T-RecX: Tiny-Resource Efficient Convolutional neural networks with early-eXit

Nikhil P. Ghanathe
nikhilghanathe@ece.ubc.ca
University of British Columbia
Vancouver, Canada

Steve Wilton
stevew@ece.ubc.ca
University of British Columbia
Vancouver, Canada

ABSTRACT

Deploying Machine learning (ML) on milliwatt-scale edge devices (tinyML) is gaining popularity due to recent breakthroughs in ML and Internet of Things (IoT). Most tinyML research focuses on model compression techniques that trade accuracy (and model capacity) for compact models to fit into the KB-sized tiny-edge devices. In this paper, we show how such models can be enhanced by the addition of an early exit intermediate classifier. If the intermediate classifier exhibits sufficient confidence in its prediction, the network exits early thereby, resulting in considerable savings in time. Although early exit classifiers have been proposed in previous work, these previous proposals focus on large networks, making their techniques suboptimal/impractical for tinyML applications. Our technique is optimized specifically for tiny-CNN sized models. In addition, we present a method to alleviate the effect of network overthinking by leveraging the representations learned by the early exit. We evaluate T-RecX on three CNNs from the MLPerf tiny benchmark suite for image classification, keyword spotting and visual wake word detection tasks. Our results show that T-RecX 1) improves the accuracy of baseline network, 2) achieves 31.58% average reduction in FLOPS in exchange for one percent accuracy across all evaluated models. Furthermore, we show that our methods consistently outperform popular prior works on the tiny-CNNs we evaluate.

CCS CONCEPTS

• Computing methodologies → Neural networks; **• Computer systems organization → Embedded and cyber-physical systems**; **• Human-centered computing → Ubiquitous and mobile computing**.

KEYWORDS

Early-exit networks, TinyML, Adaptive inference, Dynamic neural networks, Edge computing

ACM Reference Format:
Nikhil P. Ghanathe and Steve Wilton. 2023. T-RecX: Tiny-Resource Efficient Convolutional neural networks with early-eXit. In *20th ACM International Conference on Computing Frontiers (CF '23), May 9–11, 2023, Bologna, Italy.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3587135.3592204

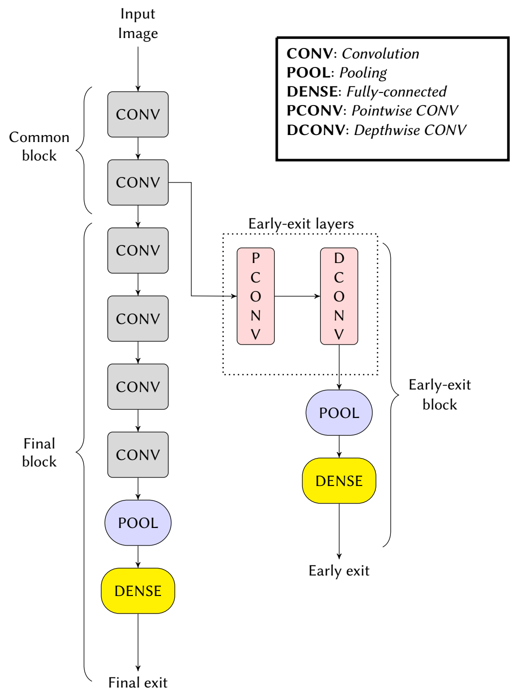


Figure 1: CNN with early-exit

1 INTRODUCTION

Recent advancements in Machine learning (ML) and IoT have produced ML models that have compute and memory requirements low enough to be run directly on small milliwatt-scale devices. These models enable several real-world applications such as smart farming [55], smart cities [5, 35], driverless cars [13, 25, 36], and smart healthcare [43, 52]. The study of deploying such ML models on ultra-low power resource-constrained milliwatt-scale edge devices is called *TinyML* [59].

The capabilities of TinyML are limited by strict power, resource, and compute constraints of the edge devices. TinyML targets ultra-low-power devices (milliwatt range) with limited compute power that are often battery-operated. TinyML models must be small enough (few kilobytes) to fit into the KB-sized tiny-edge devices and require all computations to run locally with no support from the cloud. Thus, one of the major objectives of TinyML is to minimize the *energy per prediction*. To that end, several solutions have been proposed to improve the energy-efficiency/inference time of ML models running on edge devices. The predominant body of prior work involves model compression techniques such as pruning [33], quantization [10, 11, 16] and knowledge distillation [44]. However, compression comes at the cost of model accuracy and leads to reduced model capacity. Other areas of work such as split-computing [24] and prediction cascades [56] have been proposed for the tiny-edge setting (more details in Section 2). These methods also suffer from reduced model capacity and often require delegation to the cloud to conserve network accuracy. As a result, they are impractical for our applications.

In contrast, early-exit networks preserve the network's model capacity while reducing the average inference time. Early-exit networks place intermediate classifiers along the depth of a baseline DNN that are capable of producing an output that approximates the output of the final classifier. These intermediate classifiers act as additional exit paths for the DNN. If the *confidence* score of the output at any intermediate classifier is above a predefined/learned threshold, the DNN exits and the output of that intermediate classifier is forwarded to the output of the DNN. Figure 1 shows a convolution neural network (CNN) with one early-exit block. Early-exit strategies are quite general and can be applied to off-the-shelf state-of-the-art networks as a means to improve the average inference time with minimal overhead [51]. In addition, early-exit can be applied in conjunction with other energy reduction techniques such as pruning and quantization.

However, early-exits are detrimental to the network performance [21], and this negative effect worsens as model size decreases (Section 3). Our key contributions are summarized as follows.

- We identify key challenges in applying early-exits for tiny-CNNs (Section 3), and demonstrate that prior early-exit works are inefficient/ineffective (Section 6.4).
- We propose T-RecX, an early-exit architecture specialized for tiny-CNNs that addresses these challenges (Section 4).
- We develop a method to mitigate the effect of network *overthinking* [26] for tiny-CNNs that reclaims some of the lost accuracy due to early-exit (Section 4.3).
- T-RecX achieves 31.58% average reduction in FLOPS for one percent accuracy trade-off across all the CNNs we evaluate from the MLPerf tiny benchmark suite [2].
- Our methods increases the accuracy of the baseline network in all tiny-CNNs we evaluate.

To the best of our knowledge, we are the first to propose the use of an early-exit strategy for tiny-CNNs that optimizes for both accuracy and floating-point operations per second (FLOPS) reduction.

2 RELATED WORK

Resource efficient machine learning Several works have been proposed for improving the average inference time of CNNs at the edge. The primary methods relevant for the tinyML space involve model compression techniques [10, 16, 17, 33, 44] to fit models into tiny-edge devices. Other approaches propose changes to network topology and explore resource and computationally efficient versions of traditional neural network models [9, 14, 15, 29, 42]. These efforts are complementary to our work, and can be used in combination with our proposed early-exit technique. Two other solutions, which are closer to our approach, are split computing [12, 19, 24] and prediction cascades [1, 4, 46, 56]. Split computing seeks to partition the ML model between the edge device and cloud. Cascade networks consist of a cascade of DNN models that get progressively larger to obtain a prediction cascade. Unlike split computing, the computations are not reused in cascading networks. Since both approaches rely on the cloud, they are impractical for tinyML.

Early-exit networks Teerapittayanon et al. [51] first introduced early-exit strategies for CNNs. That work proposes adding two early-exits and uses multiple convolution blocks as part of its early-exit architecture. Similarly, Szegedy et al. [49] use convolution layers in its early-exit blocks. Another relevant work, Multiscale Densenet (MSDNet) [21] generates multiple feature maps after each layer with different scales. The authors show that adding early-exits interferes with the features learnt at the later layers. Hence, the multi-scale feature maps help in maintaining coarse-level features throughout the network, which helps the accuracy of early-exit classifiers. The early-exit blocks consists of one average pooling layer followed by a dense layer. Bonato et al. [3] focuses on boosting the classification rate of a specific class at early-exit. Its early-exit architecture is identical to that of MSDNet. The overall accuracy of the network is improved since each early-exit block has access to all feature maps from preceding layers. Skipnet [57] selectively skips convolution layers during inference by using a small gating network without sacrificing accuracy. Shallow deep network [26] introduce early-exits to CNNs to study the problem of network overthinking and mitigate its destructive effect. It uses multiple early-exits with pooling and dense layers. Leontiadis et al. [30] describes hierarchical models from a global base model for the mobile edge. The appropriate model is selected at runtime depending on the computation budget.

