-based mechanism (Teerapittayanon, McDanel, and Kung 2016). It is clear that considering the consistency of classification decisions leads to significantly better performance. Moreover, CBEE can achieve a considerably better trade-off between speed and accuracy by adjusting predefined count k, which we will describe below.

Experiments on an ARM processor and an iPhone. Since the proposed ReX is designed for edge devices, we investigate the practical inference speed of our method on an ARM processor[1] and an iPhone 12 (with Apple A14 Bionic) using Pytorch Mobile[2]. The single-thread mode with batch size 1 is used following (Howard et al. 2019). We first measure the time consumption of obtaining the prediction at each possible exit, and then take the weighted average according to the number of validation samples exiting at each exit. The results are shown in Figure 4. We change the count within $k \in \{1, 2, 3, 4\}$, and plot the corresponding Accuracy v.s. Latency relationships in black dots. We also present the variants of baselines with the same SDN architectures. One can observe that ReX effectively accelerates the inference of MobileNetV2 (6 exits).

Modification for modules or classifiers. Note that here we replace varying modules, and feature reduction layer of different classifiers. We use MobileNetV2 (6 exits) as base models and the performance of each classifier (colored dots) is presented in Figure 5. The figure suggests that changing any intermediate classifier can achieve better performance, while replacing early module one leads to better efficiency with an insufficient computational budget and without sacrificing later accuracy. ReX is most effective when used to replace early module one where the activation sizes are large, and hence, require the most memory and computation. One can observe the later classifiers suffer from a severe degradation (even more than 10%) when more modules are modified.

Exploring the Larger Receptive Field

In this section, we are interested in whether the convergence of ReX is affected by RNN and if ReX still maintains accuracy for a downstream task even when the receptive field is large. To this end, we consider the image classification task

[1] Quad-Core ARM Cortex-A57 MPCore combined with Dual-Core NVIDIA Denver 2 64-Bit CPU.

[2] https://pytorch.org/mobile/home/.

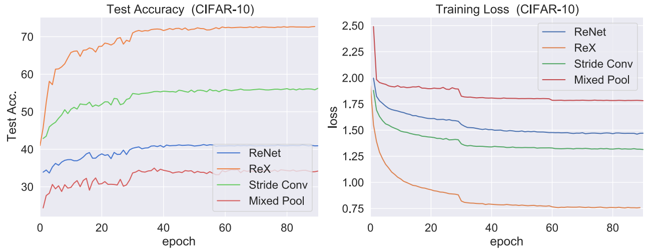


Figure 6: Comparison of ReX and baselines in terms of training loss and test accuracy. ReX converges better than baselines with the same shape of input-output feature maps, and generalizes better than ReNet.

with CIFAR-10 dataset (Krizhevsky and Hinton 2009) but the ReX and baselines are required to down-sample the input of $32 \times 32 \times 3$ to a vector of size $1 \times 1 \times 128$ in *one go*, i.e., both patch size and stride are 32. This is followed by a fully connected (FC) layer projection to 10 from 128. We use $h_1 = 32, h_2 = 16$ for the ReX with patch size and stride as 32. For Mixed pooling models, a 1×1 convolution of 128 filters is used before Max and Average pooling. Here, we follow the prescribed optimization procedure from (Visin et al. 2015), using SGD with an initial learning rate of 0.05. We run training for 90 epochs and compare with baselines. Results are shown in Figure 6. Our method achieves an accuracy of **72.71%**, while the convolution layer, ReNet, and mixed pooling's accuracy are 56.24%, 41.16% and 34.46%, respectively. The ReX outperforms other methods by a substantial amount both in test accuracy and training loss.

Analytical Results

Ablation: ReX. We first consider the changes in accuracy, peak RAM, and FLOPs on different hyperparameters of ReX like patch size, hidden dimensions, and stride. Then we validate the effectiveness of different components. Here we use MobileNetV2 (6 exits) as the base network and the dataset is ImageNet-1k. Note that the last line refers to the ReX-MobileNetV2 (6 exits) in Table 1, and the first block of Table 4 are variations on it. We also remove or alter the components of the ReX architecture in the second block of Table 4. For a clear comparison, here we do not change the classifier in the architecture. One can observe that further fine-tuning hyper-parameter leads to a better trade-off between accuracy and compute requirements. Moreover, adopting a linear projection $\boldsymbol{f}_{\mathrm{L}}$ and a bidirectional RNN $\boldsymbol{R_2}$ are both important techniques to achieve high accuracy.

