TREC: Transient Redundancy Elimination-based Convolution

Jiawei Guan$^\diamond$, Feng Zhang$^\diamond$, Jiesong Liu$^\diamond$, Hsin-Hsuan Sung*, Ruofan Wu$^\diamond$, Xiaoyong Du$^\diamond$, Xipeng Shen*
$^\diamond$Key Laboratory of Data Engineering and Knowledge Engineering (MOE), and School of Information, Renmin University of China
*Computer Science Department, North Carolina State University
guanjw@ruc.edu.cn, fengzhang@ruc.edu.cn, liujiesong@ruc.edu.cn, hsung2@ncsu.edu, ruofanwu@ruc.edu.cn, duyong@ruc.edu.cn, xshen5@ncsu.edu

Abstract

The intensive computations in convolutional neural networks (CNNs) pose challenges for resource-constrained devices; eliminating redundant computations from convolution is essential. This paper gives a principled method to detect and avoid *transient redundancy*, a type of redundancy existing in input data or activation maps and hence changing across inferences. By introducing a new form of convolution (TREC), this new method makes *transient redundancy* detection and avoidance an inherent part of the CNN architecture, and the determination of the best configurations for redundancy elimination part of CNN backward propagation. We provide a rigorous proof of the robustness and convergence of TREC-equipped CNNs. TREC removes over 96% computations and achieves 3.51× average speedups on microcontrollers with minimal (about 0.7%) accuracy loss.

1 Introduction

Convolutional Neural Networks (CNNs) are computation intensive, making their deployment on resource-constrained devices (e.g., Microcontrollers equipped with 2MB memory) challenging. Removing computation redundancy in convolutions is hence an important way to speed up CNN inferences.

Depending on where the redundancy comes from, redundancy in CNN can be classified into *lasting redundancy* and *transient redundancy*. Lasting redundancy arises from CNN parameters. As the parameters typically stay unchanged after CNN is deployed for inference, such redundancy can be detected and removed through *offline* methods (e.g., DNN pruning [9, 10, 13, 23, 24, 35, 36] and quantization [16, 18, 27, 34]). Transient redundancy, on the other hand, arises from input data or activation maps, manifesting as similar tiles within an image or an activation map. Unlike *lasting redundancy*, *transient redundancy* has to be detected and removed in *every inference*, elusive to offline methods.

Although *transient redundancy* has been observed in images [26] and videos [17, 37, 38, 39], understanding to it is much less than to *lasting redundancy*. Prior studies have treated *transient redundancy* in an ad-hoc manner. Deep Reuse [26], for instance, diverts, before a convolution, the input images or activation maps from the DNN pipeline to a randomized (to avoid bias) online clustering component to detect the contained redundancy, and then redirects the clustering results back to the DNN. Such ad-hoc treatments work well sometimes on some data, but perform badly in other times (e.g., over 5% accuracy fluctuations in different runs). The reason is the lack of understanding to some fundamental aspects of *transient redundancy*. Examples include

(1) How to reach the optimal setting to minimize the accuracy loss caused by redundancy elimination while maximizing the eliminated redundancy?
(2) How to ensure a stable robust inference performance of *transient redundancy* elimination across runs?
(3) How to minimize the interference imposed by the *transient redundancy* elimination to the result of the converged training of CNN?

This paper proposes a principled way to detect and avoid *transient redundancy* for CNN inference. By introducing a new CNN operator named TREC, it fuses the detection and avoidance of *transient redundancy* into CNN, making them part of its inherent architecture. With the systematic design, finding the best configuration to maximize redundancy-elimination benefits becomes part of the CNN training process, seamlessly integrated as part of the backward propagation. Based on the design, we are able to investigate *transient redundancy* elimination for CNN in a principled way, providing a rigorous proof of the robustness and convergence for TREC, and an intuitive comparative analysis of the computational complexity.

TREC is compatible with both forward and backward propagations, enabling a plug-and-play replacement for convolutional layers in mainstream CNNs without any additional effort. Through a novel LSH back-propagation mechanism, the optimal parameters of TREC are automatically determined; they remain deterministic, clearing the threat to robust performance that hampers practical adoptions of the existing solutions.

We evaluate TREC on diverse popular CNNs, CifarNet [1], SqueezeNet (with and without complex bypass) [15], ZfNet [32], and ResNet-34[12] on a microcontroller (MCU). In particular, ZfNet is preprocessed with channel pruning to meet the strict memory constraints of MCU. Experiments show that by removing 96.25% *transient redundancy*, TREC achieves an average of 4.40$\times$ speedup compared to the conventional convolution operator. When applied to the full neural networks, TREC achieves an average of 3.51$\times$ speedup with virtually no accuracy loss.

2 Related Work

Extensive studies have been conducted to eliminate the *lasting redundancy* of CNNs. Factorization [4, 5, 6, 20, 22, 30, 33], pruning [9, 10, 13, 23, 24, 35, 36], and quantization [16, 18, 27, 34] techniques are some of the important methods. They exploit the redundancy contained in the CNN parameters.

Transient redundancy elimination is complementary to lasting redundancy elimination, dedicated to detecting and removing redundancy from input images or activation maps. It is more challenging to do as it has to happen during online inference. Such redundancy can be further categorized into *temporal* redundancy (e.g., similarities between adjacent video frames) and *spatial* redundancy (e.g., similar tiles within an image). Prior studies [7, 11, 19, 31, 37, 38] have paid attentions mostly to *temporal* similarities between adjacent frames. Deep Reuse [26] is the main method proposed before for detecting and eliminating spatial redundancy within an image for CNN. It takes place as extra operations outside CNN, using Locality-Sensitive Hashing (LSH) with some random hashing vectors for online data clustering. Such an ad-hoc treatment causes important uncertainty about the impact of the clustering errors on the CNN performance. Experimental results show that Deep Reuse causes significant (e.g., 5%) fluctuations on CNN accuracy (detailed in Section 5).

3 TREC

In this section, we elaborate on the design of our transient redundancy elimination-based convolution. First, we present the architecture of TREC and how it operates in the forward pass. Then, we delve into the key challenge in the creation of TREC, how to make back-propagation function on TREC-based CNNs.

3.1 Basic TREC Architecture

The basic architecture of our proposed TREC is shown in Figure 1. TREC comprises three components in series on top of *Im2col+GEMM* based convolution: LSH-based clustering that reduces the size of the input to the conversion, matrix multiplication of weights and the compressed input, and a recovery step that restores results to the original GEMM output sizes.

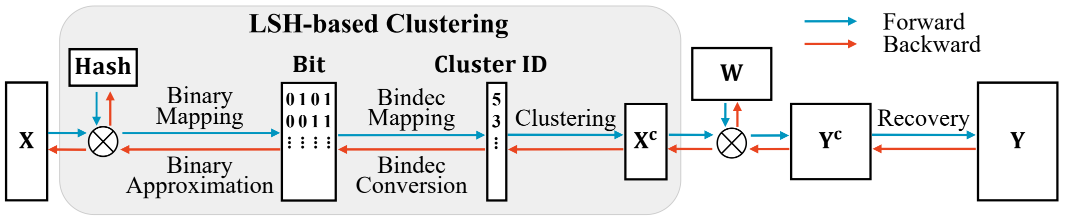

Figure 1: Architecture design of TREC. "$\otimes$" denotes matrix multiplication.

Recall that in *Im2col+GEMM* based convolution, the input feature maps and filters are unfolded to an input matrix X and a weight matrix W, and then the convolution is materialized as a matrix multiplication. At a high level, TREC uses Locality-Sensitive Hashing (LSH) to group vectors in the input matrix. It has four steps. First, it applies a hash function matrix *Hash* to X. Second, it performs a simple element-wise binary mapping to obtain a bit matrix. Third, each row vector of the bit matrix is converted to a decimal value by bindec mapping (i.e., a mapping from binary to decimal). These integer values indicate the cluster IDs of the neuron vectors in X. Fourth, the input matrix X reduces to X^c, which is composed of the centroids of clusters. In this way, the size of input matrix is significantly reduced, lowering the computational complexity of the subsequent matrix multiplication. Finally, in the recovery step, vectors in the same cluster reuse the computation results of the corresponding centroids to obtain the final result Y.

3.2 Breaking the Back-Propagation Barriers for TREC

The key challenges for building a TREC-based CNN are two. The first is in making back-propagation work with TREC. It is challenging because of the discrete nature of the LSH-based clustering in TREC. The second is to understand the impact of TREC on the convergentability of CNN.

We address the first challenge in this section and address the second challenge in Section 4. At a high level, our strategy consists of two key components. First, we replace the piecewise mappings in TREC with similar-shaped functions as binary approximation approaches. Second, we propose a bindec conversion algorithm to resolve the complexities in obtaining centroids. Before explaining them, we first give a closer examination of the operations in the forward pass of the basic TREC, which will help the follow-up explanations.

A Closer Look at Redundancy Elimination in the Forward Pass. Figure 2 illustrates the redundancy elimination in the forward pass of the basic TREC. Specifically, let $X \in \mathbb{R}^{4\times 6}$ be an unfolded input matrix after *Im2col*. TREC first slices X into L groups vertically ($L = 2$ in Figure 2 *Left*), so each sub-matrix is of size 4×3. The slicing reduces vector length which can often allow the clustering to identify more similar vectors [25]. Applying the LSH clustering to each sub-matrix produces the centroid matrix X^c.