Several other works [6, 38, 39, 41, 50, 61] present early-exit networks. Some tinyNN-specific early-exit works [31, 32, 60] exit purely based on energy (battery) constraints and sacrifice heavily on accuracy. In our work, we optimize for both accuracy and FLOPS and provide a tunable parameter that can trade-off between both. Almost all prior works include multiple early-exit classifiers for balancing the network accuracy, which is unsustainable for tiny-CNNs. Furthermore, we find that apart from the works in [3] and [21], none explore utilizing high-level representations learned by the earlier layers at the final layers. Unlike T-RecX, both [3] and [21] leverage feature maps from multiple exit points, which results in a prohibitive cost for the tiny-edge environment (Section 4.3).

3 MOTIVATION AND CHALLENGES

Deep neural networks are growing increasingly deeper in the pursuit of higher accuracy. However, the accuracy gains by using deeper networks are paltry after a certain point [56]. For example, a Resnet50 [18] model for an image classification task on the ImageNet dataset [8] achieves an accuracy of 76.2%, but, a Resnet101 model (double the layers) for the same task achieves an accuracy of 77.4%. It is evident that a smaller model will suffice for many applications. Previous works [3, 21, 51, 56] have demonstrated that there are a large number of *easy-to-classify* samples that do not require the full depth of the DNNs. Early-exit networks exploit this property to improve the average inference time of the network. However, as described in Section 2, most of the contemporary works target larger neural networks that are not suitable for deployment on tiny-edge devices. In this section, we enumerate the major challenges in applying the early-exit technique on tiny-CNNs.

Additional exit-path affects network accuracy TinyML models are primarily optimized to reduce their memory and compute requirements through aggressive model compression [11, 33, 44, 45] to fit on low-end devices while compromising on accuracy to some degree. Consequently, the compressed models are stripped to their bare-bones and have a high degree of *neuron co-adaptations* between successive layers compared to their larger counterparts. Any change to the model architecture, like adding an early-exit is adversarial and detriments the performance of the network. Additionally, given the size of tiny-CNNs, the early-exit block has to be attached as close to the input as possible to see any real performance gains, which irreversibly disrupts the crucial initial layers of the CNN [62]. Figure 2 illustrates this effect for a 4-layer depthwise separable CNN model (DS-CNN). Figure 2 shows the standalone accuracy of the final exit of the DS-CNN [63] model when a single early-exit block is attached at $\frac{1}{4}^{\text{th}}$, half and $\frac{3}{4}^{\text{th}}$ of the network. The standalone accuracy of final exit refers to the accuracy of the final classifier when all inputs take the final exit (no early-exit). The standalone accuracy of the early-exit refers to the accuracy when all samples take the early-exit. From the figure, we observe that the configuration where the early-exit is attached at $\frac{1}{4}^{\text{th}}$ of the network i.e. closest to the input results in the highest harm to the standalone accuracy of the final classifier for the reasons discussed above.

Existing early-exit strategies are ineffective for tiny-CNNs The predominant body of existing work on early-exits target large neural networks, which also suffer from accuracy degradation with even a single early-exit classifier [21]. To combat this challenge, a majority of the existing works attach multiple early-exit blocks along the length of the CNN [3, 26, 47, 51] to avoid the final-exit as much as possible. For example, early-exit networks from Kaya et. al. [26] place their early-exit classifiers at 15%, 30%, 45%, 60%, 75% and 90% of the Resnet-56 network for image classification on the CIFAR-10 dataset [28]. For the same task, Teerapittayanon et. al. [51] adds two early-exit branches after the 2nd convolution layer and 37th convolution layer of the Resnet-110 network. In contrast, the tiny-CNN model we evaluate for the same task and dataset is a tiny Resnet-8 model [2]. Furthermore, the large CNN networks targeted in prior works are less vulnerable to an extra output compared to tiny-CNNs because the larger networks have enough layers after the early-exit block to relearn the complex neuron relations disturbed by the early-exit. Moreover, the relative placement of the early-exit in larger networks usually bypasses the initial crucial layers. Another approach adopted by prior works [3, 21] is to combine feature maps from all previous early-exits. However, the cost incurred by both placing multiple early-exits and combining feature maps from previous layers is untenable for tiny-CNNs. For example, the Resnet-8 model with two early-exits results in 17.2% increase in model parameters, which is huge from a tinyML perspective. As a result, tiny-CNNs are limited to a single early exit placed as close to the input as possible because the inference cost (FLOPS) quickly grows beyond that of the unmodified baseline network as we move away from the input. Unfortunately, the proximity of the early-exit to the input in CNNs results in notable accuracy degradation. Thus, early-exit in tiny-CNNs induce a trade-off between network accuracy and FLOPS. The primary contribution of this work is to study the feasibility and effects of early-exit on tiny-CNNs and formulate techniques that balances the overall network accuracy while delivering notable reduction in FLOPS.

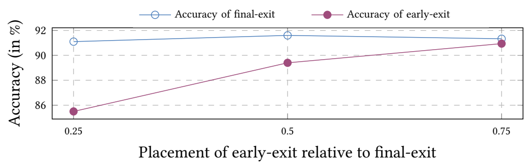


Figure 2: The accuracy of final classifier with respect to the relative location of early-exit.

4 T-RECX

T-RecX modifies the baseline tiny-CNN model by adding a single early-exit block as shown in Figure 1. However, the reduced model capacity of the early-exit classifier induces a high scope for misclassification. T-RecX addresses this challenge using a novel early-exit block for tiny-CNNs and employing a joint-training method. We formulate the problem formally in Section 4.1. Section 4.2 describes the architecture of the early-exit block and a training method with a joint loss function. Section 4.3 describes a method to mitigate the effect of network *overthinking* at the final classifier by utilizing information from the early-exit block, thereby enhancing the overall network accuracy.

4.1 Problem Setup

Given a baseline trained tiny-CNN model $M_B(x, \theta)$, T-RecX adds an early-exit block E_e as shown in Figure 1. θ represents the model parameters learned after training. The model after adding the early-exit block is denoted by $M_{B_e}(x, \theta_e)$ and is defined as,

$$M_{\mathrm{B_e}}(x, \theta_e) = \begin{cases} argmax(z_{i:\text{E-e}}), & \text{if } max(z_{i:\text{E-e}}) \geq \rho \\ argmax(z_{i:\text{E-f}}), & \text{otherwise} \end{cases} \tag{1}$$

where $z_{i:\text{E-e}}$ and $z_{i:\text{E-f}}$ are the softmax scores of the early-exit classifier and the final classifier respectively computed from the i^{th} input sample x_i. ρ is the *confidence threshold* such that $0 \leq \rho \leq 1$, which can be used to tune the early-exit rate (EE_{rate}) to tradeoff between classification accuracy and network inference cost. The softmax score at early-exit is used as the confidence metric. If the highest score of the softmax output i.e. the score of the predicted label at the early-exit is greater than or equal to a threshold (ρ), the CNN exits at the early-exit and the prediction of the early-exit classifier is forwarded to the output of the CNN. Otherwise, the prediction of the final classifier is forwarded to the output.

The total accuracy of the model is given by,

$$\text{acc}_{\text{total}} = \frac{\sum_{\forall i \in \mathcal{N}_e}(y_i == \hat{y}_i) + \sum_{\forall i \in \mathcal{N}_f}(y_i == \hat{y}_i)}{\eta_e + \eta_f} \tag{2}$$

where $\mathcal{N}_e$ and $\mathcal{N}_f$ are the set of input samples that exit at the early-exit (E_e) and the final exit (E_f) and, η_e and η_f are the number of test samples present in $\mathcal{N}_e$ and $\mathcal{N}_f$ respectively. The total number of test samples (η) is given by $\eta = \eta_e + \eta_f$. y_i and $\hat{y}_i$ denote the predicted label and the true label respectively.