Impact of Predefined Count k**.** As illustrated in Figures 4 and 7, varying count k can lead to different speed-up ratios and performance. For a MobileNetV2 (6 exits), CBEE reaches peak performance with a count of 3 or 4. On ImageNet and Visual Wake Words, CBEE can always outperform the baseline with a count between 4 and 5. One can observe that the latency drops as the count k goes down (see Figure 4). An interesting phenomenon is that the relationship between accuracy and counts is an inverted-U curve, which may be attributed to the overthinking problem (Kaya, Hong, and Dumitras 2019). That is to say, the later classifier may make a wrong prediction for samples that were classified correctly by an early classifier, while increasing the count k help more samples flow to the later classifier. Thus, it is clear that a proper count leads to considerably better performance. Besides, the optimal k may be different on varying datasets.

Ablation	FLOPs	Peak RAM	Acc.
Stride $s = 6$	0.19G	0.1MB	70.5%
Patch Size $h = w = 4$	0.23G	0.2MB	71.7%
Patch Size $h = w = 12$	0.26G	0.2MB	72.0%
Hidden Sizes $h_1 = 8$, $h_2 = 8$	0.21G	0.1MB	71.1%
Hidden Sizes $h_1 = 16$, $h_2 = 16$	0.28G	0.4MB	73.1%
w/o Global Vectors $\boldsymbol{V}^{\mathrm{G}}$	0.21G	0.1MB	69.7%
w/o Row Summary $\boldsymbol{V}^{\mathrm{R}}$	0.22G	0.1MB	70.9%
w/o Column Summary $\boldsymbol{V}^{\mathrm{C}}$	0.22G	0.1MB	70.8%
Patch Size $h = w = 6$; $s = 4$; Hidden Sizes $h_1 = 16$, $h_2 = 8$	0.24G	0.2MB	72.8%

Table 4: Comparison of accuracy, peak RAM and FLOPs for variations in hidden dimensions, patch size and stride in ReX for MobileNetV2 (6 exits) and on ImageNet-1k dataset.

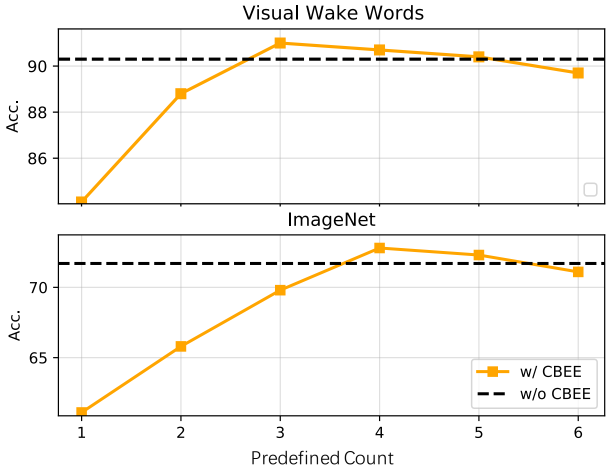


Figure 7: Accuracy scores under different count k with MobileNetV2 (6 exits). The black dotted lines represent the original MobileNetV2 without internal exits.

Conclusion

This paper has proposed a memory-efficient recurrent aggregation operator (ReX), which uses RNNs to effectively aggregate intra-patch features within a large receptive field, while reducing peak RAM and power consumption for adaptive networks. The resulting model, named ReXNet, can be easily extended to stop inference dynamically by introducing a CBEE criteria, which is based on the consistency of classification decisions over several modules. Extensive experiments demonstrate that our method outperforms existing works in terms of both theoretical computational efficiency and actual inference speed.

Acknowledgments

This work was supported in part by the Natural Science Foundation of China under Grant 61825601, Grant U21B2049, Grant U21B2044 and Grant 61906096.

References

Acuna, D.; Ling, H.; Kar, A.; and Fidler, S. 2018. Efficient interactive annotation of segmentation datasets with polygon-rnn++. In *The IEEE conference on Computer Vision and Pattern Recognition (CVPR)*, 859–868.