LSH is chosen because as a lightweight online clustering method, it produces good clustering results without incurring excessive overhead. Figure 2 *Right* shows more details of LSH clustering with an example. X_1 is the first sub-matrix of X in Figure 2 *Left*. For each input vector x in X_1, it is hashed through the following function parameterized with a random vector v:

$$h_v(x) = \begin{cases} 1 & if \quad v \cdot x > 0 \\ 0 & if \quad v \cdot x \leq 0 \end{cases} \tag{3.1}$$

First, X_1 is transformed into a projected matrix with $H = 2$ random hash vectors (i.e., 2 column vectors of the hash matrix). Second, the projected vectors are mapped to bit vectors with 2^H possibilities, as defined by Eq. 3.1. Third, the bindec rule converts each bit vector to an integer value, which can be used as the cluster ID of vector x. In this way, similar vectors have a high chance of being mapped to the same cluster. Finally, each vector in X^c is computed using vectors with the same cluster ID. Note that in practice, H is much smaller than the length of neural vectors, so the overhead introduced by LSH is low.

Binary Approximation. To solve the problem of discontinuity in binary mapping during back-propagation, we introduce binary approximation to make it differentiable. The binary mapping handles the projected matrix in an element-wise manner, which acts as a binary classification that sets

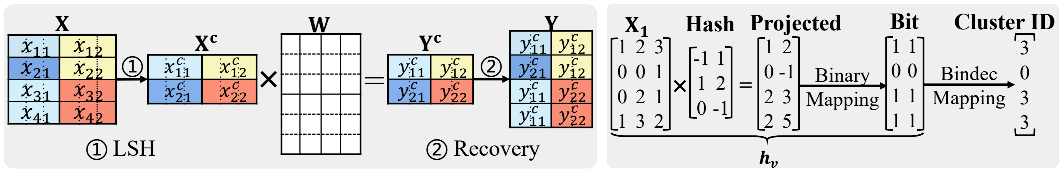

Figure 2: *Left*: The illustration of redundancy elimination. In matrix X, vectors in the same color belong to the same cluster. *Right*: An illustration of LSH. X_1 is the first sub-matrix of X on the left.

elements greater than zero to one, and otherwise to zero. To that end, we employ a sigmoid function as a substitute of the binary mapping:

$$S(x) = \frac{1}{1 + e^{-a \times (x-b)}}, \tag{3.2}$$

where a denotes the slope and b denotes the abscissa value of the central point of the curve. The sigmoid function suits TREC because it is 1) monotonic and bounded; 2) smooth and differentiable. Meanwhile, the sigmoid function offers a parameterized way to materialize the effects of binary mapping for TREC. The parameters a and b in the sigmoid function are hyperparameters determined empirically. When setting them, the rationale worth mentioning is that the value of a shall be large such that as few points as possible appear at the mid-slope of the sigmoid function, which would allow positive and negative numbers to approach one and zero respectively. And the value of b shall shift the function slightly to the right, which would allow the value at $x = 0$ to approach zero. The bit vectors can be then approximated by an application of such a sigmoid function to the projected matrix.

Bindec Conversion. Another step that impedes back-propagation for TREC is bindec mapping, that is, the rightmost step in the example in Figure 2, which converts every binary vector into its corresponding integer value. To make this process differentiable, we design a transformation matrix as follows:

$$\begin{bmatrix} 2^{H-1}/1 & 2^{H-1}/2 & \cdots & 2^{H-1}/2^H \\ \vdots & \vdots & \ddots & \vdots \\ 2^1/1 & 2^1/2 & \cdots & 2^1/2^H \\ 2^0/1 & 2^0/2 & \cdots & 2^0/2^H \end{bmatrix} \in \mathbb{R}^{H \times 2^H}. \tag{3.3}$$

Eq. 3.3 shows that elements in the same row of the transformation matrix have the same numerator, and elements in the same column have the same denominator. Horizontally, the denominators increase from 1 to 2^H. Vertically, the numerators are $[2^{H-1}, ..., 2^1, 2^0]$ from top to bottom, which are the coefficients when converting a binary number to decimal.

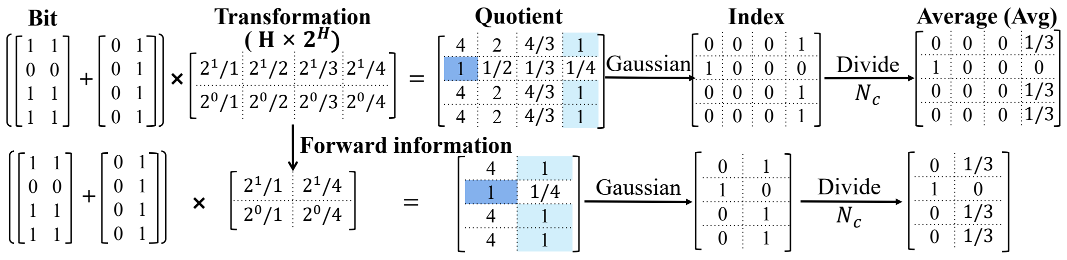

Figure 3: The overall idea of constructing the Avg matrix.

The overall idea of bindec conversion is shown in Figure 3, which follows the example of Figure 2. First, we add 1 to the rightmost column of the bit matrix. In this way, the range of the integers changes from $[0, 2^H - 1]$ to $[1, 2^H]$, which is consistent with the denominators of the transformation matrix. The result matrix is then multiplied with the transformation matrix, which can be seen as two steps: 1) multiply the bit vectors with the numerators of the transformation matrix's column vectors to obtain integer values; 2) divide the integer values by the denominators of the column vectors. As a result, in the quotient matrix, for row i where vector x_i is located, $Quotient_{i,j} = 1$ appears only at the position where the column number j equals x_i's cluster ID. For example, in Figure 3, the bit vector in the second row (i.e., row 1) is $[0, 0]$, and the cluster ID is 0. In row 1 of the quotient matrix, only column 0 has 1, i.e., $Quotient_{1,0} = 1$.

The method can successfully do the mapping. But as H increases, the dimension of the transformation matrix increases exponentially, resulting in a huge memory waste. To ease the space and computation, we apply the above process only in the backward pass, while the forward pass still uses binary and bindec mappings. This means that the results of the forward pass such as cluster IDs, the number of vectors per cluster (denoted as N_c), and the number of clusters (denoted as C), can be reused by back-propagation. Therefore, we can directly construct a transformation matrix of size $H \times C$, where each column represents a known cluster ID that is derived from the forward calculation. As shown in the lower part of Figure 3, the use of forward information can significantly avoid space waste.

Now, we can obtain an index matrix to indicate the vector-cluster relationship. We can do that by applying a element-wise function to the quotient matrix, which gives 1 if the element is 1 and 0 otherwise. Such a function is however not differentiable. To solve this problem, we introduce an impulse-like Gaussian function of the following form:

$$G(x) = e^{-\frac{(x-1)^2}{2\sigma^2}}, \tag{3.4}$$

It is a symmetrical bell-shaped curve centered at $x = 1$, with σ controlling its width. With a small σ, the Gaussian function can be used to approximate the desired index matrix. Then, we divide the index matrix by the number of vectors per cluster N_c, and we obtain an average matrix denoted as Avg. Finally, the cluster centroids can be calculated as follows:

$$X^c = Avg^T \cdot X. \tag{3.5}$$

For the example in Figure 2, we get the cluster centroids as follows:

$$\begin{bmatrix} \frac{1}{3} & 0 & \frac{1}{3} & \frac{1}{3} \\ 0 & 1 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1 & 2 & 3 \\ 0 & 0 & 1 \\ 0 & 2 & 1 \\ 1 & 3 & 2 \end{bmatrix} = \begin{bmatrix} \frac{2}{3} & \frac{7}{3} & 2 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.6}$$

Parameter Setting. TREC uses Sigmoid and Gaussian functions to simulate the mappings. As mentioned, TREC needs the range of function parameters to meet certain conditions, such as using a small σ to narrow the width of the Gaussian function. However, fixing a σ value is subject to gradient explosion, resulting in a large gradient variance and convergence difficulties. Therefore, we propose a solution that dynamically sets σ based on H.

We use Figure 4 to illustrate the gradient explosion problem. The curves are of a Gaussian function with $\sigma = 1$ and its derivative. We can see that the derivative function varies widely on both sides of $x = 1$, and its maximum and minimum values occur at $x = 1 \pm \sigma$. The value range of the derivative function increases as σ increases. Therefore, manually setting a fixed σ has a high chance of incurring a large gradient variance.

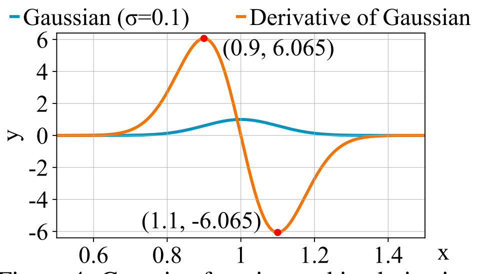

Figure 4: Gaussian function and its derivative with $\sigma = 1$.

