This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TETC.2021.3068063, IEEE Transactions on Emerging Topics in Computing

Convolutional Autoencoder-Based Transfer Learning for Multi-Task Image Inferences

Jie Lu, *Student Member, IEEE*, Naveen Verma, *Member, IEEE*, and Niraj K. Jha, *Fellow, IEEE*

Abstract—Pattern-recognition algorithms from machine learning play a prominent role in embedded sensing systems to derive inferences from sensor data. Very often, such systems face severe energy constraints, especially when dealing with high-dimensional data, such as images. The focus of this study is on reducing computational energy by exploiting the concept of transfer learning and energy-efficient dataflow accelerators. We show that the use of convolutional autoencoders can enable various opportunities to reduce computational energy and avoid significant reduction in inference performance when multiple task categories are targeted for inference. We validate our approach through a multi-task case study. The study targets a set of pictures with each picture containing four different task categories: gender, smile, glasses, and pose. In order to minimize inference time and computational energy, a convolutional autoencoder is used for learning a generalized representation of the images. Three scenarios are analyzed: transferring layers using convolutional autoencoders, transferring layers using convolutional neural networks trained on different tasks, and no layer transfer. We show that when the convolutional layers with one fully-connected layer are transferred using convolutional autoencoders, we can achieve a reduction of $6.58\times$ in computational energy, while improving performance by $1.98\%, 1.88\%, 4.11\%$, and 1.47% for gender, smile, glasses, and pose inferences, respectively, as compared to the no-transfer method, when the number of training samples is small.

Index Terms—Convolutional neural networks; Energy reduction; Machine learning, Multi-task images; Transfer learning.

1 Introduction

Data mining and machine learning technologies have achieved significant success, thanks to advances in both the algorithms and the hardware implementations. These successes have enabled the implementation of energy-efficient systems with improved accuracy [1]. One of the recent efforts has been to move toward multi-task learning [2], [3]. Data, such as images and sounds, carry rich information that can be used for various classification tasks. For example, an image alone can be used for object detection, object categorization or scene recognition. At a finer level, each object can be further recognized or categorized depending on the details of the object. However, in order to achieve high inference accuracy, these methods often require a large amount of manually labeled training data for each performed task. Such labeled data might be difficult to obtain and, further, developing and training a separate model for every new task is often prohibitive (in terms of energy and application constraints). Thus, we focus our attention on energy-efficient inference for multi-task learning.

While a complete semantic understanding leading to human-level performance (in terms of processing speed and accuracy) on answering any arbitrary question based on a set of data remains an extremely difficult task, researchers have taken up a range of approaches to begin to address this problem. In particular, they have identified sub-tasks that are useful for working towards overall efficient and complete learning. These approaches range from transfer learning, which uses previously developed models and/or representations so that relatively fewer samples are required when training for new tasks, to reinforcement learning that allows quick learning of new tasks from existing ones, to blind source separation, which aims to separate a set of mixture signals without any prior knowledge. [3], [4], [5], [6], [7].

The need for transfer learning arises when data easily become outdated or a new task needs to be learned. Without transfer learning, the original features used for one task may not be suitable for a new task. The original features can lead to poor inference accuracy when used for a new inference task. For task-specific feature learning, large quantities of labeled samples are often necessary for high inference performance. However, a large number of labeled samples may not be available. Thus, the inference performance suffers. In order to ease the training of new tasks with the same type of data, various transfer-learning techniques that help with information preservation and transfer have been proposed [2].

In this work, we focus on quickly retraining a machine-learning model for a new task with the few labeled samples that are available, at very low computational energy. In particular, we target frequently focused-on image-related machine-learning tasks. Convolutional neural networks (CNNs) are one of the most highly used and best-performing machine-learning models for image-related tasks [3], [4], [5], [8]. However, they can be very energy-intensive due to the large number of weights, activations, and computations involved [9]. In order to achieve high inference accuracy for new tasks at low computational energy, we propose the synergistic use of three concepts, spanning algorithms and systems:

- **Generalized feature learning.** Autoencoders are an unsupervised learning technique that utilizes neural

This work was supported by NSF under Grant No. CNS-1617640. *The authors are with the Department of Electrical Engineering, Princeton University, Princeton, NJ, 08544 USA e-mail: {jielu, nverma, jha}@princeton.edu.*

2168-6750 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Princeton University. Downloaded on March 25,2021 at 12:29:31 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TETC.2021.3068063, IEEE Transactions on Emerging Topics in Computing

networks for feature or representation learning [10]. Autoencoders can be used to learn an encoding for a set of data by transforming the data into a latent space and back into the original input space [10]. The unsupervised learning aspect of autoencoders means that the features do not lose information that may be important for a new set of tasks but are irrelevant to the existing task.

- **Low-bit (quantized) neural networks.** By operating on low-bit weights and activations, a network is able to work with low-bit kernels to reduce the computations required during a real-time inference operation [11]. The exact number of bits used for weights and activations of the network can vary based on the accuracy requirements.
- **Fixed low-energy transfer layer.** Since the feature extraction layers of the autoencoder are universal to a particular type of input, they can be fixed for transfer learning. This suggests that with a low-bit neural network, we can implement fixed layers in specialized architectures to improve energy savings.

We take advantage of information transfer, whereby the performance of learning, carried out with a small fraction of labeled samples in the target category, can be improved with the availability of a large number of unlabeled samples. In addition, taking into account the fact that the transfer layers may be common to all learning tasks, one can efficiently execute the transferred layer with highly specialized hardware. It should be noted here that we use an autoencoder for information transfer rather than models built with data labeled for a given task. Such models belong to the source task category [2]. However, such pre-existing models that are built using labeled data (source task category) may be limited when labeled data are scarce.

The objectives of this work are as follows.

1) Use of a low-bit convolutional autoencoder to extract generalized features, thereby reducing computational energy per operation while maintaining high inference performance for the target category task with fewer labeled samples.
2) Implementation of the feature extraction algorithm using specialized hardware to further reduce the computational energy while performing the inference task.
3) Implementation of a weight-stationary and output-stationary dataflow, by taking advantage of the fixed low-bit parameters trained by the autoencoders to reduce inference energy.

The rest of this article is organized as follows. Section 2 presents a general overview of the topic along with the definitions of some notations used in this work. Section 3 provides a brief survey of transfer learning and neural network binarization, and motivation behind the algorithm described in this article. Section 4 gives the details of the proposed approach. Section 5 discusses the application of the developed technique to a multi-task face dataset and presents the results. Section 6 draws conclusions from the presented work and discusses future work.

2 Background

In this section, we give a brief overview of transfer learning using autoencoders and energy-efficient implementations of such autoencoders.

2.1 Transfer Learning

Transfer learning is a machine-learning method in which the model developed for one task is reused or partially reused as the starting point for a model constructed for a second task [3]. In particular, if the tasks are truly related, there is a shared structure between the related source and target categories that can be exploited by transferring knowledge [2]. Several popular approaches are based on this method [2], [3], [12], [13]. Transfer learning is utilized in many areas to improve inference performance. For example, in the domain of brain-computer interfaces, transfer learning using variational autoencoders has been considered for generalizing learning parameters across different subjects [12]. With a similar goal, but a different methodology, Du et al. have also proposed the extraction of underlaying common features across a set of varied action gestures under different modalities [7]. Other than attempting to discover underlying commonality in a dataset, efforts have also been extended to co-training related models. For example, Zhao et al. have attempted to combine and co-train convolutional neural networks (CNNs) with autoencoders to simultaneously complete the task of facial expression recognition and facial expression synthesis [13].

In computer vision, examples of transfer learning include trying to overcome an insufficient number of training samples for a target category by adapting classifiers trained for a source category [14]. Other approaches deal with the source and target domains from the same task categories but with a different data distribution [15], for example, data collected under the same conditions that differ due to lighting, background, or viewpoint variations. In contrast to these works, we use transfer representations that are trained on unlabeled images.

Similar to our work, Oquab et al. [3] report a technique in which a CNN structure is trained from the source domain and transfers the convolutional layers, called the transfer parameters, to a new target domain. The data are not only in different domains, but also belong to different task categories. The transferred layers remain fixed. Only the new fully-connected (FC) layers are trained on the target task. Unlike our approach, however, the transferred layers are trained based on a specific task and require the corresponding labels in the source domain. Huang et al. [5] and Kemker et al. [4] both propose the use of single or multiple convolutional autoencoders to transfer information from the source domain to a target domain. However, the former study [5] utilizes the transferred convolutional layers as a starting point for CNN design. Thus, fine-tuning or retraining of a CNN is done every time a new target task is presented. Another study [4] uses multiple convolutional autoencoders to concatenate a set of features for use in classification. Unlike the methods mentioned above, we pre-train the convolutional layers of the CNNs on a large-scale unsupervised task, where the inputs are mapped to a different space and reconstructed as the original inputs. The

2168-6750 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Princeton University. Downloaded on March 25,2021 at 12:29:31 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TETC.2021.3068063, IEEE Transactions on Emerging Topics in Computing

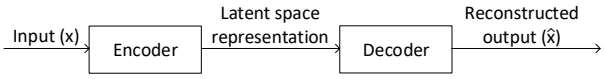

Fig. 1: A general autoencoder structure.

encoding convolutional layers and FC layers are then used for transferring knowledge without retraining.

