QuiltNet: Efficient Deep Learning Inference on Multi-Chip Accelerators Using Model Partitioning

Jongho Park*, HyukJun Kwon*, Seowoo Kim*, Junyoung Lee*
Minho Ha§, Euicheol Lim§, Mohsen Imani†, Yeseong Kim*
DGIST* SK Hynix§ UC Irvine†
{psyractal1123, durwjsdnehd3523, dusk2box, lolcy3205, yeseongkim}@dgist.ac.kr*
{minho1.ha, euicheol.lim}@sk.com§, m.imani@uci.edu†

Abstract

We have seen many successful deployments of deep learning accelerator designs on different platforms and technologies, e.g., FPGA, ASIC, and Processing In-Memory platforms. However, the size of the deep learning models keeps increasing, making computations a burden on the accelerators. A naive approach to resolve this issue is to design larger accelerators; however, it is not scalable due to high resource requirements, e.g., power consumption and off-chip memory sizes. A promising solution is to utilize multiple accelerators and use them as needed, similar to conventional multiprocessing. For example, for smaller networks, we may use a single accelerator, while we may use multiple accelerators with proper network partitioning for larger networks. However, partitioning DNN models into multiple parts leads to large communication overheads due to inter-layer communications. In this paper, we propose a scalable solution to accelerate DNN models on multiple devices by devising a new model partitioning technique. Our technique transforms a DNN model into layer-wise partitioned models using an autoencoder. Since the autoencoder encodes a tensor output into a smaller dimension, we can split the neural network model into multiple pieces while significantly reducing the communication overhead to pipeline them. Our evaluation results conducted on state-of-the-art deep learning models show that the proposed technique significantly improves performance and energy efficiency. Our solution increases performance and energy efficiency by up to 30.5% and 28.4% with minimal accuracy loss as compared to running the same model on pipelined multi-block accelerators without the autoencoder.

Keywords

Deep learning acceleration, DNN partitioning, Scalable DNN acceleration

ACM Reference Format:
Jongho Park*, HyukJun Kwon*, Seowoo Kim*, Junyoung Lee* and Minho Ha§, Euicheol Lim§, Mohsen Imani†, Yeseong Kim*. 2022. QuiltNet: Efficient Deep Learning Inference on Multi-Chip Accelerators Using Model Partitioning. In *Proceedings of the 59th ACM/IEEE Design Automation Conference (DAC) (DAC '22), July 10–14, 2022, San Francisco, CA, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3489517.3530589

DAC '22, July 10–14, 2022, San Francisco, CA, USA

ACM ISBN 978-1-4503-9142-9/22/07…$15.00
https://doi.org/10.1145/3489517.3530589

1 Introduction

With the advance of various deep learning accelerators, deep learning technology, also known as Deep Neural Networks (DNN), has become an essential component of current computer systems. For example, recent mobile systems equip application-specific integrated chip (ASIC) processors, called neural processing units, that perform deep learning inference with trained models for improved performance. State-of-the-art GPUs also employ specialized computation units for DNN inference, e.g., NVIDIA TensorRT [1], to serve efficient inference procedures enhanced by several model optimizations. Many deep learning acceleration techniques have also been studied on FPGA platforms [2]. Some support inference with various standardized DNN models, e.g., TensorFlow, PyTorch, Caffe, and ONNX, which are usually trained on high-end GPUs and/or ASIC platforms such as Google TPU [3].

To provide a better quality of service, recent deep learning models employ more complex and deeper (i.e., an increased number of layers) model structures at the cost of computation and resource usage. This significantly affects the efficiency and capability of the deep learning accelerators. For example, ResNet, which is a state-of-the-art model structure utilized as a base form of many advanced learning models for image recognition, can be configured with more than a hundred layers, e.g., 151 layers having 11.3 billion FLOPs, to provide high quality. This trend inevitably leads to the degradation of the performance and energy efficiency of existing accelerators. Furthermore, many accelerators utilize a large-sized off-chip memory, which is the key technical concern in a system's design due to its high energy consumption and slow communication speed [4].

For these reasons, it is necessary to scale the accelerator designs to achieve desired performance. Scaling a single chip is not often straightforward due to the limited area and power requirements. An alternative solution is to utilize *multiple chips* to accelerate a single but complex DNN model. However, it poses another technical challenge – "how do we partition the deep learning computation workload into multiple pieces?" At heart, the DNN workload consists of different layer-wise computations, e.g., matrix multiplication, where a layer transfers tensors, i.e., multi-rank arrays, to its next layers as inputs and outputs. It implies that, when deploying multiple chips to process different parts of a DNN model, multiple chips essentially need to communicate tensors.

Unfortunately, the size of processed tensors is not small since modern deep learning models employ a large size of convolution channel depths and feature maps. In this case, transfer rates for communicating data across different chips would not keep up with the memory bandwidth and/or computation power. Furthermore, direct peer-to-peer communication between heterogeneous chips is not often feasible in practice. For example, memory data would need to be transferred through the host system where multiple

accelerators are equipped; to avoid such communication overhead, the system needs specialized interfaces such as NVLink [5].