To address this problem, we first analyze the quotient matrix. According to the rules of the transformation matrix, the quotient matrix takes values in the range $[\frac{1}{2^H}, 2^H]$, and the difference is at least $\frac{1}{2^H}$. Moreover, the i^{th} vector has $Quotient_{i,j} = 1$ when the cluster ID is equal to column j. Based on the observations, we avoid producing large gradients simply by ensuring

$$\sigma < \frac{1}{k \cdot 2^H} \quad and \quad k > 1. \tag{3.7}$$

Consequently, the Gaussian function not only provides an ideal approximation of the index matrix, but also ensures the stability of the training under any H value.

4 Theoretical Analysis

In this section, we provide a theoretical analysis of TREC in terms of robustness, convergence, and complexity. All proofs are available in the Appendix.

4.1 Robustness

Neural networks are sensitive to inputs [8, 28]. In practice, a small perturbation of the input image can misdirect the neural network and significantly decrease its classification accuracy. To prove that TREC does not impair the stability of the convolutional layer, we use the *Lipschitz constant*, which is commonly used to assess the robustness of neural networks, to upper bound the relationship between the input perturbation and output variation [29]. In the following, we use $\|X\|_\infty$ to denote the L_∞-norm of the matrix space $\mathbb{R}^{m\times n}$.

Definition 1. *A function* $f : \mathbb{R}^{m_1\times n_1} \to \mathbb{R}^{m_2\times n_2}$ *is called Lipschitz continuous if there exists a constant* $L > 0$ *such that*

$$\forall X, Y \in \mathbb{R}^{m\times n}, \quad \|f(X) - f(Y)\|_\infty \leq L\|X - Y\|_\infty. \tag{4.1}$$

The smallest L that makes the previous inequality true is called the *Lipschitz constant* of f and can be denoted as $L(f)$.

Suppose $Conv : X \to X\cdot W$ is a conventional convolution operator, and $TREC : X \to Avg\cdot X\cdot W$ is our proposed method. Recall the description of the Avg matrix in Section 3.2 and Eq. 3.6; $\sum_i Avg_{i,j} \leq 1$ is satisfied for any column j. Combining Definition 1 with the above discussion, we have:

$$L(Conv) = \|W\|_\infty, \quad L(TREC) = \|W \otimes Avg^T\|_\infty \leq \|W\|_\infty, \tag{4.2}$$

where $\otimes$ denotes the Kronecker product. In summary, under the same network configuration, the Lipschitz constant of TREC is no larger than that of the conventional convolution. That is to say, TREC has the ability to maintain the robustness of convolutional layers.

4.2 Convergence

In this section, we provide a theoretical analysis of the convergence of TREC-equipped CNNs. Analyzing the convergence of the above CNNs directly is difficult, since the specific CNN architectures are unknown except for the convolutional layers. Instead, we address this problem by building a connection with stochastic gradient descent, and prove that the proposed method is guaranteed to converge under reasonable assumptions. In the following, we use $\|x\|_2$ to denote the L_2-norm of the Hilbert space $\mathbb{R}^n$.

We assume a CNN network with R TREC layers and T other layers. Without loss of generality, we use $W = [W_1, \dots, W_{R+T}] \in \mathbb{R}^{d_1}$ to denote the weights of all layers, and $H = [H_1, \dots, H_R] \in \mathbb{R}^{d_2}$ to denote the hash functions of all convolutional layers. In accordance with the method [2], we represent a sample (or set of samples) as a random seed ξ, and refer to the loss for a given (W, H, ξ) as $f(W, H; \xi)$. In addition, given a dataset with n samples, we refer to the objective function $F : \mathbb{R}^D \to \mathbb{R}$ as either $F(W, H) = \mathbb{E}[f(W, H; \xi)]$ or $F(W, H) = \dfrac{1}{n}\sum_{i=1}^n f_i(W, H)$.

Following the stochastic gradient descent, we define the parameter update rule at iteration k as:

$$W_{k+1} \leftarrow W_k - \alpha_k \cdot g(W_k, H_k; \xi_k), \quad H_{k+1} \leftarrow H_k - \alpha_k \cdot q(W_k, H_k; \xi_k), \tag{4.3}$$

where α_k denotes the stepsize, $g(W_k, H_k; \xi_k)$ and $q(W_k, H_k; \xi_k)$ represent the gradients of the loss function with respect to W_k and H_k separately, which can be the stochastic gradients or the mean gradients of a mini-batch:

$$g(W_k, H_k; \xi_k) = \begin{cases} \nabla f_W(W_k, H_k; \xi_k) \\ \frac{1}{n_k}\sum\limits_{i=1}^{n_k} \nabla f_W(W_k, H_k; \xi_k) \end{cases}, q(W_k, H_k; \xi_k) = \begin{cases} \nabla f_H(W_k, H_k; \xi_k) \\ \frac{1}{n_k}\sum\limits_{i=1}^{n_k} \nabla f_H(W_k, H_k; \xi_k). \end{cases} \tag{4.4}$$

Now we summarize the key assumptions required to establish our results.

Assumption 1 (Lipschitz-continuous objective gradients)**.** *The objective function* $F : \mathbb{R}^D \to \mathbb{R}$ *is continuously differentiable and the partial derivatives of F, namely,* $\nabla F_W : \mathbb{R}^{d_1} \to \mathbb{R}^{d_1}$ *and* $\nabla F_H : \mathbb{R}^{d_2} \to \mathbb{R}^{d_2}$*, are Lipschitz continous with Lipschitz constants* $L > 0$*, i.e.,*

$$\begin{aligned} \|\nabla F_W(W, H) - \nabla F_W(\overline{W}, H)\|_2 &\leq L\|W - \overline{W}\|_2, \quad \textit{for all } \{W, \overline{W}\} \subset \mathbb{R}^{d_1}, \\ \|\nabla F_H(W, H) - \nabla F_H(W, \overline{H})\|_2 &\leq L\|H - \overline{H}\|_2, \quad \textit{for all } \{H, \overline{H}\} \subset \mathbb{R}^{d_2}. \end{aligned} \tag{4.5}$$

Assumption 1 ensures that the partial derivatives of F do not change arbitrarily fast with respect to the parameters, thus well indicating how far to move to decrease F. To move forward, we denote the variances of $g(W_k, H_k; \xi_k)$ and $q(W_k, H_k; \xi_k)$ as:

$$\mathbb{V}_{\xi_k}[h(W_k, H_k; \xi_k)] := \mathbb{E}_{\xi_k}\left[\|h(W_k, H_k; \xi_k)\|_2^2\right] - \left\|\mathbb{E}_{\xi_k}[h(W_k, H_k; \xi_k)]\right\|_2^2, \quad h \in \{g, q\}. \tag{4.6}$$

Assumption 2 (First and second moment limits)**.** *The objective function and the stochastic gradient algorithm satisfy the following conditions:*

(a) The sequence of iterations $\{(W_k, H_k)\}$ is contained in an open set over which F is bounded below by a scalar F_{inf}.

(b) There exist scalars $\mu_G \geq \mu > 0$ such that, for all $k \in \mathbb{N}$,

$$\nabla F_W(W_k, H_{k+1})^T \mathbb{E}_{\xi_k}[g(W_k, H_k; \xi_k)] \geq \mu \|\nabla F_W(W_k, H_k)\|_2^2, \tag{4.7a}$$
$$\nabla F_H(W_k, H_k)^T \mathbb{E}_{\xi_k}[q(W_k, H_k; \xi_k)] \geq \mu \|\nabla F_H(W_k, H_k)\|_2^2, \tag{4.7b}$$
$$\|\mathbb{E}_{\xi_k}[g(W_k, H_k; \xi_k)]\|_2 \leq \mu_G \|\nabla F_W(W_k, H_k)\|_2, \tag{4.7c}$$
$$\|\mathbb{E}_{\xi_k}[q(W_k, H_k; \xi_k)]\|_2 \leq \mu_G \|\nabla F_H(W_k, H_k)\|_2. \tag{4.7d}$$

(c) There exist scalars $M \geq 0$ and $M_V \geq 0$ such that, for all $k \in \mathbb{N}$,

$$\begin{aligned} \mathbb{V}_{\xi_k}[g(W_k, H_k; \xi_k)] &\leq M + M_V \|\nabla F_W(W_k, H_k)\|_2^2, \\ \mathbb{V}_{\xi_k}[q(W_k, H_k; \xi_k)] &\leq M + M_V \|\nabla F_H(W_k, H_k)\|_2^2. \end{aligned} \tag{4.8}$$

We first analyze the convergence with a fixed stepsize and prove that the CNNs equipped with TREC can converge sub-linearly to the neighborhood of the critical points for the non-convex problem under the above assumptions.