2.2 Autoencoder

Artificial neural networks represent a popular way for generating autoencoders. An autoencoder is a system used for learning encodings of data that are efficient in some metric or characterized by other specific properties of interest (e.g., sparsity) [16]. Autoencoders have many interesting applications, such as data compression, visualization, and pre-training of neural networks [12], [16], [17], [18]. Instead of training the network to predict some target value y, given inputs x, an autoencoder is trained to reconstruct its own inputs (x). Therefore, the output vector has the same dimensionality as the input vector. The general structure of an autoencoder is shown in Fig. 1.

During its learning process, the autoencoder is optimized by minimizing the reconstruction error between the input (x) and resulting output ($\hat{x}$). The encoding part of the autoencoder can then be used as the learned generalized transferable layer. For samples with rich information, such as images, a stacked autoencoder consisting of multiple layers has been widely used [17]. The deep autoencoder was first proposed by Hinton et al. [19] and has been extensively studied since then [10], [18], [20]. This encoder consists of multiple encoding layers and forwards the code learned from one layer to the next, as done by FC neural networks [18]. A deep autoencoder is often trained using stochastic gradient descent (SGD) and has been widely studied for different applications. However, similar to the FC neural networks, a deep autoencoder does not perform well for image-related datasets.

Convolutional autoencoders, on the other hand, are suitable for image-related tasks. Each input or convolutional layer convolves a set of kernels with the input channels, generating the next convolutional layer with varying number of output channels [10]. The convolution operation has three main advantages [21]: 1) the weight-sharing mechanism in each convolutional layer reduces the number of parameters that need to be tuned; 2) local connectivity, where the same filter is used for all the positions of a given layer of the input, thereby learning correlations among neighboring pixels; 3) translational and rotational invariance (with respect to directional shifting in the image or rotation of the image) with respect to the location of the pixels of interest within the image. Owing to these advantages, the convolutional autoencoder is a better choice for learning a generalized representation for image-related data.

A convolutional autoencoder combines the local convolution connections with the autoencoder. The encoding layers of a convolutional autoencoder are similar to a CNN architecture, such as LeNet [22]. Its decoding layers are used to convert the encoded layer to outputs. The set of convolutional layers of the decoder is called the convolutional decoder.

The convolutional decoder carries out a transpose convolutional operation [21]. This operation attempts to reconstruct the inputs based on the encoder outputs as precisely as possible. Transpose convolution can be thought of as an operation that enables the recovery of the initial input size. The transpose of a non-padded convolution is equivalent to convolving a zero-padded input [21]. From the unsupervised greedy training of the standard autoencoder, the parameters of the encoding and decoding operation can be computed.

2.2.1 Low-bit autoencoder

Quantized neural networks have recently garnered significant interest. The use of neural networks for forming autoencoders makes it possible to readily apply quantization techniques to autoencoders. Previous efforts on implementing a quantized neural network can be divided into two categories: quantizing pre-trained, full-precision models (with or without retraining) [23] and training a quantized model from scratch [11], [24]. In this work, we focus on approaches that belong to the second category. The resultant quantized networks can be used for inference under resource constraints. For training quantized neural networks from scratch, many authors have suggested maintaining a high-precision copy of the weights when updating them while feeding quantized weights and activations to the forward and backward propagation passes, respectively [11], [25]. This approach yields good empirical performance. Another widely used solution employing only low-precision weights is stochastic rounding, which involves rounding with the additional consideration of probability of how to round [24]. DoReFa-Net is one such quantized network that explores reducing precision during the forward as well as backward pass. It allows setting of weights, activations, and gradients to their individual number of bits during training [11]. During the training step, the weights and gradients are maintained in floating-point during backward propagation. Quantization is applied during forward propagation. Quantized activations and weights enhance network energy efficiency while carrying out the inference task.

2.3 Hardware Implementation

Data movement and memory accesses are key considerations for neural network hardware energy, for they are often the most energy-intensive part [9], [26]. Many architectures have been proposed to fully explore efficiency of data movement and memory access. Among them is the spatial architecture based on a two-dimensional array of processing elements (PEs) [27]. Each PE is an arithmetic-and-logic unit (ALU) with its own control logic and local memory that can be a scratchpad or a register file (RF). Specifically, different types of dataflow are used based on the optimization target. No local reuse, weight-stationary, and output-stationary are the three commonly studied dataflow setups [28], [29], [30].

No local reuse takes advantage of inter-PE communication for input reuse and partial sum accumulation [28]. The PE array is divided into groups of PEs. Within a given

2168-6750 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Princeton University. Downloaded on March 25,2021 at 12:29:31 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TETC.2021.3068063, IEEE Transactions on Emerging Topics in Computing

group, the same input pixels are used with different weights from the same input channel. The partial sums across different PE groups are accumulated to produce the final output. For this dataflow, no RF storage is required since it is more reliant on inter-PE connections and ALU datapaths. This leaves a larger area for the global buffer.

For weight-stationary dataflow, each filter weight is stored in the RF within each PE [29]. The same set of weights is reused to maximize filter reuse. Each pixel from the same channel of a given input/convolutional layer is broadcasted to the same set of PEs sequentially, and the partial sums are accumulated. Thus, RF is used to store stationary weights, and if a large number of partial sums is produced at a time, these partial sums are temporarily stored in the global buffer.

For output-stationary dataflow, each output is stationary in a PE [30]. The partial sums are stored in the same RF for accumulation. RF is used for partial sum storage to achieve stationary accumulation. In addition, RF storage for input buffering is also needed to exploit convolutional reuse. At the system level, the inputs and weights are broadcasted from the global buffer to the PE array. Once completed, the outputs are streamed back to the global buffer.

By using convolutional autoencoders, we can keep the weights of the network constant. This means that for a set of inputs for new task categories, there would be no need to tune or change the weights of the fixed layers. Given the static and non-changing nature of the network weights, we are now able to further take advantage of a form of the weight-stationary and output-stationary dataflow. This dataflow is designed to minimize the computational energy for reading and writing partial sums/intermediate results. It keeps the partial sum accumulations for the same output activation value locally in an RF within each PE, with a size of few kilobytes or less. The multiple levels of memory hierarchy help improve the autoencoder energy efficiency by providing low-cost data access. For example, fetching data from the RF or neighboring PEs costs one to two orders of magnitude lower energy than fetching data from dynamic random access memory (DRAM).

2.4 Energy Estimation

The number of operations required by a neural network does not necessarily reflect the computational energy required by the model for carrying out an inference task [1]. This is because the major source of energy consumption is memory access for weights and inputs/outputs while fetching data. In fact, fetching data from the DRAM through the memory hierarchy to the computing elements can consume energy that is orders of magnitude higher than a multiply-accumulate (MAC) operation [31]. Therefore, we model two sources of computational energy in a CNN: computation and memory access [1], [27]. CNN MAC operations account for over 99% of the total operations [1] and thus constitute the bulk of computation energy. For memory access energy, the numbers can vary depending on memory level and where the data come from. Thus, it is crucial to ensure minimal access to data from higher energy-consuming memory structures, such as DRAM.

3 Related Work

Feature learning is an essential part of machine learning. It is based on extracting features from the raw data representation. Compared to conventional heuristic or manual approaches, data-driven feature learning via deep learning has exhibited a much higher performance [32]. Using deep learning, simple features are first extracted from raw data, following which more complex features are derived in subsequent layers [32]. Using multi-iteration learning, feature extraction parameters are continuously optimized via forward and backward propagation.

Feature learning is often classified into two categories: supervised and unsupervised [33]. In supervised learning, the input data have associated with them corresponding output labels and are forwarded from the input to the last layer for making a prediction. The architecture is trained by minimizing the value of the cost function that captures how different the target output value and the predicted output value are. Backward propagation is used to optimize the connection parameters, i.e., the weights, between each pair of layers. CNNs [22] are an example of how features can be learnt this way. CNNs are widely used for image analysis, speech recognition [34], text analysis, and many other tasks. In particular, in the field of image analysis, CNNs have had a great success in performing tasks such as face recognition [8], scene parsing [35], cell segmentation [36], neural circuit segmentation [37], and analysis of various medical images [38], [39].