Bell, S.; Lawrence Zitnick, C.; Bala, K.; and Girshick, R. 2016. Inside-Outside Net: Detecting Objects in Context With Skip Pooling and Recurrent Neural Networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Bolukbasi, T.; Wang, J.; Dekel, O.; and Saligrama, V. 2017. Adaptive Neural Networks for Fast Test-Time Prediction. In *International Conference on Machine Learning (ICML)*.

Cho, K.; Van Merriënboer, B.; Gulcehre, C.; Bahdanau, D.; Bougares, F.; Schwenk, H.; and Bengio, Y. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

Cho, M.; and d, D. 2017. MEC: memory-efficient convolution for deep neural network. In *International Conference on Machine Learning (ICML)*, 815–824.

Chowdhery, A.; Warden, P.; Shlens, J.; Howard, A.; and Rhodes, R. 2019. Visual Wake Words Dataset. *arXiv preprint arXiv:1906.05721*.

Deng, J.; Dong, W.; Socher, R.; Li, L.-J.; Li, K.; and Fei-Fei, L. 2009. Imagenet: A large-scale hierarchical image database. In *The IEEE conference on Computer Vision and Pattern Recognition (CVPR)*.

Figurnov, M.; Collins, M. D.; Zhu, Y.; Zhang, L.; Huang, J.; Vetrov, D.; and Salakhutdinov, R. 2016. Spatially Adaptive Computation Time for Residual Networks. *arXiv preprint arXiv:1612.02297*.

Graves, A. 2016. Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*.

Gural, A.; and Murmann, B. 2019. Memory-optimal direct convolutions for maximizing classification accuracy in embedded applications. In *International Conference on Machine Learning (ICML)*, 2515–2524.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep Residual Learning for Image Recognition. In *The IEEE conference on Computer Vision and Pattern Recognition (CVPR)*.

He, Y.; Zhang, X.; and Sun, J. 2017. Channel Pruning for Accelerating Very Deep Neural Networks. In *The IEEE International Conference on Computer Vision (ICCV)*.

Hinton, G.; Deng, L.; Yu, D.; Dahl, G. E.; Mohamed, A.-r.; Jaitly, N.; Senior, A.; Vanhoucke, V.; Nguyen, P.; Sainath, T. N.; et al. 2012. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal processing magazine*, 29(6): 82–97.

Howard, A.; Sandler, M.; Chu, G.; Chen, L.-C.; Chen, B.; Tan, M.; Wang, W.; Zhu, Y.; Pang, R.; Vasudevan, V.; Le, Q. V.; and Adam, H. 2019. Searching for MobileNetV3. In *The IEEE International Conference on Computer Vision (ICCV)*.

Hu, H.; Dey, D.; Hebert, M.; and Bagnell, J. A. 2019. Learning Anytime Predictions in Neural Networks via Adaptive Loss Balancing. In *Thirty-Third AAAI Conference on Artificial Intelligence*.

Hu, T.; Chen, T.; Wang, H.; and Wang, Z. 2020. Triple Wins: Boosting Accuracy, Robustness and Efficiency Together by Enabling Input-Adaptive Inference. In *International Conference on Learning Representations (ICLR)*.

Huang, G.; Chen, D.; Li, T.; Wu, F.; van der Maaten, L.; and Weinberger, K. Q. 2018. Multi-Scale Dense Networks for Resource Efficient Image Classification. In *International Conference on Learning Representations (ICLR)*.

Huang, G.; Liu, Z.; van der Maaten, L.; and Weinberger, K. Q. 2017. Densely connected convolutional networks. In *The IEEE conference on Computer Vision and Pattern Recognition (CVPR)*.

Jiang, H.; Kim, B.; Guan, M.; and Gupta, M. 2018. To trust or not to trust a classifier. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Jouppi, N. P.; Young, C.; Patil, N.; Patterson, D.; Agrawal, G.; Bajwa, R.; Bates, S.; Bhatia, S.; Boden, N.; Borchers, A.; et al. 2017. In-datacenter performance analysis of a tensor processing unit. In *International Symposium on Computer Architecture (ISCA)*, 1–12.