Theorem 1 (Fixed stepsize)**.** *Assume that Assumptions 1 and 2 hold, and the fixed stepsize is $\alpha_k = \bar{\alpha}$ for all $k \in \mathbb{N}$ satisfying $0 < \bar{\alpha} \leq \frac{\mu}{LM_G}$. Then, the average-squared partial derivatives of F corresponding to the stochastic gradient iterations satisfy the following inequality for all $K \in \mathbb{N}$:*

$$\frac{1}{K}\sum_{k=1}^{K} \mathbb{E}\left[\|\nabla F_W(W_k, H_k)\|_2^2 + \|\nabla F_H(W_k, H_k)\|_2^2\right] \leq \frac{2\bar{\alpha}LM}{\mu} + \frac{2(F(W_1, H_1) - F_{inf})}{K\mu\bar{\alpha}}. \tag{4.9}$$

Therefore, the optimal solution we can obtain is dominated by $\frac{2\bar{\alpha}LM}{\mu}$. Further, we also prove the convergence with a dimishing stepsize that meets the requirements in the study [2]:

$$\lim_{K \to \infty} \sum_{k=1}^{K} \alpha_k = \infty \quad and \quad \lim_{K \to \infty} \sum_{k=1}^{K} \alpha_k^2 < \infty. \tag{4.10}$$

Theorem 2 (Dimishing stepsize)**.** *Assume that Assumptions 1 and 2 hold and the dimishing stepsize sequence $\{\alpha_k\}$ satisfies Eq. 4.10. Then, with $A_K := \sum_{k=1}^{K} \alpha_k$,*

$$\mathbb{E}\Big[\tfrac{1}{A_K} \sum_{k=1}^{K} \alpha_k \big(\|\nabla F_W(W_k, H_k)\|_2^2 + \|\nabla F_H(W_k, H_k)\|_2^2\big)\Big] \leq \tfrac{2(F(W_1, H_1) - F_{inf})}{\mu A_k} + \tfrac{2LM}{\mu A_k} \sum_{k=1}^{K} \alpha_k^2 \xrightarrow{K \to \infty} 0. \tag{4.11}$$

Remark 1. *Theorem 2 has shown the decaying of gradients $\|\nabla F_W(W_k, H_k)\|_2$ and $\|\nabla F_H(W_k, H_k)\|_2$ even with noise when the stepsize is dimishing. This is because the second condition in Eq. 4.10 implies the right-hand side of Eq. 4.11 to converge to a finite limit as K increases.*

Theorems 1 and 2 prove convergence in the limit. Taking a step forward, we study the theoretical bounds to understand the convergence of TREC-equipped CNNs in practical applications. According to Eq. 4.9 and 4.11, as long as the practical application satisfies Assumptions 1 and 2, we can obtain a theoretical upper bound of the sum of squared gradients without considering the limit. The relevance of this theoretical upper bound to practical applications is reflected in the values of the constants such as L, μ, M, and the specific setting of the learning rate. From Eqs. 4.5- 4.8, the constants L, M, and $M_G = M_v + \mu_G^2$, which are greater than 0, exist according to the L2-norm. Through experimental verification, we find that the constant μ is always greater than 0 during the training process and gradually converges to 1 with time, satisfying the sufficient direction assumption. We present in Appendix B the trend of the constant μ.

4.3 Complexity

We show in Table 1 an intuitive comparative analysis of two major complexity metrics. The complexity of binary and bindec mappings are not taken into account because they are parameterless and insignificant compared with other computational overheads. For the parameter count, TREC exceeds the conventional convolution by a hash matrix of $O(L \cdot H)$ size, with trivial KB-level space occupancy. For the computation complexity, we define *redundancy ratio*: $r_t = 1 - N_c/N$ as the size reduction of the input matrix to indicate the fraction of *transient redundancy* within input images or activation maps. As long as $H < M \cdot r_t$ is satisfied, TREC can yield computational benefits. In practical validation (Section 5.2), we notice that the reduction of *transient redundancy* of a single convolutional layer reaches up to 96.25%, which brings confidence to the advantages of TREC.

Table 1: Parameter counts and FLOPs for conventional GEMM-based convolution and TREC. The input and weight matrices unfolded by Im2col are of size $N \times K$ and $K \times M$. The hash matrix of TREC is of size $L \times H$. N_c denotes the number of clusters.

	Parameters	FLOPs
Conventional	$K \cdot M$	$N \cdot K \cdot M$
TREC	$K \cdot M + L \cdot H$	$(\frac{H}{M} + 1 - r_t) \cdot N \cdot K \cdot M$

5 Evaluation

5.1 Experimental Setup

Methodology. To demonstrate the efficacy of TREC on CNN inference acceleration, we first apply TREC to single convolutional layers to measure single-layer speedups and accuracy. Second, we apply TREC to the complete neural networks to measure end-to-end inference performance. Third, we analyze the influence of different factors on performance, including the dynamic setting of the Gaussian parameter, and the clustering configurations.

Platforms. As computation reduction is most needed on resource constrained devices, we focus our evaluation on Microcontrollers (MCU). Specifically, all inferences are performed on an STM32F469NI MCU with 324KB SRAM and 2MB Flash, using the CMSIS-NN kernel optimized for Arm Cortex-M devices [21]. All trainings are performed using PyTorch 1.10.1 (open-source software with a BSD license) on a machine equipped with a 20-core 3.60GHz Intel Core i7-12700K processor, 128GB of RAM, and an NVIDIA GeForce RTX A6000 GPU with 48 GB memory.

Workloads. We evaluate TREC with five compact CNNs that fit MCU, CifarNet [1], ZfNet [32], SqueezeNet with and without complex bypass [15], and ResNet-34 [12]. Most trainings and inferences are performed on CIFAR-10, while for ResNet, we use the downsampled ImageNet [3] with 64×64 resolution (since the original large images cause ResNet-34 to run out of MCU memory). All networks are optimized by SGD. The learning rate starts from 0.001 and decreases by 0.1 every 15 epochs. The batch size, weight decay, and momentum are set to 256, 0.9, and 10^{-4}, respectively, and the maximal epoch is set to 100. The batch size at inference is 1 due to the stringent memory limit on MCU. It is worth noting that ZfNet is preprocessed with channel pruning to meet the strict memory constraints of MCU. It also allows us to see how well the transient redundancy elimination (TREC) works in the presence of lasting redundancy elimination (pruning).

5.2 Single-Layer Performance

We report in Appendix C the single-layer speedups and accuracy loss achieved by replacing the conventional convolution with TREC. For comparison, we also include the results from Deep Reuse [26], a state-of-the-art method in avoiding *transient redundancy* in CNN. We have the following observations. First, TREC detects and eliminates about 96.25% of the *transient redundancy*. Second, TREC brings significant performance benefits for individual layers with an average speedup of 4.40×. Especially for the SqueezeNet expand layers, TREC gets up to 18.66× speedup, indicating that TREC is especially beneficial to computation-intensive layers. The speedup is lower than the ratio of *transient redundancy* that TREC eliminates, due to the overhead introduced by the clustering. Third, the influence on accuracy by TREC is small and mixed. When applied to some layers, it may cause around 0.8% accuracy drop, but it may also give slight accuracy improvements when applied to some other layers. TREC overall preserves the accuracy much better (about 3.33% on average) than Deep Reuse does. Furthermore, the *Conf.* columns report the LSH configurations in our experiments (Section 5.4 gives more details on the impacts of the hyperparameters L and H). We also illustrate the impact of batch size on performance by measuring the redundancy ratio of a single layer in Appendix C.2 , which shows that TREC has strong redundancy removal ability at any batch size, and that this ability increases with batch size. All above results demonstrate that TREC can exert its full potential, achieving significant speedups while minimizing the accuracy loss.

5.3 End-to-End Performance

Table 2 compares the end-to-end performance brought by TREC, Deep Reuse, and conventional convolution. Given the inherent randomness of Deep Reuse, we present its result interval of 150 runs. Note that after channel pruning, the baseline accuracy of ZfNet is slightly lower than its standard accuracy of 83%, but the model size is 7.6% of the original model, which can well meet the memory constraint of MCU.

Table 2: End-to-end performance comparison.

Network	Conv Method	Average Time per Image (ms)	Top-1 Accuracy (%)
CifarNet	Conventional	217.32	78.2
	Deep Reuse	154.44	73.2 $\sim$ 76.1
	TREC	153.92	76.5
ZfNet	Conventional	3557.32	80.1
	Deep Reuse	814.03	72.5 $\sim$ 76.6
	TREC	814.01	78.9
Vanilla SqueezeNet	Conventional	1639.51	83.5
	Deep Reuse	328.97	79.8 $\sim$ 81.9
	TREC	327.90	83.0
SqueezeNet + Complex Bypass	Conventional	1998.86	85.3
	Deep Reuse	543.71	80.5 $\sim$ 83.1
	TREC	544.03	84.6
ResNet-34 (ImageNet-64$\times$64)	Conventional	4242.77	52.6
	Deep Reuse	1379.26	46.7 $\sim$ 49.9
	TREC	1378.75	52.2

Several findings can be drawn from Table 2. First, compared to the conventional convolution, TREC achieves 3.51$\times$ speedup on average with minor (about 0.7%) accuracy loss. Second, TREC and Deep Reuse have comparable inference time, but TREC is superior in terms of stable performance and higher accuracy. Third, TREC cuts the accuracy loss of Deep Reuse from as much as 7.6% to 0.4$\sim$1.7%, an up to 84% reduction.