At a high level, the objective of T-RecX is to maximize both the early-exit rate given by $EE_{rate}(\rho) = \frac{\eta_e}{\eta_e + \eta_f}$, and overall accuracy of the model given by Eq 2. The $EE_{rate}(\rho)$ is increased by lowering the threshold ρ. However, the accuracy at early-exit degrades quickly as the EE_{rate} increases beyond a certain point because of the limited model capacity of the early-exit classifier. On the other hand, as the EE_{rate} is reduced, the network inference cost (C) increases. We describe the network inference cost in terms of FLOPS. As shown in Figure 1, the early-exit model $M_{B_e}(\theta_e)$ can be divided into three parts i.e. common block, early-exit block and the final block. The network inference cost is given by,

$$C = \begin{cases} C_{\text{E-e}}, & \text{if early-exit} \\ C_{\text{E-f}}, & \text{otherwise} \end{cases} \tag{3}$$

where, $C_{\text{E-e}} = C_{\text{common}} + C_{\text{EE}}$ and $C_{\text{E-f}} = C_{\text{E-e}} + C_{\text{final}}$

C_{common}, C_{EE} and C_{final} are the inference cost of the common block, early-exit block and the final block of the network respectively. $C_{\text{E-e}}$ and $C_{\text{E-f}}$ give the inference cost of the early-exit-classifier and the final classifier. The objective of T-RecX distills to minimizing the total network inference cost (C) with an acceptable trade-off on classification accuracy.

4.2 Architecture of Early-exit block and Training Methodology

Since the feature map input to the early-exit block is a high-dimensional variable, it is not highly informative of its true label [53]. As a result, most prior works only have a feature reduction stage (pooling) before the classification (dense) layer. Our preliminary experiments showed that for tiny-CNNs, naïve feature reduction leads to massive information loss and accuracy degradation. Further investigations revealed that high-dimensional feature maps do not posses sparse class-specific information that help in lowering network confusion (ambiguity in prediction). Our analysis showed that the early-exit classifier requires additional information to resolve *confusion*. To accomplish this, we increase the number of channels by introducing a combination of *pointwise* and *depthwise* convolutions layers. Figure 1 shows the architecture of T-RecX's early-exit block. The early-exit block consists of a pointwise convolution that increases the number of channels followed by a depthwise convolution that extracts feature information from each channel. By increasing the number of channels, we increase the *perceptions* of the input image thereby lowering the *confusion* of the early-exit classifier. Simultaneously, we increase the stride of the depthwise convolution layer to achieve partial feature reduction before the pooling and dense (classification) layers. Our analyses showed that the accuracy of the early-exit steadily increases as its channel width expands, as long as the channel width of the early-exit is below or equal to that of the final exit. The accuracy gain saturates quickly beyond this point. We find that the optimal configuration in terms of both accuracy and resource usage is when the PCONV layer of the early-exit doubles the number of channels at its input. A higher channel width incurs significant memory and compute overhead whereas, a lower channel width results in meager accuracy gains. In our evaluations, we configure the PCONV layer for the same.

Training Methodology Given an untrained baseline model M_{B_e} attached with an early-exit block as shown in Figure 1, T-RecX trains the network parameters θ_e, such that the network inference cost C is minimized with an acceptable trade-off on accuracy. T-RecX uses the cross-entropy loss with the Adam optimizer [27] as the objective function, which is the same as that used by the reference models [2]. The early-exit block is added to the baseline network and all the weights are learned simultaneously. Since the early-exit CNN is a dual-output neural network, the combined loss value is obtained by adding the loss terms from each output. The proximity of the early-exit to the input layer results in the weights of the initial layers (common block) being heavily optimized for the early-exit leading to accuracy drop at the final classifier (Section 3). Therefore, to attenuate the effect of the early-exit on the common block layers, T-RecX weights the loss term at the early-exit by a factor $w_{\text{E-e}}$ such that $0 < w_{\text{E-e}} < 1$. Thus, the total loss is given by, $\mathcal{L} = w_{\text{E-e}} \cdot \mathcal{L}_{\text{E-e}} + \mathcal{L}_{\text{E-f}}$. The loss terms computed at the early and final exit are $\mathcal{L}_{\text{E-e}}$ and $\mathcal{L}_{\text{E-f}}$ respectively. A smaller weighting of the loss function at early-exit reduces the impact on the co-adaptations of neurons in the common block of the network. We experiment with various values of $w_{\text{E-e}}$ for each network we evaluate. The $w_{\text{E-e}}$ value that gives the maximum standalone accuracy at the final classifier i.e. the value of $w_{\text{E-e}}$ that causes minimum disturbance in the complex relations learned between the layers of the common block and the final block is selected. The values that satisfy this constraint are reported in Section 5.

4.3 Early-view assisted classification

Despite the effectiveness of the proposed early-exit block and the training method, there are still some samples that are misclassified at the early-exit but would have been correctly classified at the final exit. Similarly, there exists a set of samples that would have been correctly classified at the early-exit had they exited there. The final classifier misclassifies these samples due to *overthinking*. From Kaya et al. [26], a network is said to *overthink* when 1) the higher-level representations of an input sample computed by an early layer suffices for a correct classification and 2) further computation by the subsequent layers lead to misclassification by the final classifier. Hence, the biggest challenge with adding a single early-exit is to determine which input samples should exit early, and which should not. At best, this can be predicted by a heuristic, inevitably leading to accuracy loss. This is an open problem and almost all solutions to tackle it involve multiple early-exits. Although, the problem of overthinking in tiny-CNNs is not as dominant as in large CNNs, we observe that the problem still persists. Early-exit networks provide an opportunity to mitigate the effect of overthinking and reclaim some of the lost accuracy. Since the early-exit block is computed before the final classifier, it presents us with an opportunity to leverage information from the early-exit block to improve the performance of the final classifier. However, naïve concatenation of the output of the early-exit block with the final layer yields poor results. T-RecX recognizes that the filter weights of the convolution layers in the early-exit block capture the essence of the features that are learnt at the early-exit.

A convolution filter extracts certain features from the input depending on the values of the filter weights. For example, a Sobel filter [23] detects edges whereas a Laplacian filter [48] detects blobs. Similarly, the filter weights learned by each convolution layer in a CNN extracts relevant features in a high-dimensional space. The filters weights learned by the convolution layers in the earlier layers are adept at extracting higher-level features like shapes, appearance and outlines. On the other hand, the convolution layers closer to the final classifier focus more on complex features. Occasionally, the eminence of higher-level features learnt by the earlier layers diminishes at the final classifier leading to overthinking. T-RecX addresses this problem by enhancing the higher-level features at the final exit. Figure 3 illustrates the setup for early-view assisted classification. T-RecX assists the final classifier in making its prediction by providing an *early-view* of the input image as follows.

1) T-RecX makes an identical copy of the *depthwise convolution* layer ($DCONV_{\text{E-e}}$) from the early-exit block. The newly created depthwise convolution layer is denoted as $DCONV_{\text{E-f}}$.

2) The input to $DCONV_{\text{E-f}}$ are the feature maps generated by the final convolution block of the CNN.

3) The output of $DCONV_{\text{E-f}}$ is concatenated with the output of the final convolution block, which is subsequently fed into the final classifier (dense layer) for a prediction.

During training, the filter weights learned by $DCONV_{\text{E-e}}$ are copied over to $DCONV_{\text{E-f}}$ at the end of every training batch for approximately the first 80% of the training epochs. For the final 20% of the epochs, the filter weights of $DCONV_{\text{E-f}}$ are free to tune itself to learn the co-adaptations required for assisting the final classifier.

Thus, the $DCONV_{\text{E-f}}$ layer convolutes the output of the final convolution block using the filter weights learned at the early-exit. Since the filter weights learned by the earlier convolution layers provide a different (high-level) *view* of the input image, concatenating this view with that of the final convolution layer allays some confusion in the final classifier caused due to overthinking. The early-view (EV) feature maps thus obtained act as a *regularizer* for the final dense layer and mitigates the negative effect of overthinking. Therefore, the accuracy of the final classifier improves thereby, increasing the overall accuracy of the early-exit network. However, our preliminary evaluations revealed that excessive channel width of $DCONV_{\text{E-f}}$ results in accuracy degradation because the early-view overpowers the final-view. Thus, the channel width must be chosen such that the early-view aids the final-view instead of suppressing it. Our empirical evaluations indicate that the maximum performance gains are obtained when 1) the channel width of $DCONV_{\text{E-f}}$ is less than half the channel width of the final CONV block, and 2) the channel width of $DCONV_{\text{E-f}}$ is at least half the channel width of the early-exit block. To optimize for memory, in our evaluations, we set the channel width of $DCONV_{\text{E-f}}$ to be half the channel width of the early-exit. Section 6.3 discusses the effectiveness of this approach.