In the unsupervised learning approaches, such as restricted Boltzmann machine (RBM) [40], deep belief network [19], autoencoders [41], and stacked autoencoders [42], unlabeled data are used to learn features, while a small amount of labeled data may be used for fine-tuning the parameters. For example, Kalleberg et al. propose a convolutional autoencoder approach to analyze breast images [43]. Chen et al. propose learning algorithms for medical image analysis using a convolutional autoencoder [38]. However, these methods require retraining from the input layer, through the convolutional layers and then the FC layers, for each inference task. This is different from the approach we propose here since the weights of all transferred layers need to stay the same for any inference task.

In this work, we propose a convolutional autoencoder unsupervised learning algorithm for general image processing. Compared to a conventional CNN [39], [44], the proposed scheme is an improvement in that the unsupervised autoencoder and CNN are collaboratively used to extract features from the image. For example, Whatmough et al. proposed a similar concept in fixing the first few layers of a CNN to function as a fixed feature extractor for image related tasks [44]. They proposed to first train a well-defined CNN architecture which has been trained on one image dataset. The first few layers of the CNN are then fixed and transferred to retrain the later layers for a new image dataset. In a similar manner, Zhao et al. [13] proposed a weight-transfer mechanism. Though the weights are not fixed, they are transferred and improved by correlating a current architecture with one that is designed for a new task [13]. Both have shown promising results in either improving the inference performance or the energy efficiency of the

2168-6750 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Princeton University. Downloaded on March 25,2021 at 12:29:31 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TETC.2021.3068063, IEEE Transactions on Emerging Topics in Computing

target tasks. However, due to the scarcity of image labels, this work uses a large amount of unlabeled data to train the feature-learning network, while only a small amount of labeled data is used to train the FC layers that are tailored to the task.

4 METHODOLOGY

In this work, we replace the convolutional layers of a CNN with low-bit convolutional encoding layers of an autoencoder. We then take advantage of the static kernel weights to further reduce the energy of the hardware implementation. The energy analysis method proposed by Sze et al. [27] is used for this purpose.

4.1 Convolutional Autoencoder Transfer Learning

The overall structure of the convolutional autoencoder layers with label-specific training and inference is illustrated in Fig. 2.

The autoencoder is first trained using non-labeled data, as can be seen from Fig. 2(b). The model parameters are updated based on how well the non-labeled data can be reconstructed. Once trained, the encoding part of the autoencoder, or autoencoder feature extraction, is kept constant and used for task-specific classifier training. During the classifier training phase, the autoencoder feature extraction is kept the same for all inference tasks, but the classifier is trained specifically for a given inference task. In the testing phase, autoencoder feature extraction remains constant and universal for all tasks whereas the classifier can be switched based on the presented inference task. The convolutional autoencoder (Fig. 2(a)) consists of an encoding part and a decoding part. The encoding part is a collection of convolutional layers plus an FC layer. The decoding part also contains an FC layer, followed by multiple deconvolutional layers, with sizes matching the corresponding convolutional layers.

Fig. 3 shows details of the convolutional autoencoder. The input/convolutional layer, $x \in R^{l_{IN} \times l_{IN} \times n}$, is either derived from the input images or from the outputs of the previous layer. Each convolutional layer contains n input channels. The convolutional autoencoder operation includes $n \times m$ convolutional kernels, where m is the number of output channels per layer. The outputs of a given layer is of size $l_{OUT} \times l_{OUT} \times m$. When inputs of a layer are derived from the outputs of the previous layer, n_{layer} is equivalent to the output channels, $m_{layer-1}$, from the previous layer (i.e., $n_{layer} = m_{layer-1}$). The size of the convolutional kernel is $d \times d$, where $d \leq l$. $\theta = \{W, \hat{W}, b, \hat{b}\}$ represents the set of parameters learned during training of the convolutional autoencoder, where $b \in R^m$ and $W = \{w_k, k = 1, 2, ..., m\}$ represents the biases and convolutional kernels of the encoding layers. w_k is of size $d \times d \times n$. $\hat{W} = \{\hat{w}_k, k = 1, 2, ..., m\}$ and $\hat{b}$ represent the parameters in the deconvolutional layers of the convolutional decoder, where $\hat{b} \in R^m, \hat{w}_k \in R^{d \times d \times n}$.

First, the input image x is convolved with the kernels, w_k, of the kth convolutional kernel for the output value, o_k, where $k = 1, 2, ..., m$. The computation is performed as follows:

$$o_k = f(x) = \sigma(w_k * x + b_k) \tag{1}$$

In Eq. 1, σ represents a nonlinear activation function. We use the rectified linear unit (ReLU):

$$ReLU(x) = max(0, x). \tag{2}$$

Since the encoded outputs o_k are fed to the decoding part of the autoencoder, the reconstructed input, $\hat{x}$ can be computed as:

$$\hat{x} = g(f(x)) = \sigma(\hat{w}_k * o_k + \hat{b}_k). \tag{3}$$

We use the L2 norm to compare the reconstructed input $\hat{x}$ with x. The resulting loss function used in optimization is described by Eq. 4 given below:

$$L(x, \hat{x}) = ||x - \hat{x}||_2^2. \tag{4}$$

The loss function is minimized using SGD [22] to optimize the convolutional autoencoder layer.

In order to assess the effectiveness of the convolutional autoencoder transfer, we also consider the exact convolutional autoencoder architecture, without implementing transfer, as the baseline. In addition, we consider a baseline transfer-learning paradigm where the CNN architecture is first trained on a source task. We then transfer the same layers as the convolutional autoencoder (i.e., four convolutional layers and an FC layer) and retrain the remaining FC layer for each target task specifically.

4.2 Hardware Setup

Many research studies focus on improving energy efficiency of CNN processing without sacrificing application accuracy or increasing hardware cost [9]. A typical CNN is composed of two layer types: convolutional and FC. The fundamental computation of both layer types is MAC operations that can be easily parallelized. We focus on spatial architectures in which the ALUs form a processing chain to pass data from one unit to another directly. Each ALU may have its own control logic and local memory, which can be a static random-access memory (SRAM) or RF. The corresponding block is referred to as a PE.

Specialized hardware accelerators based on PEs aim to jointly optimize computation and data access [27]. Specifically, with additional memory structures included within the PEs, many different types of dataflow (i.e., no local reuse, weight-stationary, or output-stationary) tailored to different CNN types can be implemented [27], [28], [29], [30]. No local reuse, for example, reads the same input pixels while changing the filter weights [27]. Weight-stationary reuses the same filter weights while input pixels are broadcasted to the PEs [29]. Output-stationary has each PE storing the same output pixel [30]. These types of architectures also have storage hierarchies to reduce memory accesses for weights and activations. Similarly, we consider a structure with four levels of storage hierarchy: DRAM, global buffer, inter-PE communication array, and RF, as shown in Fig. 4. The relative cost incurred by a MAC operation to access each storage level in a commercial 65-nm process is given in Table 1 [27]. The energy analysis consists of two parts: (1)

2168-6750 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Princeton University. Downloaded on March 25,2021 at 12:29:31 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TETC.2021.3068063, IEEE Transactions on Emerging Topics in Computing