5.4 Configuration Analysis

Dynamic and Static σ of Gaussian. In this set of experiments, we measure the effect of the strategy proposed in Section 3.2 to dynamically set σ based on H. To eliminate the influence of irrelevant factors, we choose the most computation-intensive convolutional layers among four networks and replace them with TREC. As shown in Figure 5, we compare our proposed dynamic strategy with the static approach that manually sets a fixed σ value of 10^{-4}. As seen, the accuracy of the models using a fixed σ drops sharply after $H = 12$, and it shows a decreasing trend as H increases. Recalling the discussion in Section 3.2, when $H > 12$, we have $\sigma = 10^{-4} > \frac{1}{k \cdot 2^H}$. Therefore, the gradient of the hash matrix tends to fall into the maximum value range of the Gaussian derivatives, resulting in a large gradient variance and difficulty in model convergence. Thus, our dynamic σ strategy can effectively guarantee the performance.

Impacts of Hyperparameters L and H Two hyperparameters in TREC are 1) L: the width of the sub-matrices obtained by slicing, and 2) H: the number of hash functions. In application of TREC, they can be chosen empirically like other CNN hyperparameters. In this part, we provide some observations on their impacts. Figure 6 shows the accuracy of TREC, Deep Reuse, and conventional convolution under different L and H configurations. The upper part is the result of changing H with

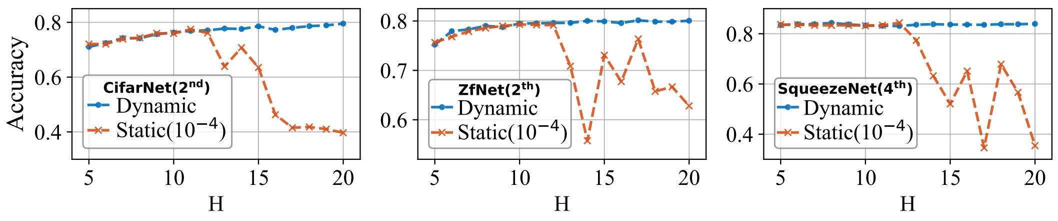

Figure 5: Effects of dynamic and static settings of σ.

fixed L, and the range of H is set to 5 to 20. The lower part is the result of changing L with fixed H, and the range of L changes according to the input and weight dimensions of different layers. We have the following observations: 1) TREC and Deep Reuse show a consistent trend of accuracy, which goes up as H increases, and goes down as L increases. 2) TREC has robust and stable inference performance, which is preferable to the non-deterministic Deep Reuse. 3) TREC is more inclined to stay at high accuracy, proving its ability to mine richer data features.

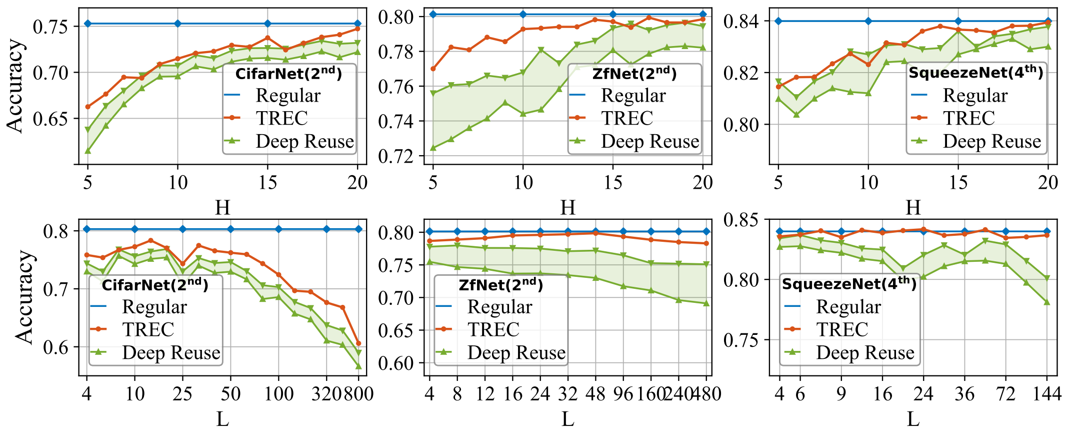

Figure 6: Influence of LSH parameter configuration on accuracy.

6 Conclusion

In this paper, we propose a principled approach to detecting and avoiding *transient redundancy*. It introduces a new form of convolutional operator called TREC, which integrates the redundancy avoidance as well as the optimal configuration determination into CNN as an intrinsic part of the CNN architecture. We experimentally and theoretically demonstrate the effectiveness of TREC. This new approach may open many new possibilities for CNN on resource-constrained devices. As a foundational research outcome, it is neutral in terms of societal impacts. A limitation of this study is that it focuses on CNNs, but transient redundancy may also exist in other types of Neural Networks as well, exploration of which is left for future to explore.

Acknowledgments and Disclosure of Funding

This material is supported by the National Natural Science Foundation of China (No. 62072458, 62172419, 61732014, and 62072459) and Beijing Nova Program, and is based upon work supported by the National Science Foundation (NSF) under Grants CCF-1703487, CCF-2028850 and CNS-1717425. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of NSF. Feng Zhang is the corresponding author of this paper.

References

[1] CifarNet. `http://places.csail.mit.edu/deepscene/small-projects/TRN-pytorch-pose/model_zoo/models/slim/nets/cifarnet.py`, 2020.

[2] Léon Bottou, Frank E Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *Siam Review*, 60(2):223–311, 2018.

[3] Patryk Chrabaszcz, Ilya Loshchilov, and Frank Hutter. A downsampled variant of imagenet as an alternative to the cifar datasets. *arXiv preprint arXiv:1707.08819*, 2017.

[4] Misha Denil, Babak Shakibi, Laurent Dinh, Marc'Aurelio Ranzato, and Nando De Freitas. Predicting parameters in deep learning. *Advances in neural information processing systems*, 26, 2013.

[5] Emily L Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. *Advances in neural information processing systems*, 27, 2014.

[6] Haisong Ding, Kai Chen, Ye Yuan, Meng Cai, Lei Sun, Sen Liang, and Qiang Huo. A compact CNN-DBLSTM based character model for offline handwriting recognition with Tucker decomposition. In *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, volume 1, pages 507–512. IEEE, 2017.

[7] Christoph Feichtenhofer, Axel Pinz, and Andrew Zisserman. Detect to track and track to detect. In *Proceedings of the IEEE international conference on computer vision*, pages 3038–3046, 2017.

[8] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[9] Song Han, Huizi Mao, and William J Dally. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. *arXiv preprint arXiv:1510.00149*, 2015.

[10] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. *Advances in neural information processing systems*, 28, 2015.

[11] Wei Han, Pooya Khorrami, Tom Le Paine, Prajit Ramachandran, Mohammad Babaeizadeh, Honghui Shi, Jianan Li, Shuicheng Yan, and Thomas S Huang. Seq-NMS for Video Object Detection. *arXiv preprint arXiv:1602.08465*, 2016.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[13] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE international conference on computer vision*, pages 1389–1397, 2017.

[14] Zhouyuan Huo, Bin Gu, and Heng Huang. Training neural networks using features replay. *Advances in Neural Information Processing Systems*, 31, 2018.

[15] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size. *arXiv preprint arXiv:1602.07360*, 2016.

[16] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2704–2713, 2018.

[17] Dinesh Jayaraman and Kristen Grauman. Slow and steady feature analysis: higher order temporal coherence in video. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3852–3861, 2016.

[18] Sangil Jung, Changyong Son, Seohyung Lee, Jinwoo Son, Jae-Joon Han, Youngjun Kwak, Sung Ju Hwang, and Changkyu Choi. Learning to quantize deep networks by optimizing quantization intervals with task loss. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4350–4359, 2019.

[19] Kai Kang, Hongsheng Li, Junjie Yan, Xingyu Zeng, Bin Yang, Tong Xiao, Cong Zhang, Zhe Wang, Ruohui Wang, Xiaogang Wang, et al. T-CNN: Tubelets with Convolutional Neural Networks for Object Detection from Videos. *IEEE Transactions on Circuits and Systems for Video Technology*, 28(10):2896–2907, 2017.

[20] Yong-Deok Kim, Eunhyeok Park, Sungjoo Yoo, Taelim Choi, Lu Yang, and Dongjun Shin. Compression of deep convolutional neural networks for fast and low power mobile applications. *arXiv preprint arXiv:1511.06530*, 2015.

[21] Liangzhen Lai, Naveen Suda, and Vikas Chandra. CMSIS-NN: Efficient Neural Network Kernels for Arm Cortex-M CPUs. *arXiv preprint arXiv:1801.06601*, 2018.

[22] Vadim Lebedev, Yaroslav Ganin, Maksim Rakhuba, Ivan Oseledets, and Victor Lempitsky. Speeding-up Convolutional Neural Networks Using Fine-tuned CP-Decomposition . *arXiv preprint arXiv:1412.6553*, 2014.

[23] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning Filters for Efficient ConvNets. *arXiv preprint arXiv:1608.08710*, 2016.

[24] Zhenhua Liu, Jizheng Xu, Xiulian Peng, and Ruiqin Xiong. Frequency-domain dynamic pruning for convolutional neural networks. *Advances in neural information processing systems*, 31, 2018.

[25] Lin Ning, Hui Guan, and Xipeng Shen. Adaptive Deep Reuse: Accelerating CNN training on the fly. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pages 1538–1549. IEEE, 2019.