5 EVALUATION METHODOLOGY

We evaluate T-RecX on three tasks from the MLPerf tiny benchmark suite [2] published by *MLCommons*: image classification, keyword spotting and visual wake words. These models are tailor-made for tinyML. Further, we evaluate with the exact training scripts (and hyperparamters) used by MLPerf tiny on their official github page [37]. The frameworks used are Tensorflow v2.3 with Python v3.7 and models are trained on a GeForce RTX 2080 GPU.

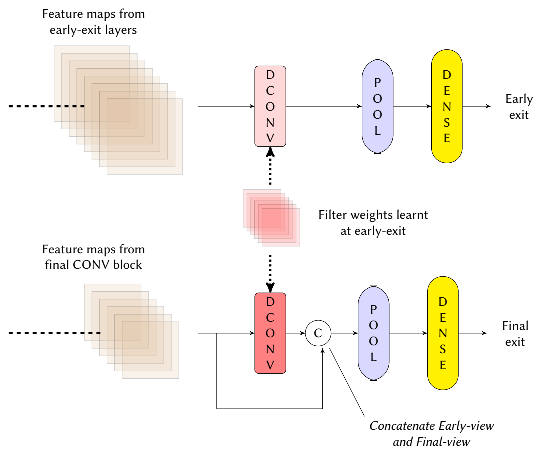

Feature maps from early-exit layers
D C O N V
P O O L
D E N S E
Early exit
Filter weights learnt at early-exit
Feature maps from final CONV block
D C O N V
C
P O O L
D E N S E
Final exit
Concatenate Early-view and Final-view

Figure 3: Early-view assisted classification

Image classification The image classification task uses a customized Resnet [2] that classifies images into ten categories (class) of the CIFAR-10 dataset [28]. The model is trained for 500 epochs with a mini-batch size of 32. The Resnet model has 3 residual stacks. We place the early exit immediately after the 1st residual stack.

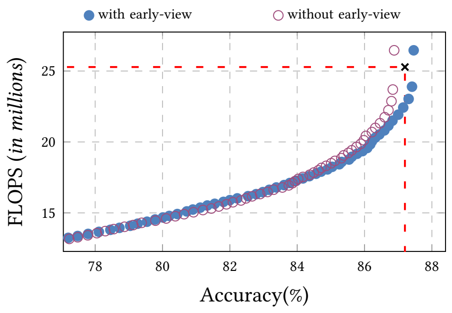
Keyword spotting Keyword spotting recognizes a short phrase uttered by the user. The model used for the task is a depthwise-separable CNN (DS-CNN) [63] with 4 layers of depthwise separable convolutions. It distinguishes between 12 different classes on the Speech command dataset [58]: 10 words + *silence* + *unknown* class. The model is trained for 36 epochs with a mini-batch size of 100. The early-exit is placed immediately after the 2nd depthwise separable layer (i.e. at $\frac{1}{2}^{\text{th}}$ of the network).

Visual wake words The visual wake words is a binary classification problem that detects the presence of a person if the person occupies more than 2.5% of the input image. The model used is MobilenetV1 [20], which outputs two classes: person and no person. The model is evaluated on the visual wake words dataset [7] derived from the MSCOCO 2014 dataset [34]. For evaluation, we use the COCO minival dataset [22] [40]. The Mobilenet model is trained for 50 epochs with a mini-batch size of 32. The mobilenet model has 14 layers; the early-exit is placed after the 4th layer.

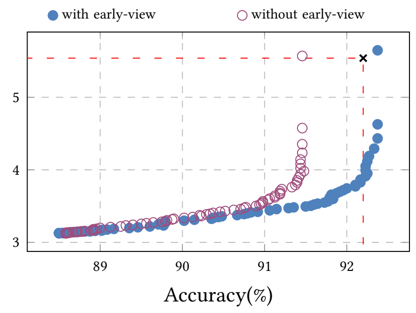
The value of $w_{\text{E-e}}$ used in our experiments that gives the maximum standalone accuracy at the final exit are: Resnet:0.5, DS-CNN:0.3 and Mobilenet:0.4 (see Section 4.2).

6 RESULTS

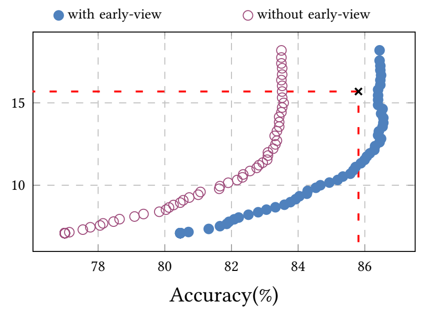
T-RecX achieves an average of **31.58%** reduction in FLOPS (inference time) in exchange for one percent accuracy across all evaluated models (Section 6.1) with ~ 9% increase in model size on average. The FLOPS are obtained using [54]. Further, we show that the proposed techniques enhance the classification accuracy of the baseline network in all models that we evaluate. Section 6.2 compares the performance of T-RecX's early-exit block with the baseline. Next, in Section 6.3 we compare the effectiveness of early-view assisted classification and discuss its role in increasing the accuracy of the



(a) Resnet



(b) DS-CNN



(c) Mobilenet

Figure 4: Benefit curve for all evaluated models

Model	Baseline			T-RecX					
	Accuracy	FLOPS (milions)	Params	FLOPS (millions) at 1% tradeoff	2% tradeoff	3% tradeoff	Params	$Acc_{standalone}$ E-e	E-f
Resnet (IC)	87.2%	25.28	78.7K	20.04	18.43	17.59	89.7K	73.3%	**87.4%**
DS-CNN (KWS)	92.2%	5.54	24.9K	3.45	3.31	3.18	28.2K	88.5%	**92.37%**
Mobilenetv1 (VWW)	85.81%	15.69	221.7K	9.99	8.97	8.36	226.2K	80.46%	**86.44%**

Table 1: Comparison of T-RecX with baseline network

final classifier. Finally, Section 6.4 shows that T-RecX consistently outperforms prior methods. The proposed early-exit strategy is orthogonal to other energy reduction techniques such as pruning and quantization, and can be used in conjunction with them. For example, applying post-training INT8 quantization on the proposed early-exit equipped Mobilenet model results in 70.4% reduction in model size.

6.1 Accuracy vs Inference cost (FLOPS)

Figure 4 plots the FLOPS consumed as a function of the total accuracy (Equation 2), averaged across all test samples. We refer to this plot as the *benefit curve*. All plots are obtained by varying ρ from 0 to 1 in step sizes of 0.01 (see Eq 1). As ρ decreases, EE_{rate} increases leading to a steady FLOPS reduction. However, the accuracy starts to drop quickly with an increase in EE_{rate}. This effect is illustrated in the benefit curves. Figure 4 plots the benefit curve for the evaluated networks with and without early-view assisted classification. The accuracy and the FLOPS of the baseline model is marked by '×' in each plot. For higher values of ρ, the accuracy of the network is maintained (better even in all cases) close to the baseline accuracy. Table 1 reports the improvement in the average FLOPS as ρ reduces. We report the accuracy and the FLOPS at three trade-off points. The trade-off points corresponds to the reduction in FLOPS obtained for every percent of accuracy sacrificed. For example, for the Resnet model with a baseline accuracy of 87.2%, we report the FLOPS at the points closest to 86.2%, 85.2% and 84.2%.

Image Classification (IC) For the image classification (IC) task, the Resnet model achieves a 20.7% reduction in FLOPs on average at the first trade-off point, and an additional FLOPS reduction of 6.39% and 9.71% respectively at the next two trade-off points. As seen in Figure 4 and Table 1, the maximum benefit is gained in exchange for the first percent of accuracy. The improvement in inference time far outweighs the accuracy reduction in this region. In addition to this, T-RecX attains a higher accuracy than that of the baseline. It delivers around 8.9% reduction in the average FLOPS even before the accuracy starts to fall below that of the baseline. The ideal benefit curve has a steep slope. The longer the curve maintains a steep slope, the higher is the reduction in FLOPS without compromising accuracy. The slope of the benefit curve for the Resnet model is affected by two factors. First, there is some disparity in the learning capacities of the early-exit and the final exit. The standalone accuracy of the early-exit i.e. when all inputs exit at the early-exit is 73.39%. On the other hand, the standalone accuracy of the final exit (i.e. no early-exit) is 87.46%. As a result, as ρ decreases more input samples exit at the early-exit and the overall accuracy is impacted by higher misclassifications at the early-exit. Second, the image classification task on the CIFAR-10 dataset is a 10-class classification problem. We observe that as the number of classes increases the *confusion* (misclassification rate) of the network rises. We investigate this phenomenon with the Resnet model. In our experiments, we add an extra class to the Resnet model with no changes to either the training data or model hyperparameters. We observe that the standalone accuracies of the early and final exit drop by 0.54% and 0.75% respectively. This observation is consistent with the other models we evaluate. The proposed techniques mitigate the confusion at the early-exit to an extent by instilling higher confidence in the predictions of the early-exit classifier. Section 6.2 presents more details.