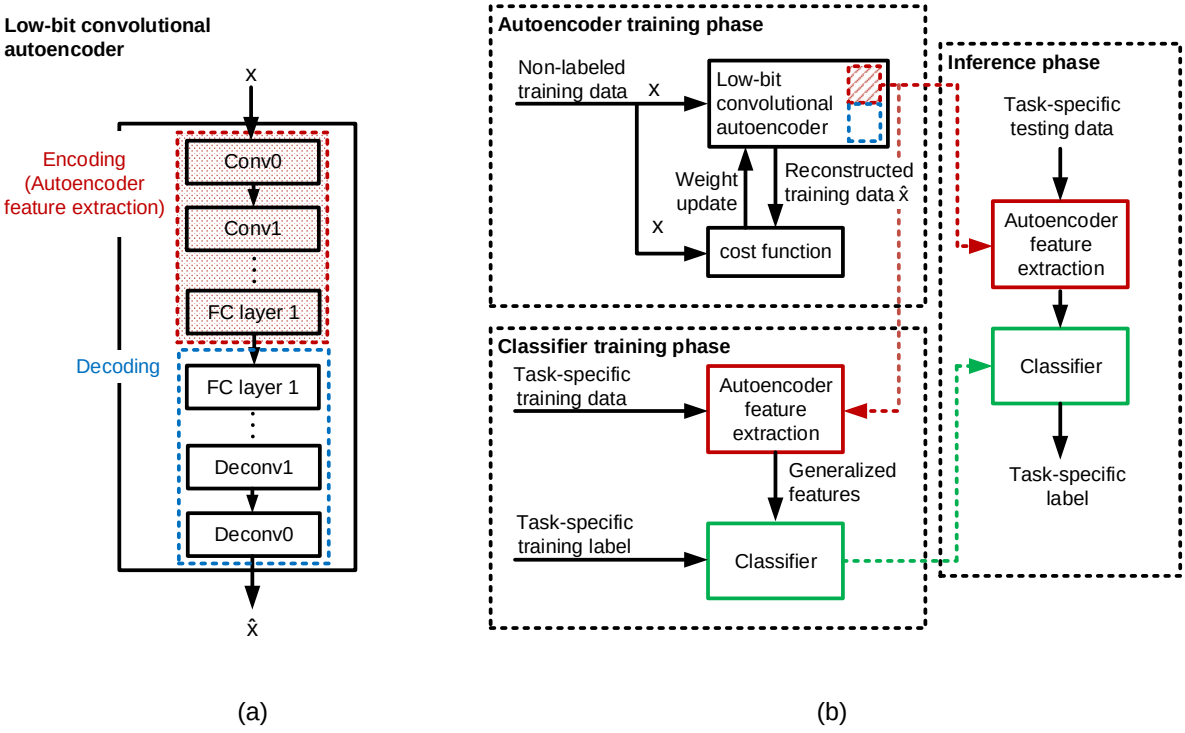

Fig. 2: Design flow of the convolutional autoencoder training phase, classifier training phase, and testing phase. (a) Low-bit convolutional autoencoder structure with multiple convolutional layers and one FC layer for the encoding layers, and an FC layer and multiple deconvolutional layers for the decoding layers. (b) Design flow of the proposed structure with the encoding layers of the autoencoder kept constant once trained and used for task-specific classifier training. Only the classifier is specifically trained for each task and changed during the inference phase depending on the desired inference task.

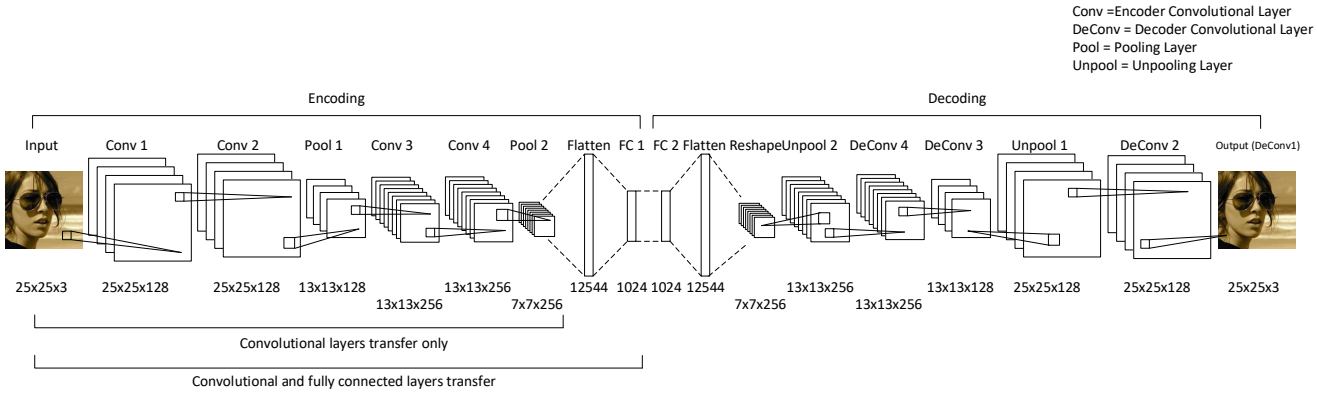

Fig. 3: Convolutional autoencoder with four convolutional layers and one FC layer for encoding and an FC layer and four convolutional layers for decoding.

2168-6750 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Princeton University. Downloaded on March 25,2021 at 12:29:31 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TETC.2021.3068063, IEEE Transactions on Emerging Topics in Computing

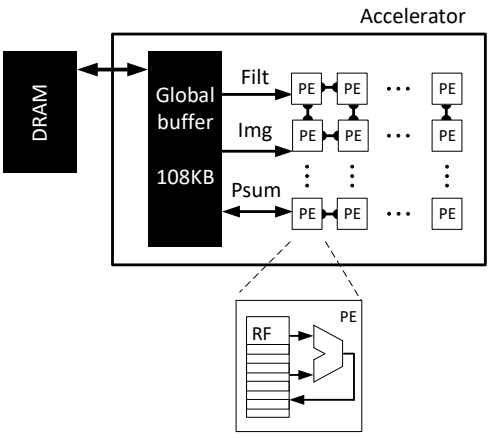

Fig. 4: Eyeriss memory levels with off-chip DRAM, global buffer, PE structures, and within-PE RF [27].

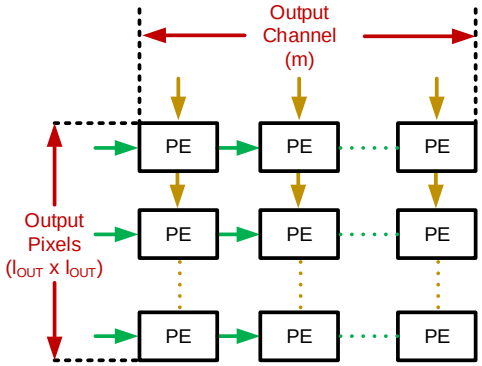

Fig. 5: Logical mapping of input feature maps and output feature maps onto PEs of size 13×12. A PE column produces a specific channel of outputs. Each PE row produces a specific output across all channels within a feature map.

input data access energy cost that includes the cost for filters and convolutional layer inputs, and (2) output accumulation energy cost. Energy costs are quantified by counting the number of accesses to each level of the storage hierarchy and weighting these accesses with the cost from Table 1.

TABLE 1: Normalized energy cost relative to a MAC operation from a commercial 65nm process [27].

	DRAM	Global Buffer (>100kB)	Array (inter-PE)	RF (0.5kB)
Norm. Energy	200 ×	6×	2×	1×

To implement 2-D convolution, convolutional layers with different dimensions can be physically mapped to a PE array that has a fixed physical dimension. Such a mapping takes advantage of reusing input data and kernels implemented on each PE. In a typical layer, the number of 2-D convolutions is equal to the product of the number of input channels (n), input size ($l \times l$), and the number of output channels (m). For our implementation, we use a physical PE array of size 13×12. Each PE row maps to a single output location across certain output channels and each PE column maps to a certain output channel. This is called logical mapping. The logical mapping of the PE is shown in Fig. 5. Fig. 6 shows the computations performed for an output element within an output channel on a PE. We use a combination of weight-stationary and output-stationary dataflow with this PE array. Since all the kernels are low-bit and fixed, they can be stored within a specialized memory structure within each PE. Therefore, each PE contains a unique set of kernels to be reused throughout the inference phase. Only the inputs need to be broadcasted from a higher storage level, such as the global buffer, to each PE.

Each PE within the array also has an input RF and a partial sum RF, each capable of storing 12 elements and 24 elements, respectively [27]. By taking advantage of these RFs, each PE can compute, in parallel, the output for a given i, j pixel (where $i, j = 1, 2, ..., l_{OUT}$) of up to 24 output channels. Therefore, each PE column can compute 13

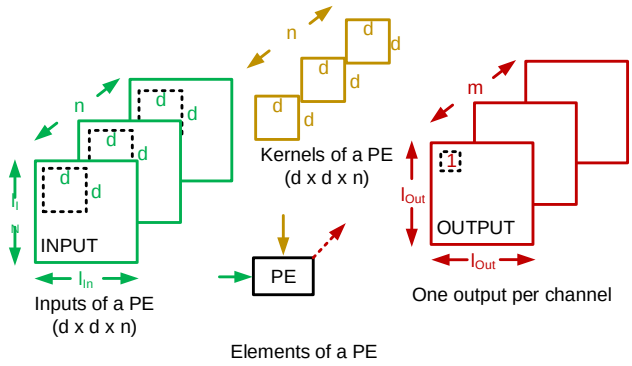

Fig. 6: PE job distribution. Each PE takes inputs of size $d \times d \times n$ and kernels of size $d \times d \times n$ to produce one output per channel.

separate output elements from up to 24 output channels at a time. Each PE row computes the same i, jth output across multiple output channels. For example, if the first PE row computes the outputs at position $i = 0, j = 0$, the second PE row computes the outputs at position $i = 0, j = 1$, and so on. Once these outputs are computed, the results are written into a global buffer that is used later as an input to the next convolutional or FC layer. All PE rows then start computations for the next set of i, j outputs. For example, the 13 PE rows first compute the $i = 0, j = 0, 1, 2, ..., 12$ outputs. After these computations are executed and the outputs are stored in a global buffer, these PEs then compute the $i = 0, j = 13, 14, ..., 24$ outputs.