[26] Lin Ning and Xipeng Shen. Deep Reuse: streamline CNN inference on the fly via coarse-grained computation reuse. In *Proceedings of the ACM International Conference on Supercomputing*, pages 438–448, 2019.

[27] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks. In *European conference on computer vision*, pages 525–542. Springer, 2016.

[28] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[29] Aladin Virmaux and Kevin Scaman. Lipschitz regularity of deep neural networks: analysis and efficient estimation. *Advances in Neural Information Processing Systems*, 31, 2018.

[30] Peisong Wang and Jian Cheng. Accelerating convolutional neural networks for mobile applications. In *Proceedings of the 24th ACM international conference on Multimedia*, pages 541–545, 2016.

[31] Fanyi Xiao and Yong Jae Lee. Video object detection with an aligned spatial-temporal memory. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 485–501, 2018.

[32] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.

[33] Xiangyu Zhang, Jianhua Zou, Kaiming He, and Jian Sun. Accelerating very deep convolutional networks for classification and detection. *IEEE transactions on pattern analysis and machine intelligence*, 38(10):1943–1955, 2015.

[34] Aojun Zhou, Anbang Yao, Yiwen Guo, Lin Xu, and Yurong Chen. Incremental Network Quantization: Towards Lossless CNNs with Low-Precision Weights. *arXiv preprint arXiv:1702.03044*, 2017.

[35] Hao Zhou, Jose M Alvarez, and Fatih Porikli. Less is more: Towards compact CNNs. In *European conference on computer vision*, pages 662–677. Springer, 2016.

[36] Michael Zhu and Suyog Gupta. To prune, or not to prune: exploring the efficacy of pruning for model compression. *arXiv preprint arXiv:1710.01878*, 2017.

[37] Xizhou Zhu, Yujie Wang, Jifeng Dai, Lu Yuan, and Yichen Wei. Flow-guided feature aggregation for video object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 408–417, 2017.

[38] Xizhou Zhu, Yuwen Xiong, Jifeng Dai, Lu Yuan, and Yichen Wei. Deep feature flow for video recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2349–2358, 2017.

[39] Will Zou, Shenghuo Zhu, Kai Yu, and Andrew Ng. Deep learning of invariant features via simulated fixations in video. *Advances in neural information processing systems*, 25, 2012.

A Proofs

Lemma 1. *Assume that Assumptions 1 and 2 hold, the iterations satisfy the following inequality for all $k \in \mathbb{N}$:*

$$\begin{aligned}\mathbb{E}_{\xi_k}[F(W_{k+1}, H_{k+1})] - F(W_k, H_k) \leq &-(\mu - \frac{1}{2}\alpha_k L M_G)\alpha_k \left[\|\nabla F_W(W_k, H_k)\|_2^2 + \|\nabla F_H(W_k, H_k)\|_2^2\right] \\ &+ \alpha_k^2 LM.\end{aligned} \tag{A.1}$$

Proof. Under Assumption 1 , for any $\{W, \widetilde{W}\} \in \mathbb{R}_1^d$ and $\{H, \widetilde{H}\} \in \mathbb{R}_2^d$, one obtains

$$\begin{aligned}F(W,H) - F(\widetilde{W},\widetilde{H}) &= \big(F(W,H) - F(\widetilde{W},H)\big) + \big(F(\widetilde{W},H) - F(\widetilde{W},\widetilde{H})\big) \\ &= \int_0^1 \frac{\partial F\big(\widetilde{W} + t(W-\widetilde{W}), H\big)}{\partial t} dt + \int_0^1 \frac{\partial F\big(\widetilde{W}, \widetilde{H} + t(H-\widetilde{H})\big)}{\partial t} dt \\ &= \int_0^1 \nabla F_W\big(\widetilde{W} + t(W-\widetilde{W}), H\big)^T (W-\widetilde{W}) dt + \int_0^1 \nabla F_H\big(\widetilde{W}, \widetilde{H} + t(H-\widetilde{H})\big)^T (H-\widetilde{H}) dt \\ &= \nabla F_W(\widetilde{W},H)^T(W-\widetilde{W}) + \int_0^1 \left[\nabla F_W(\widetilde{W} + t(W-\widetilde{W}), H) - \nabla F_W(\widetilde{W},H)\right]^T (W-\widetilde{W}) dt \\ &\quad + \nabla F_H(\widetilde{W},\widetilde{H})^T(H-\widetilde{H}) + \int_0^1 \left[\nabla F_H(\widetilde{W}, \widetilde{H} + t(H-\widetilde{H})) - \nabla F_H(\widetilde{W},\widetilde{H})\right]^T (H-\widetilde{H}) dt \\ &\leq \nabla F_W(\widetilde{W},H)^T(W-\widetilde{W}) + \int_0^1 L\|t(W-\widetilde{W})\|_2 \|W-\widetilde{W}\|_2 dt \\ &\quad + \nabla F_H(\widetilde{W},\widetilde{H})^T(H-\widetilde{H}) + \int_0^1 L\|t(H-\widetilde{H})\|_2 \|H-\widetilde{H}\|_2 dt \\ &\leq \nabla F_W(\widetilde{W},H)^T(W-\widetilde{W}) + \nabla F_H(\widetilde{W},\widetilde{H})^T(H-\widetilde{H}) + \frac{1}{2}L\big(\|W-\widetilde{W})\|_2^2 + \|H-\widetilde{H}\|_2^2\big).\end{aligned}$$

Therefore, the iterations generated by stochastic gradient algorithm satisfy

$$\begin{aligned}F(W_{k+1}, H_{k+1}) - F(W_k, H_k) &\leq \nabla F_W(W_k, H_{k+1})^T (W_{k+1} - W_k) + \frac{1}{2}L\|W_{k+1} - W_k\|_2^2 \\ &\quad + \nabla F_H(W_k, H_k)^T (H_{k+1} - H_k) + \frac{1}{2}L\|H_{k+1} - H_k\|_2^2 \\ &\leq -\alpha_k \nabla F_W(W_k, H_{k+1})^T g(W_k, H_k; \xi_k) + \frac{1}{2}\alpha_k^2 L\|g(W_k, H_k; \xi_k)\|_2^2 \\ &\quad - \alpha_k \nabla F_H(W_k, H_k)^T q(W_k, H_k; \xi_k) + \frac{1}{2}\alpha_k^2 L\|q(W_k, H_k; \xi_k)\|_2^2.\end{aligned}$$

Taking expectations in these inequalities with respect to the distribution of ξ_k, and noting that (W_{k+1}, H_{k+1})—but not (W_k, H_k)—depends on ξ_k, we obtain

$$\begin{aligned}\mathbb{E}_{\xi_k}[F(W_{k+1}, H_{k+1})] - F(W_k, H_k) &\leq -\alpha_k \nabla F_W(W_k, H_{k+1})^T \mathbb{E}_{\xi_k}[g(W_k, H_k; \xi_k)] \\ &\quad - \alpha_k \nabla F_H(W_k, H_k)^T \mathbb{E}_{\xi_k}[q(W_k, H_k; \xi_k)] \\ &\quad + \frac{1}{2}\alpha_k^2 L \mathbb{E}_{\xi_k}\left[\|g(W_k, H_k; \xi_k)\|_2^2\right] + \frac{1}{2}\alpha_k^2 L \mathbb{E}_{\xi_k}\left[\|q(W_k, H_k; \xi_k)\|_2^2\right]\end{aligned}$$

Combine Assumption 2 with Definition 4.6, we have the second moment of $g(W_k, H_k; \xi_k)$ and $q(W_k, H_k; \xi_k)$ satisfy

$$\begin{aligned}&\mathbb{E}_{\xi_k}\left[\|g(W_k, H_k; \xi_k)\|_2^2\right] \leq M + M_G\|\nabla F_W(W_k, H_k)\|_2^2, \\ &\mathbb{E}_{\xi_k}\left[\|q(W_k, H_k; \xi_k)\|_2^2\right] \leq M + M_G\|\nabla F_H(W_k, H_k)\|_2^2, \quad \textit{with } M_G := M_V + \mu_G^2 \geq \mu^2 > 0.\end{aligned}$$

Therefore we have

$$\begin{aligned}\mathbb{E}_{\xi_k}\big[F(W_{k+1}, H_{k+1})\big] - F(W_k, H_k) &\leq -\mu\alpha_k \left[\|\nabla F_W(W_k, H_k)\|_2^2 + \|\nabla F_H(W_k, H_k)\|_2^2\right] \\ &\quad + \frac{1}{2}\alpha_k^2 L\Big[M + M_G\big(\|\nabla F_W(W_k, H_k)\|_2^2 + \|\nabla F_H(W_k, H_k)\|_2^2\big)\Big] \\ &\leq -(\mu - \frac{1}{2}\alpha_k L M_G)\alpha_k \left[\|\nabla F_W(W_k, H_k)\|_2^2 + \|\nabla F_H(W_k, H_k)\|_2^2\right] \\ &\quad + \alpha_k^2 LM.\end{aligned}$$