Keyword spotting (KWS) The benefit curve of DS-CNN model [63] for keyword spotting sees the highest reduction in FLOPS (37.7% reduction) in the one-percent trade-off region. At the next two trade-off points, the FLOPS consumed by the network falls down to 40.2% and 42.5%. Furthermore, the standalone accuracies of the early-exit and final exit are 88.5% and 92.37% respectively. Like

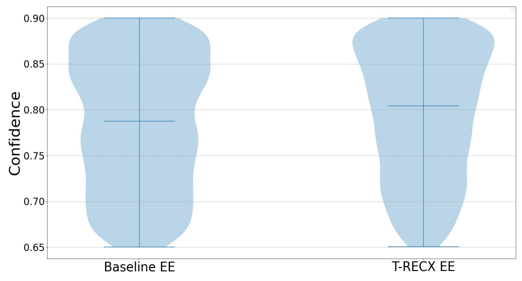


Figure 5: Violin Plot of medium confidence predictions ($0.65 \leq max(softmaxscores) < 0.9$)

Resnet, the standalone accuracy of the final classifier exceeds that of the baseline network, and the total accuracy remains very close to the baseline accuracy for the most part. The model delivers the same network accuracy as that of the baseline network while consuming just 69.6% of the FLOPS compared to the baseline.

The lower difference in the standalone accuracies of the two exits in DS-CNN compared to Resnet is because the early and the final exits are separated by just two depthwise separable convolutions. Thus, the proximity in the learning capacities of the two exits ensures that as the EE_{rate} increases the early-exit classifier reliably classifies the input samples in an equivalent manner to that of the final classifier in the 1% tradeoff region thereby, leading to a steep benefit curve. Beyond this region, as the EE_{rate} increases, the disparity in the model capacities of the early and final exit becomes more prominent leading to steady accuracy degradation.

Visual wake word detection (VWW) The Mobilenet model for visual wake word detection possesses the best benefit curve compared to the other models we evaluate. For the most part, the curve lies beyond the baseline network's accuracy while consistently delivering reduction in FLOPS. Also, the slope remains steep for the majority of the curve. In fact, it delivers a peak reduction of 29.2% in FLOPS without falling below the baseline's accuracy. Our investigations revealed that since the early and the final exit are separated by 10 depthwise separable convolution layers, the problem of network overthinking is more prominent in Mobilenet. Therefore, the early-view (EV) assisted classification technique delivers higher returns. This demonstrates the effectiveness of EV-assitance in mitigating overthinking, potentially for even large CNNs. Further, the reduction in the average FLOPS consumed is 36.3% at the first trade-off point and 42.8% and 46.7% at the next two trade-off points respectively. The standalone accuracies of the early and final exit for Mobilenet are 80.46% and 86.44% respectively. In addition to this, there is a more gradual reduction of FLOPS as ρ decreases compared to the abrupt fall we observe in DS-CNN. Our analysis showed that this behavior is because visual wake word detection is a binary classification task. The classification criteria for a binary classification is when either of the class scores is ≥ 0.5. Therefore, there is less scope for confusion. As a result, although the confidence of the early-exit classifier lowers progressively as ρ reduces, its effect on the accuracy only becomes evident much later. Hence, despite a 10-layer separation between the two exits, the standalone accuracies are closer than expected. Thus, the slope of the benefit curve remains steep for large parts.

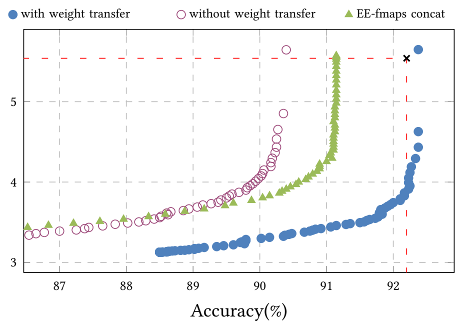


Figure 6: Benefit curve for DS-CNN 1) w/ weight transfer 2) w/o weight transfer from early to final exit, and 3) concatenation of early-exit and final exit feature maps at final exit

6.2 Effectiveness of T-RecX's early-exit block

As discussed in Section 2, in most of the prior works, the early-exit block has either 1) a simple feature reduction stage (pooling) [3, 21, 26] or 2) more learning layers such as convolutions and dense layers followed by feature reduction [49, 51]. In both cases, a final dense layer is included for classification. Since most prior works attach multiple early-exits, they opt for a minimal design with only pooling like the former case. We label this version of the early-exit block as *baseline EE*. However, we find that this design is untenable for tiny-CNNs with single early-exit. We demonstrate the effectiveness of T-RecX's early-exit block with respect to baseline EE. Figure 5 shows the confidence distribution of all predictions obtained at the early-exit for the Resnet model for two configurations: 1) with T-RecX's early-exit block and 2) with baseline EE block. The y-axis plots the maximum score for each prediction. We consider predictions that lie in the medium confidence region ($0.65 \leq max(softmaxscores) < 0.9$). We present the medium confidence region because we find that there is no discernible change in the network's accuracy for scores > 0.9. Similarly, we do not present the low confidence region ($max(softmaxscores) < 0.65$) because the accuracy degradation in this region makes it impractical. As shown in Figure 5, the violin plot of T-RecX's early-exit block is noticeably leaner below the median line compared to that of the baseline. Also, the median line for T-RecX's early-exit block is higher than that of the baseline EE. Evidently, the confidence of the predictions with T-RecX's early-exit block is notably higher than that of baseline EE. This results in a higher EE_{rate} with lower misclassifications at the early-exit. Nevertheless, to put it in context, we present the results obtained on Resnet-8, which produces the least performance improvement out of all models we evaluate.

6.3 Effectiveness of Early-view assistance

Figure 4 also plots the benefit curve for all the models we evaluate without early-view (EV) assisted classification. As seen in the figure, the EV-assisted classification outperforms the non-EV versions for all evaluated models. This underlines the effectiveness of the proposed technique in addressing the problem of network overthinking. For all models, we observe that the EV-assistance at the final exit results in significant improvement in the standalone accuracy of the final exit compared to non-EV-assisted versions i.e.

Model	T-RecX			Branchynet			SDN		
	Par-ams	Accuracy(%)		Par-ams	Accuracy(%)		Par-ams	Accuracy(%)	
		E-e	E-f		E-e	E-f		E-e	E-f
Resnet	89.7K	73.4	**87.4**	98.4K	72.4, 79.7	85.7	94K	63.3, 75	86.3
DSCNN	28.2K	88.8	**92.3**	62.6K	85, 87.6	90.9	38.7K	63.7, 81.7, 89	91.8

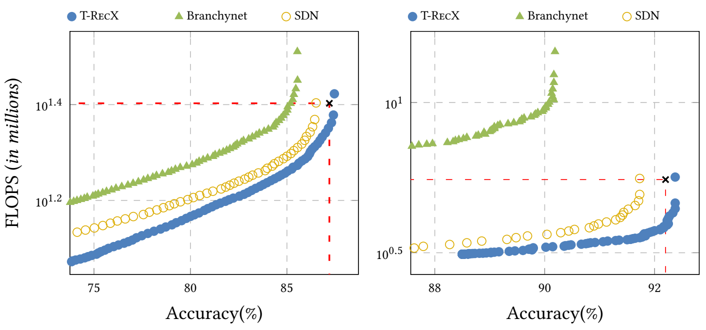
Table 2: Comparison of T-RecX with Branchynet and SDN. (Accuracy numbers reported here are standalone accuracies)

+0.6% for Resnet, +0.9% for DS-CNN and +2.9% for Mobilenet. This in turn ensures higher FLOPS reduction before the curve hits the trade-off points. Furthermore, to demonstrate the significance of knowledge transfer from the early-exit to the final exit, we evaluate the DS-CNN model with and without weight transfer using the architecture shown in Figure 3. In addition, like Huang et al. [21] and Bonato et al. [3], we also evaluate a configuration that concatenates the feature maps of the early-exit with that of the final exit (denoted by *EE-fmaps concat* in Figure 6). The benefit curves for all three configurations are plotted in Figure 6. As seen in Figure 6, the model *with weight transfer* comfortably outperforms the version without weight transfer with an average accuracy improvement of +1.98%. This clearly illustrates that the accuracy improvement is due to EV-assistance, and not simply because of additional learning layers. Interestingly, the model with concatenation of early and final exit feature maps also outperforms the version without weight transfer, which indicates the presence of *overthinking* and the effectiveness of leveraging the knowledge of early-exit to alleviate its effect on accuracy. However, as seen from Figure 6, it is evident that EV-assistance (weight transfer) method is more effectual compared to naïve concatenation of feature maps.