Since PE rows have overlaps in inputs, due to the nature of the convolution, most PEs can have inputs arrive from other PEs instead of reading them directly from the global buffer, as shown in Fig. 7. Thus, instead of a global buffer read that would cost $6\times$ the energy, only a $2\times$ array move energy is incurred. In doing so, we exploit input data reuse and output accumulation. For all inputs, we need to consider the number of times each data value is read

2168-6750 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Princeton University. Downloaded on March 25,2021 at 12:29:31 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TETC.2021.3068063, IEEE Transactions on Emerging Topics in Computing

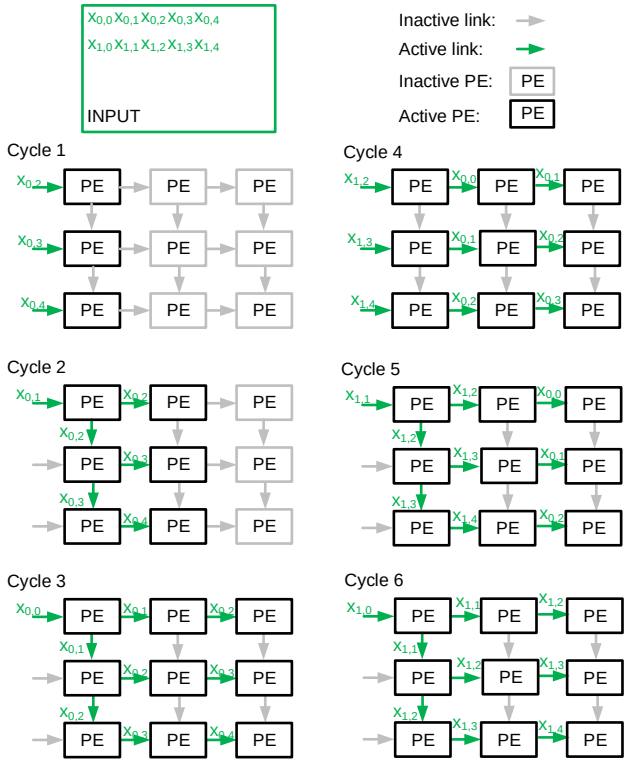

Fig. 7: Illustration of dataflow across cycles

from the highest-cost level to its lower-cost level during its lifetime. All data accesses, such as input, weight kernel, and output reads and writes, can occur at the DRAM, global buffer, array or RF level.

For each final output pixel, many partial outputs need to be computed and then summed. The output is accumulated within a local RF within each PE when each partial output is generated. Therefore, not all partial sums need to be stored upwards into the higher-cost memory levels until the computations are complete, thus saving the relevant amounts of energy. This energy saving comes from the saved number of accesses at higher memory levels. The number of accesses at each level is defined as the number of times each input data/weight/partial sum enters and exits its memory levels during its lifetime. The energy cost can then be estimated as:

$$\alpha \times dl(\text{DRAM}) + \beta \times dl(\text{global buffer}) + \gamma \times dl(\text{array}) + \delta \times dl(\text{RF}), \quad (5)$$

where $dl(*)$ represents the energy for each memory level access, as shown in Table 1, and α, β, γ, and δ represent the total number of accesses to the DRAM, global buffer, array, and RF, respectively.

In the architecture setup, there are 156 PEs, 36-element RF per PE, and 108 kB global buffer. The breakup for the 36-element RF is as follows: 12 for the inputs and 24 for the partial sum.

Our network takes an image of size 25×25 with three input channels (similar to that of Fig. 3). Thus, the input layer has $25 \times 25 \times 3$ input elements, as shown in Table 2. 3×3 convolutional kernel is used for all convolutional layers. The first and second convolutional layers have 128 channels while the third and fourth convolutional layers have 256 channels. A pooling layer is used after the second and the fourth convolutional layers only, reducing the output width and height by half. An FC layer then follows the fourth convolutional layer and leads to an output size of 1024. The second FC layer has an input size of 1024 and output size of 512, thus the weight dimension is 1024×512. The final output layer, depending on the task, has an output dimension of five or two. Thus, the last set of weights is of size 512×5 or 512×2. The total number of filtered weights is 13.74M.

TABLE 2: Kernel size, number of weights per PE, input size, and output size of each CNN layer. The autoencoder sizes refer to the fixed parameters trained by the convolutional autoencoder and total refers to all the parameters in the CNN.

Layer	Kernel Size	Weights /PE	Input Size	Output Size
CONV1	3456	288	1875	80000
CONV2	147456	12288	80000	80000
POOL1			80000	21632
CONV3	294912	24576	21632	43264
CONV4	589824	49152	43264	43264
POOL2			43264	12544
FC1	12.25M	82341	12544	1024
FC2	524288	3361	1024	512
Autoencoder Fixed	13.24M	168645		
total	13.74M	172006		

Of the 13.74M weights, the convolutional layers have 0.99M weights and the FC layers have 12.75M weights. 12.25M weights are involved in mapping the 256 convolved activation maps to the 1024 FC outputs. Therefore, it is important to also implement the first FC layer in the autoencoder so that a majority of the weights do not have to be retrained for a new task. We compare the obtained inference results when the first FC layer is included in the autoencoder versus when only the convolutional layers are included as transfer layers.

A CNN layer usually requires hundreds to thousands of processing passes to complete its processing, with opportunities for input reuse and partial sum accumulation existing across all these passes. The global buffer is used to exploit these opportunities by buffering two types of data: inputs and outputs. The inputs stored in the global buffer can be reused across multiple PEs. The outputs that are accumulated across passes use the global buffer as an intermediate storage and thus do not get stored in the DRAM until the final output values are obtained.

Given the static and low-bit nature of the weights, we can implement lookup tables (LUTs) to store the weights for each PE. Each PE needs to use 168645 weights for the

2168-6750 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Princeton University. Downloaded on March 25,2021 at 12:29:31 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TETC.2021.3068063, IEEE Transactions on Emerging Topics in Computing

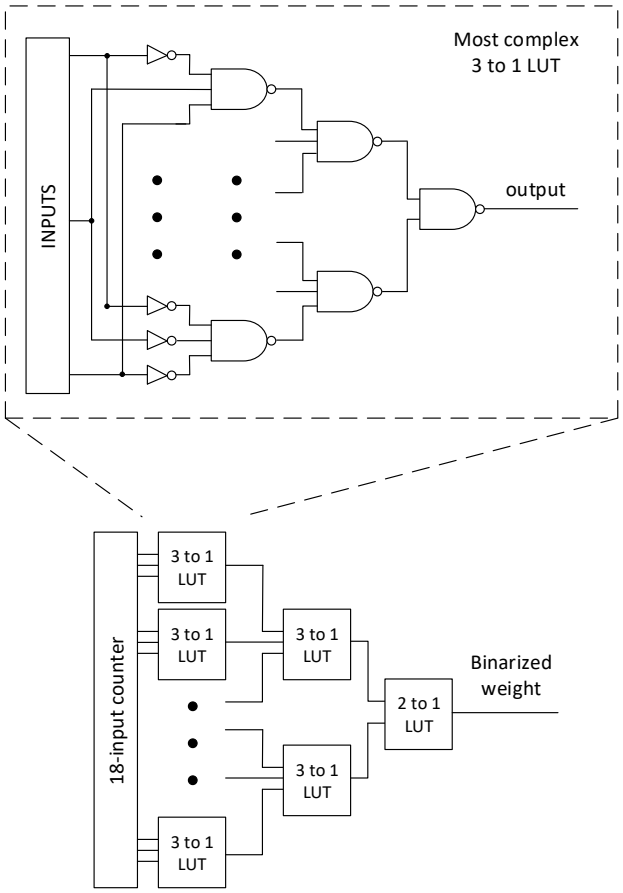

Fig. 8: 18-to-1 LUT for binarized weight storage and extraction within each PE. The critical path of the most complex 3-to-1 LUT has at most three gates, with an inverter on the input side.

CNN, as can be seen from Table 2. In other words, an 18-input LUT is needed to code all the weights within each PE, as shown in Fig. 8. In this LUT, the worst-case scenario leads to nine gate switches per weight output. For each 3-to-1 LUT read, the critical path has three gates (Fig. 8). For a 2-to-1 LUT read, the critical path has two gates. Taking into account the input inverters, the critical path thus contains nine gates. From our NanoSim simulations, we find that each LUT read uses the same order of magnitude energy as a MAC operation. Therefore, we assume that LUT read energy is the same as MAC energy.