□

Proof of Theorem 1

Proof. Taking the total expectation of Eq. A.1 and from the condition of $0 < \bar{\alpha} \leq \frac{\mu}{LM_G}$,

$$\begin{aligned}\mathbb{E}[F(W_{k+1}, H_{k+1})] - \mathbb{E}[F(W_k, H_k)] &\leq -(\mu - \frac{1}{2}\bar{\alpha}LM_G)\bar{\alpha}\mathbb{E}\big[\|\nabla F(W_k, H_k)\|_2^2 + \|\nabla F_H(W_k, H_k)\|_2^2\big] \\ &\quad + \bar{\alpha}^2 LM \\ &\leq -\frac{1}{2}\mu\bar{\alpha}\mathbb{E}\big[\|\nabla F_W(W_k, H_k)\|_2^2 + \|\nabla F_H(W_k, H_k)\|_2^2\big] + \bar{\alpha}^2 LM.\end{aligned}$$

Summing both sides of this inequality for $k \in \{1, ..., K\}$ and recalling Assumption 2 (a) gives

$$\begin{aligned}F_{inf} - F(W_1, H_1) &\leq \mathbb{E}[F(W_{K+1}, H_{K+1})] - F(W_1) \\ &\leq -\frac{1}{2}\mu\bar{\alpha}\sum_{k=1}^{K}\mathbb{E}\big[\|\nabla F_W(W_k, H_k)\|_2^2 + \|\nabla F_H(W_k, H_k)\|_2^2\big] + K\bar{\alpha}^2 LM.\end{aligned}$$

Rearranging above inequality and dividing further by K yields the result. □

Proof of Theorem 2

Proof. The second condition in Eq. 4.10 ensures that $\lim_{k\to\infty} \alpha_k = 0$, meaning that without loss of generality, we may assume that $\alpha_k LM_G \leq \mu$ for all $k \in \mathbb{N}$. Taking the total expectation of Eq. A.1, we obtain

$$\begin{aligned}\mathbb{E}[F(W_{k+1}, H_{k+1})] - \mathbb{E}[F(W_k, H_k)] &\leq -(\mu - \frac{1}{2}\bar{\alpha}LM_G)\bar{\alpha}\mathbb{E}\big[\|\nabla F(W_k, H_k)\|_2^2 + \|\nabla F_H(W_k, H_k)\|_2^2\big] \\ &\leq -\frac{1}{2}\mu\alpha_k\mathbb{E}\big[\|\nabla F_W(W_k, H_k)\|_2^2 + \|\nabla F_H(W_k, H_k)\|_2^2\big] + \alpha_k^2 LM.\end{aligned}$$

Summing both sides of this inequality for $k \in \{1, ..., K\}$ and recalling Assumption 2(a) gives

$$\begin{aligned}F_{inf} - F(W_1, H_1) &\leq \mathbb{E}[F(W_{K+1}, H_{K+1})] - F(W_1, H_1) \\ &\leq -\frac{1}{2}\mu\sum_{k=1}^{K}\alpha_k\mathbb{E}\big[\|\nabla F_W(W_k, H_k)\|_2^2 + \|\nabla F_H(W_k, H_k)\|_2^2\big] + LM\sum_{k=1}^{K}\alpha_k^2.\end{aligned}$$

Rearranging the above inequality and dividing further by $\mu/2$, we obtain

$$\sum_{k=1}^{K}\alpha_k\mathbb{E}\big[\|\nabla F(W_k, H_k)\|_2^2 + \|\nabla F_H(W_k, H_k)\|_2^2\big] \leq \frac{2(F(W_1, H_1) - F_{inf})}{\mu} + \frac{2LM}{\mu}\sum_{k=1}^{K}\alpha_k^2.$$

Then, we get the desired result by dividing A_k on the both sides. □

B Sufficient Direction Constant

Assumption 2(b) states that, in expectation, the vectors $g(W_k, H_k; \xi_k)$ and $q(W_k, H_k; \xi_k)$ are sufficient descent directions for F from W_k and H_k with norms comparable to the norms of the gradients. It guarantees that the model moves towards the descending direction of the loss function. Following the experimental setup in Section 5.1, we demonstrate that the proposed method empirically satisfies Assumption 2(b), and visualize in Figure 7 the sufficient direction constant μ for the (partial) convolutional layers of the four models during the end-to-end training using TREC. For SqueezeNet and ResNet-34, we show one block as the representative, since the other blocks have similar performance.

Several insights can be drawn from Figure 7. (i) The value of μ of each convolutional layer is consistently greater than zero, indicating that Assumption 2(b) is satisfied, further ensuring the convergence of the TREC-equipped CNNs. (ii) The lower convolutional layers have smaller μ compared to the upper ones. (iii) The value of μ gradually increases through iterations. In fact, the closer μ is to 1, the more the model moves toward the sufficient direction. Thus the gap between μ and 1 reflects the difference between the current descent direction of the model and the steepest descent direction [14]. The smaller values of μ in the lower convolutional layers in the early epochs indicate a larger difference in the descent direction. It is because the guidance obtained from the loss in the lower convolutinoal layers comes from back-propagation, which accumulates the disparities. Small μ values in the early epochs help the model avoid being trapped in local minimums, while large μ values in the later epochs help the model converge.

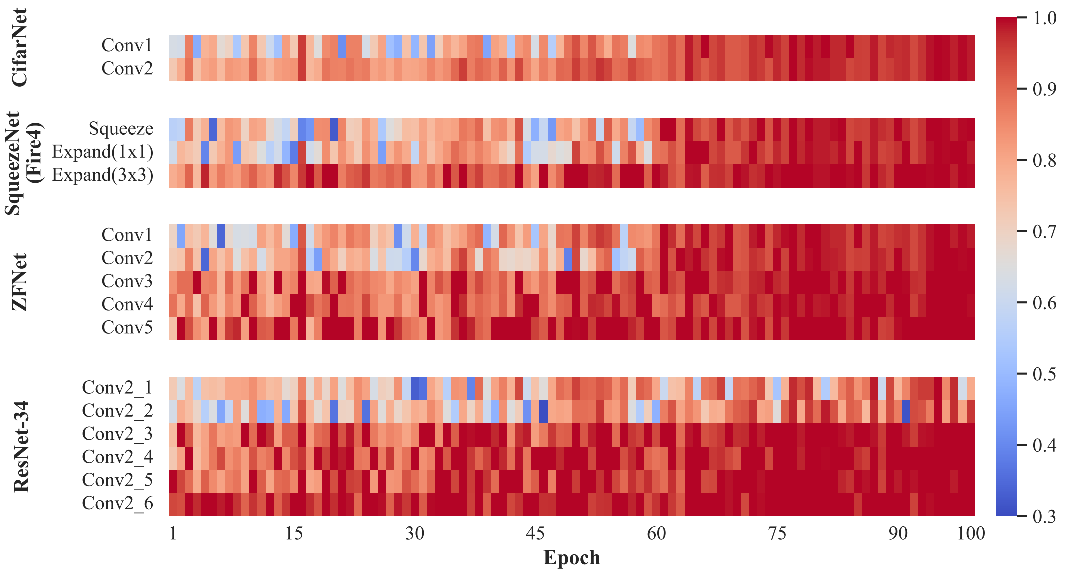

Figure 7: Sufficient direction constant μ.

C Single-Layer Performance

C.1 Single-Layer Performance Benefits

Table 3: Single-layer performance benefits. $Conf.$ means configuration, L is the width of sub-matrices, H is the number of hash functions, r_t represents the redundancy ratio, and Δ Acc is the accuracy difference.

Network	Layer	Conf.		r_t	Speedup	ΔAcc. (vs. Conventional)	ΔAcc. (vs. Deep Reuse)
		L	H				
CifarNet	Conv1	5	15	0.9429	1.81×	-0.0132	0.0428
	Conv2	10	10	0.7947	1.52×	-0.0077	0.0316
Average				0.8688	1.66×	-0.0105	0.0372
ZfNet	Conv1	147	5	0.9997	1.22×	-0.0011	0.0457
	Conv2	300	5	0.9967	4.69×	-0.0037	0.0319
	Conv3	432	5	0.9884	4.72×	-0.0076	0.0389
	Conv4	432	5	0.9982	6.23×	-0.0105	0.0228
	Conv5	288	5	0.9842	5.58×	-0.0068	0.0311
Average				0.9934	4.49×	-0.0059	0.0341
Vanilla SqueezeNet	Conv1	9	5	0.9969	1.27×	-0.0066	0.0311
	Fire2 - squeeze	96	4	0.9922	1.02×	-0.0179	0.0310
	Fire2 - 1×1 expand	8	5	0.9934	4.61×	0.0006	0.0233
	Fire2 - 3×3 expand	48	5	0.9875	6.04×	0.0040	0.0182
	Fire3 - squeeze	64	4	0.9877	1.30×	0.0029	0.0138
	Fire3 - 1×1 expand	8	5	0.9932	4.51×	0.0055	0.0132
	Fire3 - 3×3 expand	72	5	0.9876	6.43×	0.0029	0.0138
	Fire4 - squeeze	64	4	0.9980	1.07×	-0.0062	0.0337
	Fire4 - 1×1 expand	16	5	0.9879	3.61×	0.0037	0.0240