6.4 Comparison with prior works

Figure 7 compares T-RecX with two popular prior works: Shallow Deep Networks (SDN) [26] and Branchynet [51]. As discussed in Sections 2 and 3, both Branchynet and SDN focus on large ML models and employ multiple early-exits. The former places additional convolution layers at its two early-exits before pooling+dense, and the latter opts for higher number of early-exits with a minimalist design of pooling+dense at all exits. We apply the early-exit techniques presented in Branchynet and SDN on two tinyCNNs we evaluate: Resnet-8 and DSCNN. Due to space constraints we omit the results for MobilenetV1 because like DS-CNN, it is also a depthwise-separable model. We denote the branchynet versions of Resnet and DSCNN as Branchy-Resnet and Branchy-DSCNN, and their SDN versions as SDN-Resnet and SDN-DSCNN respectively. Branchy-Resnet and Branchy-DSCNN employ two early-exits at $\frac{1}{3}^{\text{rd}}$ and $\frac{2}{3}^{\text{rd}}$ of its networks. The early-exit blocks of Branchy-Resnet and Branchy-DSCNN have an additional convolution layer before the pooling+dense layers. On the other hand, SDN-Resnet uses two early-exits (after each residual stack) and SDN-DSCNN uses three early-exits (after each depthwise-separable layer) respectively. Both SDN-Resnet and SDN-DSCNN employ a simple pooling+dense configuration at each of their early exits.

Figure 7 compares all benefit curves. T-RecX consistently outperforms both Branchynet and SDN. The model parameters of T-RecX are on average 31.8% *smaller* than Branchynet and 15.8% *smaller* than SDN. Further, Branchynet consumes significantly higher FLOPS compared to other methods because of an additional CONV layer at its early-exits. This is especially noticeable



(a) Resnet **(b) DS-CNN**

Figure 7: Benefit curves of T-RecX, Branchynet [51] and Shallow Deep Networks (SDN) [26] (y-axis in log-scale)

in DSCNN because the CONV layers are more compute/memory intensive compared to depthwise separable convolutions. Also, we observe that the benefit curves of both Branchynet networks are similar to the non-EV (no weight transfer) versions of Figure 4. This result shows that simply adding additional layers at early-exits is not sufficient for accuracy preservation in tiny-CNNs, which in turn exhibits the value of EV-assistance. Furthermore, SDN networks outperform the Branchynet versions because they have a higher number of early-exits. However, the benefit curves of T-RecX is still comfortably superior compared to SDN. Table 2 reports the model sizes and the standalone accuracies of all early-exits and that of the final exit for all evaluated models. In particular, T-RecX achieves the highest standalone accuracy at the final exit due to EV-assistance, which ensures steady FLOPS reduction with minimal compromise on accuracy.

7 CONCLUSIONS

Existing works on early-exit networks either target large networks making them inapplicable for tiny-CNNs or optimize solely for energy while compromising heavily on accuracy. Further, tiny-CNNs have higher sensitivity to early-exits compared to large CNNs. This paper addresses these issues by 1) presenting a novel early-exit architecture for tiny-CNNs, 2) describing a joint training method that mitigates the disruption of neurons relations. Additionally, we show a technique to mitigate the destructive effect of network overthinking at the final exit by distilling knowledge from the early-exit block. T-RecX achieves an average of 31.58% reduction in FLOPS by trading one percent of accuracy across all evaluated models and consistently outperforms prior works. Also, the proposed techniques increase the accuracy of the baseline network in all models we evaluate. The methods presented in this work can easily be extended to large CNNs as well. T-RecX provides a simple way to trade-off between accuracy and inference time (FLOPS) by tuning a single parameter either pre/post deployment based on compute/energy budget, which is useful for the tinyML space to manage battery life.

ACKNOWLEDGEMENTS

NSERC COHESA support gratefully acknowledged.

REFERENCES

[1] Anelia Angelova, Alex Krizhevsky, Vincent Vanhoucke, Abhijit Ogale, and Dave Ferguson. 2015. Real-Time Pedestrian Detection With Deep Network Cascades. In *Proceedings of BMVC 2015*.

[2] Colby R. Banbury, Vijay Janapa Reddi, Peter Torelli, Jeremy Holleman, Nat Jeffries, Csaba Király, Pietro Montino, David Kanter, Sebastian Ahmed, Danilo Pau, Urmish Thakker, Antonio Torrini, Pete Warden, Jay Cordaro, Giuseppe Di Guglielmo, Javier M. Duarte, Stephen Gibellini, Videet Parekh, Honson Tran, Nhan Tran, Wenxu Niu, and Xuesong Xu. 2021. MLPerf Tiny Benchmark. *CoRR* abs/2106.07597 (2021). arXiv:2106.07597 https://arxiv.org/abs/2106.07597

[3] Vanderlei Bonato and Christos-Savvas Bouganis. 2021. Class-specific early exit design methodology for convolutional neural networks. *Applied Soft Computing* 107 (2021), 107316. https://doi.org/10.1016/j.asoc.2021.107316

[4] Zhaowei Cai, Mohammad Saberian, and Nuno Vasconcelos. 2015. Learning Complexity-Aware Cascades for Deep Pedestrian Detection. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*.

[5] Pablo Samuel Castro, Daqing Zhang, and Shijian Li. 2012. Urban traffic modelling and prediction using large scale taxi GPS traces. In *International Conference on Pervasive Computing*. Springer, 57–72.

[6] Zhourong Chen, Yang Li, Samy Bengio, and Si Si. 2019. You Look Twice: GaterNet for Dynamic Filter Selection in CNNs. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 9164–9172. https://doi.org/10.1109/CVPR.2019.00939

[7] Aakanksha Chowdhery, Pete Warden, Jonathon Shlens, Andrew Howard, and Rocky Rhodes. 2019. Visual Wake Words Dataset. *CoRR* abs/1906.05721 (2019). arXiv:1906.05721 http://arxiv.org/abs/1906.05721

[8] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 248–255.

[9] Don Dennis, Chirag Pabbaraju, Harsha Vardhan Simhadri, and Prateek Jain. 2018. Multiple Instance Learning for Efficient Sequential Data Classification on Resource-constrained Devices. In *Proceedings of the Thirty-first Annual Conference on Neural Information Processing Systems (NeurIPS)*. 10976–10987. all_papers/DennisPSJ18.pdf slides/DennisPSJ18.pdf.

[10] Caiwen Ding, Siyu Liao, Yanzhi Wang, Zhe Li, Ning Liu, Youwei Zhuo, Chao Wang, Xuehai Qian, Yu Bai, Geng Yuan, Xiaolong Ma, Yipeng Zhang, Jian Tang, Qinru Qiu, Xue Lin, and Bo Yuan. 2017. CNN: Accelerating and Compressing Deep Neural Networks Using Block-Circulant Weight Matrices. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture* (Cambridge, Massachusetts) *(MICRO-50 '17)*. Association for Computing Machinery, New York, NY, USA, 395–408. https://doi.org/10.1145/3123939.3124552

[11] Caiwen Ding, Shuo Wang, Ning Liu, Kaidi Xu, Yanzhi Wang, and Yun Liang. 2019. REQ-YOLO: A Resource-Aware, Efficient Quantization Framework for Object Detection on FPGAs. In *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays* (Seaside, CA, USA) *(FPGA '19)*. Association for Computing Machinery, New York, NY, USA, 33–42. https://doi.org/10.1145/3289602.3293904

[12] Amir Erfan Eshratifar, Mohammad Saeed Abrishami, and Massoud Pedram. 2021. JointDNN: An Efficient Training and Inference Engine for Intelligent Mobile Cloud Computing Services. *IEEE Transactions on Mobile Computing* 20, 2 (2021), 565–576. https://doi.org/10.1109/TMC.2019.2947893

[13] Mario Gerla, Eun-Kyu Lee, Giovanni Pau, and Uichin Lee. 2014. Internet of vehicles: From intelligent grid to autonomous cars and vehicular clouds. In *Internet of Things (WF-IoT), 2014 IEEE World Forum on*. IEEE, 241–246.