We use the Multi-Task Facial Landmark (MTFL) dataset [45] to evaluate our methodology. It contains 12995 face images, annotated with attributes of gender, smiling, wearing glasses, and head pose. Photographs in the dataset may have different sizes. For processing purposes, we resize all photographs to the same size: $25\times25\times3$. Data are divided into 10000 training samples and 2995 testing samples. The photographs in the training samples are further split into two groups randomly. The first group contains 7500 photographs that are used for autoencoder training. The remaining 2500 photographs are used for class-specific inference training.

For baseline transfer learning, we first use one class (e.g., gender) to train a CNN. Once trained, the target convolutional layers and the FC layer of the CNN are kept fixed and only the last connected layer of the CNN is retrained for the specific class (i.e., smiling, wearing glasses or head pose). We also evaluate the performance for the case when only the convolutional layers are transferred and the last two FC layers are retrained. In all, 7500 samples are used to train the CNN with the fixed convolutional layers and the remaining 2500 samples are used to train the FC layers for each task. The same set of training and testing samples are used for all task categories for ease of comparison.

5 Experiments

To evaluate the proposed method, the same 7500 samples are used to train an autoencoder that contains the same number and structure of convolutional and FC layers as the above-mentioned CNN (Fig. 3). The convolutional autoencoder is trained with the objective of minimizing the root-mean-square error of the reconstructed input, as shown in Eq. 4.

The inference accuracy is evaluated for the four attributes: gender, smiling, wearing glasses, and head pose. Gender contains male and female classes. Smiling contains the smiling and not smiling classes. Wearing glasses contains the wearing and not wearing glasses classes. Head pose contains five classes, each defining the degree at which the person is facing the camera: -60°, -30°, 0°, 30°, and 60°. The resulting performance is shown in Tables 3 and 4, and the energy requirements for each inference task are shown in Tables 5 and 6. The result comparison of transferring with convolutional autoencoder, transferring with source task, and no transfer, are discussed in the next section. Transferring with convolutional autoencoder performs the best overall. We also compare the energy results of all models. Although transferring with convolutional autoencoder and transferring with source task utilize the same energy during inference time, they both require less energy than when there is no layer transfer.

5.1 Transfer Learning with Source Task vs. Convolutional Autoencoder

In Table 3, the performance for the four cases pertaining to the source-task and the convolutional-autoencoder methods, using 7500 samples for model learning and 2500 samples for task-specific inference training, is compared. Inference results based on the use of only 2500 samples for CNN training for the four cases, without any information transfer, are also included as baseline performance. For all the cases, we also examine the scenario when enough labeled cases within the target task are not available. Further, 500 training samples are used for task-specific inference training.

The proposed convolutional autoencoder method (referred to as CAE transfer) performs the best compared to the source-task transfer method (referred to as CNN transfer) and separately-trained task-specific models (referred to as no transfer). When transferring only the convolutional layers (i.e., in Table 3), at 2500 samples, CAE transfer, CNN transfer, and no transfer methods do not differ by much.

2168-6750 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Princeton University. Downloaded on March 25,2021 at 12:29:31 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TETC.2021.3068063, IEEE Transactions on Emerging Topics in Computing

TABLE 3: Inference performance of four different tasks with only convolutional layers transferred at 2500 training samples and 500 training samples, and with convolutional layer and one FC layer transferred at 2500 training samples and 500 training samples.

Task	Only convolutional layers				Convolutional layers and one FC layer				No Transfer	
	2500 samples		500 samples		2500 samples		500 samples		2500 samples	500 samples
	CAE transfer	CNN transfer	CAE transfer	CNN transfer	CAE transfer	CNN transfer	CAE transfer	CNN transfer		
Gender	74.3	73.61	69.47	68.32	74.04	68.94	70.14	65.87	73.05	68.16
Smile	62.30	57.23	58.54	56.67	62.80	57.38	58.54	54.89	61.80	56.66
Glasses	74.20	72.63	71.51	68.47	73.12	69.73	71.51	68.46	73.39	67.40
Pose	77.08	74.17	72.30	70.60	75.31	72.49	72.63	70.10	73.07	71.16

CAE performs on an average $2.56\% \pm 1.91\%$ better than the CNN transfer method and $1.64\% \pm 1.61\%$ better than the no transfer method (Table 3). At 500 samples, CAE transfer performs, on an average, $1.95\% \pm 0.88\%$ better compared to CNN transfer and $2.11\% \pm 1.37\%$ better than no transfer. When transferring convolutional and FC layers, with 2500 training samples, CAE transfer improves accuracy, on an average, by $4.19\% \pm 1.27\%$ and $0.99\% \pm 1.02\%$ compared to CNN transfer and no transfer, respectively. With 500 training samples, CAE transfer improves accuracy, on an average, by $3.37\% \pm 0.75\%$ and $2.36\% \pm 1.19\%$ compared to CNN transfer and no transfer, respectively. The improvement in performance is more prominent in transferring convolutional and FC layers, specifically, when comparing CAE transfer and CNN transfer.

Next, we evaluate the performance when transferring only convolutional layers versus when transferring convolutional layers and the first FC layer (shown in Table 4 for 2500 training samples and 500 training samples). We see that the performance in these two cases is similar across the different tasks. For CNN transfer, transferring only the convolutional layers does better by $2.28\% \pm 2.03\%$ and $1.18\% \pm 1.13\%$ compared to transferring convolutional layers and the first FC layer, with 2500 and 500 training samples, respectively. For the proposed CAE transfer method, transferring only the convolutional layers does better by $0.65\% \pm 0.99\%$ and is worse by $0.25\% \pm 0.32\%$ compared to that with an additional FC layer, with 2500 training samples and 500 training samples, respectively. Thus, we see that the proposed CAE transfer is beneficial for enhancing generalization when the FC layer is kept fixed, which is preferred for hardware implementations due to its large number of parameters.

TABLE 4: Transferring only convolutional layers vs. transferring convolutional layers and an FC layer performance differences for 2500 training samples 500 training samples.

Task	2500 samples		500 samples	
	CAE transfer	CNN transfer	CAE transfer	CNN transfer
Gender	0.26	4.67	-0.67	2.44
Smile	-0.5	-0.15	0	1.78
Glasses	1.08	2.90	0	0.01
Pose	1.77	1.69	-0.33	0.49

5.2 System Analysis

For energy analysis, we compare the energy of loading fixed kernels through LUTs in the proposed method (Table 5) versus loading task-specific kernels for each task (Table 6), requiring a read from the DRAM to the global buffer, then into the array and each PE RF file. The DRAM and global buffer are shared by layer-wise inputs and outputs. In this case, because the kernels are different for each task, with each additional task, a new set of kernels is required. Due to the uncertainty involved in the use of kernels, LUTs cannot be employed. Therefore, the computational energy required for cases that do not involve information transfer depends on the storage levels each kernel has to traverse for a given image inference task. Since each PE does not contain a LUT for cases that do not involve information transfer, a 224-element SRAM scratchpad is used, as proposed in Eyeriss [27]. Here, for ease of comparison, the energy for reading from this SRAM is assumed to be equivalent to one MAC energy.

Based on the details given in Tables 5 and 6, and the energy consumption numbers in Table 1, we compare the energy required for inference when no information transfer takes place (Table 6) versus when inference with information transfer takes place (Table 5). Since the same CNN architecture is used, the MAC energy is the same in both cases. Since, in terms of energy, an RF access is equivalent to a MAC, and also to a LUT read, we include LUT weight reads into the number of RF accesses in Table 5. Both the proposed autoencoder method and the method without information transfer require the same dataflow and processing sequence, thus they both have the same RF energy. However, the inter-PE energy for the no information transfer method is $17.98\times$ that of the proposed method. The extra energy for the no information transfer method comes from kernel transfers between PEs. The global buffer energy and DRAM energy for the no information transfer method are $35.60\times$ and $7683.66\times$ that of the proposed method, respectively. For the method without information transfer, the entire CNN model is read from the DRAM once and from the global buffer multiple times to complete all layer-wise outputs. Since the global buffer is not big enough to store the entire CNN model, parts of the CNN model need to be read in from

2168-6750 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Princeton University. Downloaded on March 25,2021 at 12:29:31 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TETC.2021.3068063, IEEE Transactions on Emerging Topics in Computing

TABLE 5: Relevant memory access and MAC energy for a CNN with a LUT in each PE. All energy consumption numbers are based on one image inference.