Network	Layer	Conf.		r_t	Speedup	ΔAcc. (vs. Conventional)	ΔAcc. (vs. Deep Reuse)
		L	H				
Vanilla SqueezeNet	Fire4 - 3×3 expand	144	5	0.9877	5.86×	0.0035	0.0289
	Fire5 - squeeze	128	4	0.9844	1.05×	0.0001	0.0139
	Fire5 - 1×1 expand	4	5	0.9906	4.96×	-0.0009	0.0151
	Fire5 - 3×3 expand	144	5	0.9500	3.25×	-0.0064	0.0162
	Fire6 - squeeze	32	5	0.9598	1.78×	-0.0008	0.1935
	Fire6 - 1×1 expand	6	5	0.9736	2.84×	0.0064	0.0213
	Fire6 - 3×3 expand	54	5	0.9504	16.06×	-0.0078	0.0159
	Fire7 - squeeze	48	5	0.9600	1.39×	-0.0023	0.0146
	Fire7 - 1×1 expand	6	5	0.9754	2.67×	0.0049	0.0224
	Fire7 - 3×3 expand	216	5	0.9523	16.95×	-0.0052	0.0105
	Fire8 - squeeze	8	5	0.9784	1.39×	-0.0063	0.0805
	Fire8 - 1×1 expand	4	5	0.9710	4.31×	0.0065	0.0224
	Fire8 - 3×3 expand	288	5	0.9500	18.66×	-0.0042	0.0127
	Fire9 - squeeze	256	5	0.9250	1.07×	0.0062	0.2365
	Fire9 - 1×1 expand	8	5	0.8563	1.10×	0.0037	0.0201
	Fire9 - 3×3 expand	288	5	0.8156	12.63×	0.0058	0.0148
	Conv10	4	5	0.9235	1.32×	-0.0084	0.1901
Average				0.9626	4.89×	-0.0006	0.0435
SqueezeNet + Complex Bypass	Conv1	9	5	0.9969	1.27×	0.0122	0.0783
	Fire2 - squeeze	96	4	0.9926	1.07×	0.0144	0.0559
	Fire2 - 1×1 expand	8	5	0.9914	1.41×	0.0023	0.0149
	Fire2 - 3×3 expand	48	5	0.9897	5.90×	0.0002	0.0156
	Bypass1 - 1×1	32	5	0.9940	2.45×	0.0071	0.0232
	Fire3 - squeeze	64	4	0.9877	1.07×	-0.0051	0.0145
	Fire3 - 1×1 expand	8	5	0.9918	2.30×	-0.0033	0.0167
	Fire3 - 3×3 expand	24	5	0.9876	5.85×	-0.0013	0.0222
	Fire4 - squeeze	64	5	0.9895	1.00×	-0.0040	0.0162
	Fire4 - 1×1 expand	16	5	0.9881	1.45×	-0.0037	0.0171
	Fire4 - 3×3 expand	48	5	0.9876	6.36×	0.0012	0.0254
	Bypass2 - 1×1	4	8	0.9933	2.53×	-0.0012	0.0277
	Fire5 - squeeze	64	4	0.9824	1.84×	-0.0036	0.0188
	Fire5 - 1×1 expand	16	5	0.9531	4.51×	0.0016	0.0163
	Fire5 - 3×3 expand	48	5	0.9542	11.83×	0.0001	0.0197
	Fire6 - squeeze	128	5	0.9719	1.01×	-0.0018	0.0213
	Fire6 - 1×1 expand	32	5	0.9631	4.40×	-0.0071	0.0189
	Fire6 - 3×3 expand	72	5	0.9534	12.50×	0.0002	0.0242
	Bypass3 - 1×1	32	5	0.9768	5.29×	0.0021	0.0192
	Fire7 - squeeze	48	5	0.9648	1.71×	0.0003	0.0257
	Fire7 - 1×1 expand	24	5	0.9570	4.00×	-0.0046	0.0265
	Fire7 - 3×3 expand	48	5	0.9531	9.36×	-0.0025	0.0357
	Fire8 - squeeze	64	4	0.9685	1.03×	-0.0003	0.0299
	Fire8 - 1×1 expand	16	5	0.9645	2.15×	0.0012	0.0164
	Fire8 - 3×3 expand	64	5	0.9550	7.57×	-0.0022	0.0188
	Bypass4 - 1×1	128	5	0.9609	1.12×	-0.0045	0.0385
	Fire9 - squeeze	128	4	0.8938	1.01×	-0.0078	0.0138
	Fire9 - 1×1 expand	16	5	0.8391	1.53×	-0.0019	0.0188
	Fire9 - 3×3 expand	96	5	0.8406	11.64×	-0.0022	0.0235
	Conv10	4	5	0.9438	1.29×	0.0057	0.0326
Average				0.9629	3.88×	-0.0003	0.0249
ResNet (ImageNet-64×64)	Conv1	25	5	0.8396	7.64×	-0.00063	0.0236
	Conv2-1	144	5	0.9942	7.69×	-0.0010	0.0353
	Conv2-2	144	5	0.9876	7.99×	-0.0002	0.0255
	Conv2-3	144	5	0.9919	7.80×	-0.0027	0.0455
	Conv2-4	144	5	0.9880	7.98×	0.0008	0.0301
	Conv2-5	144	5	0.9884	7.96×	-0.0036	0.0201
	Conv2-6	144	5	0.9876	7.99×	-0.0027	0.0475

Network	Layer	Conf. L	Conf. H	r_t	Speedup	ΔAcc. (vs. Conventional)	ΔAcc. (vs. Deep Reuse)
ResNet (ImageNet-64×64)	Conv3-1	144	5	0.9510	6.75×	-0.0030	0.0250
	Conv3-2	144	5	0.9579	7.18×	-0.0044	0.0169
	Conv3-3	144	5	0.9500	6.87×	-0.0007	0.0302
	Conv3-4	144	5	0.9537	7.02×	-0.0045	0.0190
	Conv3-5	144	5	0.9528	6.98×	-0.0008	0.0398
	Conv3-6	144	5	0.9554	7.09×	-0.0023	0.0229
	Conv3-7	144	5	0.9557	7.10×	0.0028	0.0262
	Conv3-8	144	5	0.995	7.49×	-0.0011	0.0195
	Conv4-1	288	5	0.9815	1.88×	-0.0003	0.0386
	Conv4-2	72	5	0.9802	2.99×	-0.0036	0.0330
	Conv4-3	72	5	0.9804	3.00×	-0.0022	0.0168
	Conv4-4	72	5	0.9802	2.99×	-0.0002	0.0515
	Conv4-5	72	5	0.9800	2.99×	-0.0015	0.0275
	Conv4-6	72	5	0.9802	2.99×	-0.0009	0.0607
	Conv4-7	72	5	0.9812	3.01×	-0.0002	0.0311
	Conv4-8	72	5	0.9800	2.99×	0.0001	0.0376
	Conv4-9	72	5	0.9803	3.00×	0.0001	0.0196
	Conv4-10	72	5	0.9804	3.00×	-0.0010	0.0427
	Conv4-11	72	5	0.9838	3.06×	-0.0009	0.0500
	Conv4-12	72	5	0.9800	2.99×	-0.0013	0.0271
	Conv5-1	36	5	0.9279	1.54×	-0.0023	0.0506
	Conv5-2	114	5	0.9239	1.20×	0.0005	0.0327
	Conv5-3	96	5	0.973	1.01×	-0.0011	0.0242
	Conv5-4	96	5	0.925	1.01×	-0.0023	0.0441
	Conv5-5	96	5	0.93	1.04×	-0.0007	0.0180
	Conv5-6	96	5	0.9268	1.01×	-0.0011	0.0420
Average				0.9630	4.64×	-0.0013	0.0326

C.2 Sensitivity Study: Performance on Varying Batch Sizes

In this section, we perform an analysis of the impact of batch size on TREC inference performance. Due to the stringent memory limit of MCU, it is infeasible to run the inferences on larger batch sizes. We hence use servers for this experiment and focus on the amount of avoided redundancy. We visualize in Figure 8 the impact of batch size on the average redundancy ratio (r_t), the larger the more redundancy is avoided. It can be seen that as batch sizes increase, the redundancy ratios also increase. It is intuitive: A larger batch increases the number of neuron vectors in the input matrix, and hence increases the possibility for reusing the results. This suggests that TREC can bring in more computational savings at larger batch sizes.

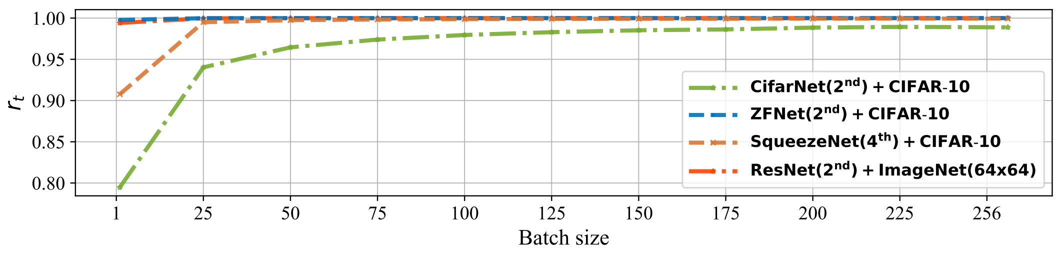

Figure 8: The impact of batch size on remaining ratio(r_t).