[14] Nikhil P Ghanathe, Vivek Seshadri, Rahul Sharma, Steve Wilton, and Aayan Kumar. 2021. MAFIA: Machine Learning Acceleration on FPGAs for IoT Applications. In *2021 31st International Conference on Field-Programmable Logic and Applications (FPL)*. 347–354. https://doi.org/10.1109/FPL53798.2021.00067

[15] Sridhar Gopinath, Nikhil Ghanathe, Vivek Seshadri, and Rahul Sharma. 2019. Compiling KB-sized Machine Learning Models to Tiny IoT Devices. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation* (Phoenix, AZ, USA) *(PLDI 2019)*. ACM, New York, NY, USA, 79–95. https://doi.org/10.1145/3314221.3314597

[16] Giuseppe Di Guglielmo, Javier Mauricio Duarte, Philip Harris, Duc Hoang, Sergo Jindariani, Edward Kreinar, Mia Liu, Vladimir Loncar, Jennifer Ngadiuba, Kevin Pedro, Maurizio Pierini, Dylan Sheldon Rankin, Sheila Sagear, Sioni Paris Summers, Nhan Tran, and Zhenbin Wu. 2020. Compressing deep neural networks on FPGAs to binary and ternary precision with HLS4ML. *Machine Learning: Science and Technology* (2020). http://iopscience.iop.org/10.1088/2632-2153/aba042

[17] Song Han, Jeff Pool, John Tran, and William J. Dally. 2015. Learning both Weights and Connections for Efficient Neural Networks. arXiv:1506.02626 [cs.NE]

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 770–778. https://doi.org/10.1109/CVPR.2016.90

[19] A. Howard, M. Sandler, B. Chen, W. Wang, L. Chen, M. Tan, G. Chu, V. Vasudevan, Y. Zhu, R. Pang, H. Adam, and Q. Le. 2019. Searching for MobileNetV3. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*. IEEE Computer Society, Los Alamitos, CA, USA, 1314–1324. https://doi.org/10.1109/ICCV.2019.00140

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

[21] Gao Huang, Danlu Chen, Tianhong Li, Felix Wu, Laurens van der Maaten, and Kilian Q. Weinberger. 2017. Multi-Scale Dense Networks for Resource Efficient Image Classification. *arXiv e-prints*, Article arXiv:1703.09844 (March 2017), arXiv:1703.09844 pages. arXiv:1703.09844 [cs.LG]

[22] Jonathan Huang, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara, Alireza Fathi, Ian Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama, et al. 2017. Speed/accuracy trade-offs for modern convolutional object detectors. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 7310–7311.

[23] FG Irwin et al. 1968. An isotropic 3x3 image gradient operator. *Presentation at Stanford AI Project* 2014, 02 (1968).

[24] Hyuk-Jin Jeong, InChang Jeong, Hyeon-Jae Lee, and Soo-Mook Moon. 2018. Computation Offloading for Machine Learning Web Apps in the Edge Server Environment. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. 1492–1499. https://doi.org/10.1109/ICDCS.2018.00154

[25] Hillol Kargupta. 2010. Onboard driver, vehicle and fleet data mining. US Patent 7,715,961.

[26] Yigitcan Kaya, Sanghyun Hong, and Tudor Dumitras. 2019. Shallow-Deep Networks: Understanding and Mitigating Network Overthinking. In *Proceedings of the 36th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 97)*, Kamalika Chaudhuri and Ruslan Salakhutdinov (Eds.). PMLR, 3301–3310. https://proceedings.mlr.press/v97/kaya19a.html

[27] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[28] Alex Krizhevsky. 2009. *Learning multiple layers of features from tiny images*. Technical Report. Citeseer.

[29] Aditya Kusupati, Manish Singh, Kush Bhatia, Ashish Kumar, Prateek Jain, and Manik Varma. 2018. FastGRNN: A Fast, Accurate, Stable and Tiny Kilobyte Sized Gated Recurrent Neural Network. In *Proceedings of the Thirty-first Annual Conference on Neural Information Processing Systems (NeurIPS)*. 9031–9042. all_papers/KusupatiSBKJV18.pdf slides/fastgrnn.pdf.

[30] Ilias Leontiadis, Stefanos Laskaridis, Stylianos I. Venieris, and Nicholas D. Lane. 2021. It's Always Personal: Using Early Exits for Efficient On-Device CNN Personalisation. In *Proceedings of the 22nd International Workshop on Mobile Computing Systems and Applications* (Virtual, United Kingdom) *(HotMobile '21)*. Association for Computing Machinery, New York, NY, USA, 15–21. https://doi.org/10.1145/3446382.3448359

[31] Yuyang Li, Yuxin Gao, Minghe Shao, Joseph T. Tonecha, Yawen Wu, Jingtong Hu, and Inhee Lee. 2021. Implementation of Multi-Exit Neural-Network Inferences for an Image-Based Sensing System with Energy Harvesting. *Journal of Low Power Electronics and Applications* 11, 3 (2021). https://doi.org/10.3390/jlpea11030034

[32] Yuyang Li, Yawen Wu, Xincheng Zhang, Jingtong Hu, and Inhee Lee. 2022. Energy-Aware Adaptive Multi-Exit Neural Network Inference Implementation for a Millimeter-Scale Sensing System. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 30, 7 (2022), 849–859. https://doi.org/10.1109/TVLSI.2022.3171308

[33] Shaohui Lin, Rongrong Ji, Chenqian Yan, Baochang Zhang, Liujuan Cao, Qixiang Ye, Feiyue Huang, and David Doermann. 2019. Towards Optimal Structured CNN Pruning via Generative Adversarial Learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

[34] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. 2014. Microsoft COCO: Common Objects in Context. *CoRR* abs/1405.0312 (2014). arXiv:1405.0312 http://arxiv.org/abs/1405.0312

[35] Nicolas Maisonneuve, Matthias Stevens, Maria E Niessen, Peter Hanappe, and Luc Steels. 2009. Citizen noise pollution monitoring. In *Proceedings of the 10th Annual International Conference on Digital Government Research: Social Networks: Making Connections between Citizens, Data and Government*. Digital Government Society of North America, 96–103.

[36] Luz Elena Y Mimbela and Lawrence A Klein. 2000. Summary of vehicle detection and surveillance technologies used in intelligent transportation systems.

[37] MLCommons. [n. d.]. Training scripts for MLPerf tiny . https://github.com/mlcommons/tiny/tree/master/benchmark/training. (Accessed on 10/28/2022).

[38] Ravi Teja Mullapudi, William R. Mark, Noam Shazeer, and Kayvon Fatahalian. 2018. HydraNets: Specialized Dynamic Architectures for Efficient Inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[39] Katayoun Neshatpour, Farnaz Behnia, Houman Homayoun, and Avesta Sasan. 2019. Exploiting Energy-Accuracy Trade-off through Contextual Awareness in Multi-Stage Convolutional Neural Networks. In *20th International Symposium on Quality Electronic Design (ISQED)*. 265–270. https://doi.org/10.1109/ISQED.2019.8697497

[40] Common objects in Context. [n. d.]. COCO val2017. http://images.cocodataset.org/zips/val2017.zip. (Accessed on 04/10/2023).

[41] Roberto G. Pacheco, Kaylani Bochie, Mateus S. Gilbert, Rodrigo S. Couto, and Miguel Elias M. Campista. 2021. Towards Edge Computing Using Early-Exit Convolutional Neural Networks. *Information* 12, 10 (2021). https://doi.org/10.3390/info12100431

[42] A. Parashar, M. Rhu, A. Mukkara, A. Puglielli, R. Venkatesan, B. Khailany, J. Emer, S. W. Keckler, and W. J. Dally. 2017. SCNN: An accelerator for compressed-sparse convolutional neural networks. In *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*. 27–40.