Layer	MAC Energy (#MACs)	Num. of RF Accesses	RF Energy (#MACs)	Num. of Inter-PE Movements	Inter-PE Energy (#MACs)	Num. of Global Buffer Accesses	Global Buffer Energy (#MACs)	Num. of DRAM Accesses	DRAM Energy (#MACs)	Total Mem. Energy/layer (#MACs)
CONV1	2.16×10^6	4.64×10^6	4.64×10^6	310241	620482	8366	50192	1875	375000	5.69×10^6
CONV2	9.22×10^7	2.05×10^8	2.05×10^8	1.32×10^7	2.65×10^7	356923	2.14×10^6	-	-	2.33×10^8
POOL1	-	-	-	-	-	160000	960000	-	-	960000
CONV3	4.98×10^7	1.11×10^8	1.11×10^8	3.58×10^6	7.16×10^6	96512	579072	-	-	1.18×10^8
CONV4	9.97×10^7	2.21×10^8	2.21×10^8	7.16×10^6	$1.43 \times^1 0^7$	193024	1.16×10^6	-	-	2.37×10^8
POOL2	-	-	-	-	-	86528	519168	-	-	519168
FC1	1.28×10^7	2.69×10^7	2.69×10^7	3.60×10^6	7.20×10^6	25088	150528	-	-	3.42×10^7
FC2	524288	1.10×10^6	1.10×10^6	293888	587776	2048	12288	-	-	1.70×10^6
total	2.57×10^8	5.69×10^8	5.69×10^8	2.82×10^7	5.64×10^7	928489	5.57×10^6	1875	375000	6.32×10^8

TABLE 6: Relevant memory access and MAC energy for a CNN with weights accessed from the memory hierarchy. All energy consumption numbers are based on one image inference.

Layer	MAC Energy (#MACs)	Num. of RF Accesses	RF Energy (#MACs)	Num. of Inter-PE Movements	Inter-PE Energy (#MACs)	Num. of Global Buffer Accesses	Global Buffer Energy (#MACs)	Num. of DRAM Accesses	DRAM Energy (#MACs)	Total Mem. Energy/layer (#MACs)
CONV1	2.16×10^6	4.64×10^6	4.64×10^6	2.30×10^6	4.61×10^6	174520	1.05×10^6	5331	1.07×10^6	1.14×10^7
CONV2	9.22×10^7	2.05×10^8	2.05×10^8	9.83×10^7	1.97×10^8	7.45×10^6	4.47×10^7	147456	2.95×10^7	4.75×10^8
POOL1	-	-	-	-	-	160000	960000	-	-	960000
CONV3	4.98×10^7	1.11×10^8	1.11×10^8	4.96×10^7	9.92×10^7	3.93×10^6	2.36×10^7	294912	5.90×10^7	2.92×10^8
CONV4	9.97×10^7	2.21×10^8	2.21×10^8	9.92×10^7	1.98×10^8	7.86×10^6	4.72×10^7	589824	1.18×10^8	5.85×10^8
POOL2	-	-	-	-	-	86528	519168	-	-	519168
FC1	1.28×10^7	2.69×10^7	2.69×10^7	3.60×10^6	7.20×10^6	1.29×10^7	7.72×10^7	1.28×10^7	2.57×10^9	2.68×10^9
FC2	524288	1.10×10^6	1.10×10^6	293888	587776	526336	3.16×10^6	524288	1.05×10^8	1.10×10^8
total	2.57×10^8	5.69×10^8	5.69×10^8	2.53×10^8	5.07×10^8	3.31×10^7	1.98×10^8	1.44×10^7	2.88×10^9	4.16×10^9

2168-6750 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Princeton University. Downloaded on March 25,2021 at 12:29:31 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TETC.2021.3068063, IEEE Transactions on Emerging Topics in Computing

the DRAM when needed. Our method obviates the need for these kernel reads and writes across the memory hierarchy. We are able to save $6.58\times$ energy overall.

6 Conclusion

In this article, we presented a method that uses a low-bit convolutional autoencoder to implement energy-efficient CNNs, targeting multiple criteria, while improving inference performance. We evaluated the method with a set of photographs that contains four labels per photograph. At 500 samples, our method achieved $70.14\% \pm 0.42\%$ accuracy as opposed to $65.87\% \pm 1.05\%$ accuracy achieved by transfer learning with the source task method for gender detection, $58.54\% \pm 0.34\%$ versus $54.89\% \pm 0.96\%$ for smile detection, $71.51\% \pm 1.25\%$ versus $68.48\% \pm 2.82\%$ for glasses detection, and $72.63\% \pm 0.44\%$ versus $70.10\% \pm 1.36\%$ for head-pose detection. Our method performed better than the no information transfer method by $1.98\%, 1.88\%, 4.11\%$, and 1.47% for gender, smile, glasses, and pose inferences, respectively. Overall energy is reduced by $6.58\times$ for the framework with a LUT implementation relative to when all reads are from the given storage hierarchy. In future work, we will consider the potential benefits of using a convolutional autoencoder based on residual network for training generalized feature extraction.

References

[1] T. J. Yang, Y. H. Chen, and V. Sze, "Designing energy-efficient convolutional neural networks using energy-aware pruning," in *Proc. IEEE Conf. Comput. Vis. Pattern Recogn.*, Apr. 2017, pp. 5687–5695.

[2] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Trans. Knowl. Data Eng.*, vol. 22, no. 10, pp. 1345–1359, Oct. 2010.

[3] M. Oquab, L. Bottou, I. Laptev, and J. Sivic, "Learning and transferring mid-level image representations using convolutional neural networks," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recogn.*, Jun. 2014, pp. 1717–1724.

[4] R. Kemker and C. Kanan, "Self-taught feature learning for hyperspectral image classification," *IEEE Trans. Geos. Remote Sens.*, vol. 55, no. 5, pp. 2693–2705, May 2017.

[5] Z. Huang, Z. Pan, and B. Lei, "Transfer learning with deep convolutional neural network for SAR target classification with limited labeled data," *Remote Sens.*, vol. 9, no. 9, p. 907, Aug. 2017.

[6] B. Du, W. Xiong, J. Wu, L. Zhang, L. Zhang, and D. Tao, "Stacked convolutional denoising auto-encoders for feature representation," *IEEE Trans. Cybernetics*, vol. 47, no. 4, pp. 1017–1027, Mar. 2016.

[7] B. Du, S. Wang, C. Xu, N. Wang, L. Zhang, and D. Tao, "Multi-task learning for blind source separation," *IEEE Trans. Image Process.*, vol. 27, no. 9, pp. 4219–4231, May 2018.

[8] Y. Sun, X. Wang, and X. Tang, "Deep learning face representation from predicting 10,000 classes," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recogn.*, Jun. 2014, pp. 1891–1898.

[9] V. Sze, Y. H. Chen, T. J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proc. IEEE*, vol. 105, no. 12, pp. 2295–2329, Mar. 2017.

[10] J. Masci, U. Meier, D. Cireşan, and J. Schmidhuber, "Stacked convolutional auto-encoders for hierarchical feature extraction," in *Ann. Report Internet Corp. for Assign. Names and Numbers (ICANN)*, Dec. 2011, pp. 52–59.

[11] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou, "DoReFa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients," *arXiv preprint arXiv:1606.06160*, pp. 1–13, Jun. 2016.

[12] O. Özdenizci, Y. Wang, T. Koike-Akino, and D. Erdoğmuş, "Transfer learning in brain-computer interfaces with adversarial variational autoencoders," in *Proc. Int. IEEE/EMBS Conf. Neural Engin.*, Mar. 2019, pp. 207–210.

[13] R. Zhao, T. Liu, J. Xiao, D. P. Lun, and K.-M. Lam, "Deep multi-task learning for facial expression recognition and synthesis based on selective feature sharing," *arXiv preprint arXiv:2007.04514*, pp. 1–8, Jul. 2020.

[14] T. Tommasi, F. Orabona, and B. Caputo, "Safety in numbers: Learning categories from few examples with multi model knowledge transfer," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recogn.*, Jun. 2010, pp. 3081–3088.

[15] K. Saenko, B. Kulis, M. Fritz, and T. Darrell, "Adapting visual category models to new domains," in *Proc. Eur. Conf. Comput. Vis.*, Sep. 2010, pp. 213–226.

[16] C. Y. Liou, W. C. Cheng, J. W. Liou, and D. R. Liou, "Autoencoder for words," *Neurocomput.*, vol. 139, pp. 84–96, Sep. 2014.

[17] S. Gao, Y. Zhang, K. Jia, J. Lu, and Y. Zhang, "Single sample face recognition via learning deep supervised autoencoders," *IEEE Trans. Inf. Forensics Security*, vol. 10, no. 10, pp. 2108–2118, Oct. 2015.

[18] A. Krizhevsky and G. E. Hinton, "Using very deep autoencoders for content-based image retrieval," in *Proc. Eur. Symp. Artif. Neural Netw.*, vol. 1, Jan. 2011, p. 2.

[19] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, Jul. 2006.