Kaya, Y.; Hong, S.; and Dumitras, T. 2019. Shallow-Deep Networks: Understanding and Mitigating Network Overthinking. In *International Conference on Machine Learning (ICML)*.

Kobayashi, T. 2019. Global feature guided local pooling. In *The IEEE International Conference on Computer Vision (ICCV)*.

Krizhevsky, A.; and Hinton, G. 2009. Learning multiple layers of features from tiny images. In *Tech Report*.

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NIPS)*.

Lai, L.; Suda, N.; and Chandra, V. 2018. Cmsis-nn: Efficient neural network kernels for arm cortex-m cpus. *arXiv preprint arXiv:1801.06601*.

LeCun, Y.; Bengio, Y.; and Hinton, G. 2015. Deep learning. *nature*, 521(7553): 436–444.

Lee, C.-Y.; Gallagher, P. W.; and Tu, Z. 2016. Generalizing pooling functions in convolutional neural networks: Mixed, gated, and tree. In *Artificial Intelligence and Statistics (AISTATS)*, 464–472.

Li, H.; Zhang, H.; Qi, X.; Yang, R.; and Huang, G. 2019. Improved Techniques for Training Adaptive Deep Networks. In *The IEEE International Conference on Computer Vision (ICCV)*.

Liu, N.; Han, J.; and Yang, M.-H. 2018. PiCANet: Learning Pixel-Wise Contextual Attention for Saliency Detection.

In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Saha, O.; Kusupati, A.; Simhadri, H. V.; Varma, M.; and Jain, P. 2020. RNNPool: Efficient Non-linear Pooling for RAM Constrained Inference. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, 20473–20484.

Sandler, M.; Howard, A.; Zhu, M.; Zhmoginov, A.; and Chen, L.-C. 2018. Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation. In *The IEEE conference on Computer Vision and Pattern Recognition (CVPR)*.

Szegedy, C.; Liu, W.; Jia, Y.; Sermanet, P.; Reed, S.; Anguelov, D.; Erhan, D.; Vanhoucke, V.; and Rabinovich, A. 2015. Going deeper with convolutions. In *The IEEE conference on Computer Vision and Pattern Recognition (CVPR)*.

Szegedy, C.; Zaremba, W.; Sutskever, I.; Bruna, J.; Erhan, D.; Goodfellow, I. J.; and Fergus, R. 2014. Intriguing properties of neural networks. In *International Conference on Learning Representations (ICLR)*.

Teerapittayanon, S.; McDanel, B.; and Kung, H. T. 2016. BranchyNet: Fast inference via early exiting from deep neural networks. In *International Conference on Pattern Recognition (ICPR)*.

Visin, F.; Kastner, K.; Cho, K.; Matteucci, M.; Courville, A.; and Bengio, Y. 2015. Renet: A recurrent neural network based alternative to convolutional networks. *arXiv preprint arXiv:1505.00393*.

Visin, F.; Romero, A.; Cho, K.; Matteucci, M.; Ciccone, M.; Kastner, K.; Bengio, Y.; and Courville, A. C. 2016. ReSeg: A Recurrent Neural Network-Based Model for Semantic Segmentation. *IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 426–433.

Wang, J.; Yang, Y.; Mao, J.; Huang, Z.; Huang, C.; and Xu, W. 2016. CNN-RNN: A unified framework for multi-label image classification. In *The IEEE conference on Computer Vision and Pattern Recognition (CVPR)*, 2285–2294.

Wang, X.; Yu, F.; Dou, Z.-Y.; Darrell, T.; and Gonzalez, J. E. 2018. SkipNet: Learning Dynamic Routing in Convolutional Networks. In *The European Conference on Computer Vision (ECCV)*.

Yang, L.; Han, Y.; Chen, X.; Song, S.; Dai, J.; and Huang, G. 2020. Resolution Adaptive Networks for Efficient Inference. In *The IEEE conference on Computer Vision and Pattern Recognition (CVPR)*.

Zhao, J.; and Snoek, C. 2021. LiftPool: Bidirectional ConvNet Pooling. In *International Conference on Learning Representations (ICLR)*.

Zhou, W.; Xu, C.; Ge, T.; McAuley, J.; Xu, K.; and Wei, F. 2020. BERT Loses Patience: Fast and Robust Inference with Early Exit. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, 18330–18341.