[43] Shishir G. Patil, Don Kurian Dennis, Chirag Pabbaraju, Nadeem Shaheer, Harsha Vardhan Simhadri, Vivek Seshadri, Manik Varma, and Prateek Jain. 2019. GesturePod: Enabling On-Device Gesture-Based Interaction for White Cane Users. In *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology* (New Orleans, LA, USA) *(UIST '19)*. Association for Computing Machinery, New York, NY, USA, 403–415. https://doi.org/10.1145/3332165.3347881

[44] Mary Phuong and Christoph Lampert. 2019. Towards Understanding Knowledge Distillation. In *Proceedings of the 36th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 97)*, Kamalika Chaudhuri and Ruslan Salakhutdinov (Eds.). PMLR, 5142–5151. https://proceedings.mlr.press/v97/phuong19a.html

[45] Jiantao Qiu, Jie Wang, Song Yao, Kaiyuan Guo, Boxun Li, Erjin Zhou, Jincheng Yu, Tianqi Tang, Ningyi Xu, Sen Song, Yu Wang, and Huazhong Yang. 2016. Going Deeper with Embedded FPGA Platform for Convolutional Neural Network. In *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays* (Monterey, California, USA) *(FPGA '16)*. Association for Computing Machinery, New York, NY, USA, 26–35. https://doi.org/10.1145/2847263.2847265

[46] H.A. Rowley, S. Baluja, and T. Kanade. 1998. Neural network-based face detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 20, 1 (1998), 23–38. https://doi.org/10.1109/34.655647

[47] Simone Scardapane, Michele Scarpiniti, Enzo Baccarelli, and Aurelio Uncini. 2020. Why Should We Add Early Exits to Neural Networks? *Cognitive Computation* 12, 5 (Jun 2020), 954–966. https://doi.org/10.1007/s12559-020-09734-4

[48] M. Sharifi, M. Fathy, and M.T. Mahmoudi. 2002. A classified and comparative study of edge detection algorithms. In *Proceedings. International Conference on Information Technology: Coding and Computing*. 117–120. https://doi.org/10.1109/ITCC.2002.1000371

[49] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going Deeper With Convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[50] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, and Quoc V. Le. 2018. MnasNet: Platform-Aware Neural Architecture Search for Mobile. *CoRR* abs/1807.11626 (2018). arXiv:1807.11626 http://arxiv.org/abs/1807.11626

[51] Surat Teerapittayanon, Bradley McDanel, and H.T. Kung. 2016. BranchyNet: Fast inference via early exiting from deep neural networks. In *2016 23rd International Conference on Pattern Recognition (ICPR)*. 2464–2469. https://doi.org/10.1109/ICPR.2016.7900006

[52] N. M. Thamrin, M. A. Haron, and Fazlina Ahmat Ruslan. 2011. A field programmable gate array implementation for biomedical system-on-chip (SoC). In *2011 IEEE 7th International Colloquium on Signal Processing and its Applications*. 187–191. https://doi.org/10.1109/CSPA.2011.5759870

[53] Naftali Tishby and Noga Zaslavsky. 2015. Deep learning and the information bottleneck principle. In *2015 IEEE Information Theory Workshop (ITW)*. 1–5. https://doi.org/10.1109/ITW.2015.7133169

[54] tokusumi. 2020. keras-flops calculator. https://pypi.org/project/keras-flops/. (Accessed on 08/04/2022).

[55] Deepak Vasisht, Zerina Kapetanovic, JongHo Won, Xinxin Jin, Ranveer Chandra, Sudipta Sinha, and Ashish Kapoor. 2017. FarmBeats: An IoT Platform for Data-Driven Agriculture. In *Networked Systems Design and Implementation (NSDI)* (networked systems design and implementation (nsdi) ed.). USENIX. https://www.microsoft.com/en-us/research/publication/farmbeats-iot-platform-data-driven-agriculture/

[56] Xin Wang, Yujia Luo, Daniel Crankshaw, Alexey Tumanov, Fisher Yu, and Joseph E. Gonzalez. 2018. IDK Cascades: Fast Deep Learning by Learning not to Overthink. arXiv:1706.00885 [cs.CV]

[57] Xin Wang, Fisher Yu, Zi-Yi Dou, Trevor Darrell, and Joseph E. Gonzalez. 2018. SkipNet: Learning Dynamic Routing in Convolutional Networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*.

[58] Pete Warden. 2018. Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition. *CoRR* abs/1804.03209 (2018). arXiv:1804.03209 http://arxiv.org/abs/1804.03209

[59] Pete Warden and Daniel Situnayake. 2019. *TinyML*. O'Reilly Media, Incorporated.

[60] Yawen Wu, Zhepeng Wang, Zhenge Jia, Yiyu Shi, and Jingtong Hu. 2020. Intermittent Inference with Nonuniformly Compressed Multi-Exit Neural Network for Energy Harvesting Powered Devices. In *Proceedings of the 57th ACM/EDAC/IEEE Design Automation Conference* (Virtual Event, USA) *(DAC '20)*. IEEE Press, Article 245, 6 pages.

[61] Qunliang Xing, Mai Xu, Tianyi Li, and Zhenyu Guan. 2020. Early Exit or Not: Resource-Efficient Blind Quality Enhancement for Compressed Images. In *Computer Vision – ECCV 2020*. Springer International Publishing, 275–292. https://doi.org/10.1007/978-3-030-58517-4_17

[62] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. 2014. How transferable are features in deep neural networks? *CoRR* abs/1411.1792 (2014). arXiv:1411.1792 http://arxiv.org/abs/1411.1792

[63] Yundong Zhang, Naveen Suda, Liangzhen Lai, and Vikas Chandra. 2017. Hello Edge: Keyword Spotting on Microcontrollers. *CoRR* abs/1711.07128 (2017). arXiv:1711.07128 http://arxiv.org/abs/1711.07128

A ARTIFACT APPENDIX

A.1 Abstract

This appendix describes the artifact associated with this work.

A.2 Artifact meta-information

- **Algorithm:** Early-exit technique for tinyCNNs
- **Program:** Tensorflow v2.3
- **Compilation:** Python v3.7, Tensorflow v2.3
- **Model:** Resnet-8, DSCNN, MobinetV1
- **Datasets:** CIFAR-10, Speech Commands, Visual wake words
- **Run-time environment:** Ubuntu Linux with miniconda
- **Hardware:** GeForce RTX 2080 GPU for model training
- **Run-time state:** set by our scripts
- **Metrics:** prediction accuracy, FLOPS
- **Output:** Accuracy vs FLOPS trade-off plot (benefit curve)
- **How much disk space required (approximately)?:** ~15GB
- **How much time is needed to prepare workflow?:** 30-40min
- **How much time is needed to complete experiments?:** Several hours. Each model takes around 4 to 5 hours to train on a GPU. Testing on either trained models (or on pretrained models included in artifact) takes ~45-60min
- **Publicly available?:** Yes. Datasets are also publicly available

A.3 Description

A.3.1 How to access. https://github.com/nikhilghanathe/t-recx

A.3.2 Hardware dependencies. The results reported are based on training done on a GeForce RTX 2080 GPU. The workflow can practically be run on any laptop/desktop with a GPU.

A.3.3 Software dependencies. Ubuntu Linux, Tensorflow=2.3, Cudatoolkit=10.1, Cudnn=7.6, Python=3.7. Other tested build configurations for tensorflow can be found at: https://www.tensorflow.org/install/source#gpu

A.4 Installation

T-RecX uses miniconda package manager. Installation instructions for Linux at: https://docs.conda.io/en/latest/miniconda.html

Prepare environment

```
$ conda create -n trecx python=3.7
$ conda activate trecx
```

Clone repository and run install script

```
$ git clone https://github.com/nikhilghanathe/t-recx.git
$ bash install_conda_packages.sh
$ pip install -r requirements.txt
```

A.5 Experiment workflow

The repository contains three directories: image_classification, keyword_spotting, visual_wake_words. These directories correspond to evaluation of 1) Resnet-8 on CIFAR-10, 2) DSCNN on Speech Commands and 3) MobilenetV1 on Visual wake words datasets respectively. The scripts for downloading datasets, preprocessing, training and testing have all been automated, and are included in each directory. The scripts included use model hyperparameters and architectures described in this paper. More details can be found in the `README.md` file in each directory.

The repository also contains pretrained models. As an alternative to several hours of training, the results presented in this work can be verified by running the test script included in each directory. Details on running the test scripts can be found in the corresponding `README.md` files.

A.6 Evaluation and expected results

The output from running the test scripts in each directory on the trained models is saved in the `results/` sub-directory. The test script generates the benefit curve plots of Figure 4, Figure 7 and Figure 6 of the paper.