[20] J. Zhang, S. Shan, M. Kan, and X. Chen, "Coarse-to-fine auto-encoder networks (CFAN) for real-time face alignment," in *Proc. Eur. Conf. Comput. Vis.*, Sep. 2014, pp. 1–16.

[21] M. D. Zeiler, "Hierarchical convolutional deep learning in computer vision," Ph.D. dissertation, Department of Computer Science, New York University, Jan. 2014.

[22] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

[23] K. Hwang and W. Sung, "Fixed-point feedforward deep neural network design using weights +1, 0, and -1," in *Proc. IEEE Workshop Signal Process. Sys.*, Oct. 2014, pp. 1–6.

[24] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," in *Proc. Int. Conf. Mach. Learn.*, Feb. 2015, pp. 1737–1746.

[25] M. Courbariaux, Y. Bengio, and D. Jean-Pierre, "Binaryconnect: Training deep neural networks with binary weights during propagations," in *Proc. Neural Inf. Process. Syst.*, 2015, pp. 3123–3131.

[26] R. Hameed, W. Qadeer, M. Wachs, O. Azizi, A. Solomatnikov, B. C. Lee, S. Richardson, C. Kozyrakis, and M. Horowitz, "Understanding sources of inefficiency in general-purpose chips," in *Proc. 37th Annu. Int. Symp. Comput. Archit.*, Jun. 2010, pp. 37–47.

[27] Y. H. Chen, J. Emer, and V. Sze, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," *ACM SIGARCH Comput. Archit. News*, vol. 44, no. 3, pp. 367–379, Sep. 2016.

[28] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing FPGA-based accelerator design for deep convolutional neural networks," in *Proc. ACM/SIGDA Int.l Symp. FPGA*, Feb. 2015, pp. 161–170.

[29] S. Chakradhar, M. Sankaradas, V. Jakkula, and S. Cadambi, "A dynamically configurable coprocessor for convolutional neural networks," in *Proc. 37th Annu. Int. Symp. Comput. Archit.*, Jun. 2010, pp. 247–257.

[30] Z. Du, R. Fasthuber, T. Chen, P. Ienne, L. Li, T. Luo, X. Feng, Y. Chen, and O. Temam, "Shidiannao: Shifting vision processing closer to the sensor," in *Proc. 42nd Annu. Int. Symp. Comput. Archit.*, Jun. 2015, pp. 92–104.

[31] M. Horowitz, "Computing's energy problem (and what we can do about it)," in *Proc. IEEE Int. Solid-State Circuits Conf.*, Feb. 2014, pp. 10–14.

[32] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE Trans. Pattern Anal.*, vol. 35, no. 8, pp. 1798–1828, Jun. 2012.

[33] T. O. Ayodele, "Types of machine learning algorithms," in *New Advances in Machine Learning*. IntechOpen, Feb. 2010.

[34] M. Chen, P. Zhou, and G. Fortino, "Emotion communication system," *IEEE Access*, vol. 5, pp. 326–337, Dec. 2016.

[35] C. Farabet, C. Couprie, L. Najman, and Y. LeCun, "Learning hierarchical features for scene labeling," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 8, pp. 1915–1929, Oct. 2012.

[36] H. Su, Z. Yin, S. Huh, T. Kanade, and J. Zhu, "Interactive cell segmentation based on active and semi-supervised learning," *IEEE Trans. Med. Imag.*, vol. 35, no. 3, pp. 762–777, Oct. 2015.

2168-6750 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Princeton University. Downloaded on March 25,2021 at 12:29:31 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TETC.2021.3068063, IEEE Transactions on Emerging Topics in Computing

[37] D. Ciresan, A. Giusti, L. M. Gambardella, and J. Schmidhuber, "Deep neural networks segment neuronal membranes in electron microscopy images," in *Proc. Neural Inf. Process. Syst.*, 2012, pp. 2843–2851.
[38] M. Chen, X. Shi, Y. Zhang, D. Wu, and M. Guizani, "Deep features learning for medical image analysis with convolutional autoencoder neural network," *IEEE Trans. Big Data*, Jun. 2017, doi:10.1109/TBDATA.2017.2717439.
[39] W. Shen, M. Zhou, F. Yang, D. Yu, D. Dong, C. Yang, Y. Zang, and J. Tian, "Multi-crop convolutional neural networks for lung nodule malignancy suspiciousness classification," *Pattern Recogn.*, vol. 61, pp. 663–673, Jan. 2017.
[40] G. E. Hinton, "Training products of experts by minimizing contrastive divergence," *Neural Comput.*, vol. 14, no. 8, pp. 1771–1800, Aug. 2002.
[41] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, "Greedy layer-wise training of deep networks," in *Proc. Neural Inf. Process. Syst.*, Dec. 2007, pp. 153–160.
[42] P. Vincent, H. Larochelle, Y. Bengio, and P. A. Manzagol, "Extracting and composing robust features with denoising autoencoders," in *Proc. Int. Conf. Mach. Learn.*, Jul. 2008, pp. 1096–1103.
[43] M. Kallenberg, K. Petersen, M. Nielsen, A. Y. Ng, P. Diao, C. Igel, C. M. Vachon, K. Holland, R. R. Winkel, N. Karssemeijer, and M. Lillholdm, "Unsupervised deep learning applied to breast density segmentation and mammographic risk scoring," *IEEE Trans. Med. Imag.*, vol. 35, no. 5, pp. 1322–1331, Feb. 2016.
[44] P. N. Whatmough, C. Zhou, P. Hansen, S. K. Venkataramanaiah, J. S. Seo, and M. Mattina, "FixyNN: Efficient hardware for mobile computer vision via transfer learning," *arXiv preprint arXiv:1902.11128*, pp. 1–13, Feb. 2019.
[45] Z. Zhang, P. Luo, C. C. Loy, and X. Tang, "Facial landmark detection by deep multi-task learning," in *Proc. Eur. Conf. Comput. Vis.*, Sep. 2014, pp. 94–108.

Jie Lu received the BSc (Hons.) degree in life sciences from Queens University, Kingston, Canada, in 2010, and the MASc degree in biomedical engineering from the University of Toronto, Toronto, Canada, in 2013, and the MA degree in electrical engineering from Princeton University, Princeton, in 2015. Currently, she is working toward the PhD degree in electrical engineering at Princeton University. Her main area of research interest is energy-efficient platforms for sensing and computing. She has presented her work at the 2013 International BCI Meeting and published in Frontiers on Brain-computer Interfaces and IEEE Transactions on Computers. During her master's and undergraduate degrees, she received James F. Crothers Fellowship in Peripheral Nerve damage in 2012, Kimel Family Graduate Student Scholarship in 2012, and NSERC USRA in 2008. She is a student member of the IEEE.

Naveen Verma received the B.A.Sc. degree in Electrical and Computer Engineering from the UBC, Vancouver, Canada in 2003, and the M.S. and Ph.D. degrees in Electrical Engineering from MIT in 2005 and 2009, respectively. Since July 2009, he has been at Princeton University, where he is currently a Professor of Electrical Engineering. His research explores how systems for learning, inference, and action planning can be enhanced through new sensing and computing technologies. He has served as a Distinguished Lecturer of the IEEE Solid-State Circuits Society. He is a recipient or corecipient of the 2006 DAC/ISSCC Student Design Contest Award, 2008 ISSCC Jack Kilby Paper Award, 2012 Alfred Rheinstein Junior Faculty Award, 2013 NSF CAREER Award, 2013 Intel Early Career Award, 2013 VLSI Symp. Best Student Paper Award, 2014 AFOSR Young Investigator Award, 2015 IEEE Trans. CPMT Best Paper Award, and several awards for teaching excellence.

Niraj K. Jha received his B.Tech. degree in Electronics and Electrical Communication Engineering from Indian Institute of Technology, Kharagpur, India in 1981 and Ph.D. degree in Electrical Engineering from University of Illinois at Urbana-Champaign, IL in 1985. He has been a faculty member of the Department of Electrical Engineering, Princeton University, since 1987. He is a Fellow of IEEE and ACM, and was given the Distinguished Alumnus Award by I.I.T., Kharagpur. He has received the Princeton Graduate Mentoring Award.

He has served as the Editor-in-Chief of IEEE Transactions on VLSI Systems and an Associate Editor of several other journals. He has co-authored five widely used books. His research has won 20 best paper awards or nominations and 21 patents. His research interests include smart healthcare, cybersecurity, machine learning, and monolithic 3D IC design. He has given several keynote speeches in the areas of nanoelectronic design/test, smart healthcare, and cybersecurity.

2168-6750 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Princeton University. Downloaded on March 25,2021 at 12:29:31 UTC from IEEE Xplore. Restrictions apply.