A promising solution is reducing the amount of data transferred across accelerators to avoid the bottleneck in the interconnect between accelerators or between the accelerator and CPU. One possible approach to this end is to utilize existing compression techniques [6]. However, it needs to change the underlying hardware/software architecture to add specialized accelerator blocks for the compression algorithms.

In this paper, we propose a *learning-driven* technique that effectively partitions a DNN model with a reduced amount of communication costs to deploy them on multi-chip accelerators. The proposed technique, called QuiltNet, takes a state-of-the-art DNN model structure as the input and inserts new *data projection layers* between two adjacent layers of the input model, i.e., at the location desired to be partitioned. The data projection layers are divided into two parts, the encoder and decoder, which are responsible for mapping (compressing) the original tensor into a smaller dimension and reverting them to the original size, respectively. Our technique learns the encoder and decoder with a given training dataset so that it extracts the learning features with tensors whose size is smaller than the output of the previous layer. Once trained, we can split the model into different parts, one producing reduced tensor outputs and the other one receiving them to make them proceed to the rest of the model, which can be accelerated on different chips. We utilize the autoencoder [7] structure to build the data projection layers. Since the autoencoder is composed of either typical fully-connected layers or convolution layers, which the state-of-the-art accelerators already support, it does not need any changes in off-the-shelf hardware to utilize our proposed technique.

To summarize, our main contributions are as follows:

- We propose QuiltNet, which is a **deep learning model partitioning technique** to reduce the inter-block communication costs for multi-chip accelerators. Unlike the existing compression-based data reduction, it needs no hardware changes to accelerate the data encoding/decoding procedure since it is integrated as a part of learning process.
- We show **how to interpose the data projection layers**, which encode and decode the transferred tensors between layers. Our proposed technique automatically learns the way to extract small enough knowledge without losing generality for the target model.
- We also present cost-effective methods to decide **where to place the data projection layers** based on the performance modeling and accuracy-efficiency tradeoff profiling.
- Through a comprehensive study for state-of-the-art deep learning models, we show that the proposed model partitioning technique can significantly **reduce the inter-block communication overhead** by two or three orders of magnitude with very minimal accuracy loss.
- In the experimental results evaluated on Xilinx U200 FPGA, we show that the proposed method improves the performance and energy efficiency of the multi-chip accelerator by up to 30.5% and 28.4% with minimal accuracy loss.

2 Related Work

As the size of the model and processed data grow, the need for acceleration increases. One way to enable the acceleration is to parallelize the computation and data of the given models with partitioning. Krizhevsky introduced a *One Weird Trick* [8], which reached data parallelism by configuring convolution layers, and model parallelism by configuring fully connected layers. With these techniques, we can assign different workers to train each part of the partitioned model and data examples.

One Weird Trick inspired many earlier studies, which focus on how to partition tensors to minimize the communication costs. For example, Hypar [9] partitioned the tensors in the neural network model, and optimized them by searching for a partition that minimizes the cost. The work shown in [10] proposes a technique, called Tofu, which partitions a dataflow graph of operation tensors and uses a recursive search algorithm to reduce the communication cost. AccPar [11] is one of those studies, which can optimize tensor partitioning in various situations, enabling parallelism on heterogeneous accelerators. Other research tried to achieve parallelism by developing hardware devices dedicated to the deep learning. One of those approaches is Simba [12], a deep learning multi-chip inference accelerator. To maximize system throughput, the architecture is focused on optimizing work partitioning and data partitioning to minimize inter-chip communication.

The prior work focused on systematical techniques which efficiently partition a given deep learning model without any model changes. In contrast, our work is an orthogonal approach in that the proposed technique *tunes* a given model structure with instrumentation for the inter-layer communication cost. Our technique fundamentally minimizes the amount of data transfers required to partition, making multi-chip acceleration more effective when combined with the prior techniques.

3 QuiltNet Technique

3.1 QuiltNet Design

The proposed technique enables efficient DNN model partitioning by instrumenting learning-driven data projection layers into an original given model. The data projection layer encodes an intermediately-computed tensor into a smaller dimension and decodes them back, making it a suitable place to partition as it minimizes data communication between different accelerator blocks.

Figure 1 illustrates the overview of the proposed QuiltNet framework. The proposed framework takes a deep learning model, either trained or untrained, as an input. It then interposes data projection layers in a form of the autoencoder network [7, 13] at the partitioning location (❶). Since the data projection layers are untrained at the beginning of this stage, we need to learn their weights with a training dataset given as another input.

There are two different learning procedures depending on the training status of the original networks. If the networks are untrained, we simply train the entire model instrumented with the data projection layers by following the standard learning procedure (❷). However, in many practices, a high-quality model trained with a full-fledged dataset would be already available on the shelf, e.g., ResNet-156 trained with ImageNet 10K, where the model deployment targets application-specific data, similar to the transfer learning [14]. In this case, we need to train the interposed autoencoder part with a target dataset of a limited size, while utilizing knowledge given in the original model as much as possible. The second learning procedure handles this case with two steps (❸).

A typical technical concern for training the autoencoder is how to guarantee the generality of the trained networks, i.e., how to avoid the overfitting issue and make the model correctly behave with data patterns not usually shown in the training dataset. To address this challenge, we employ data augmentation techniques [15] (❹). The data augmentation engine creates diverse variants for

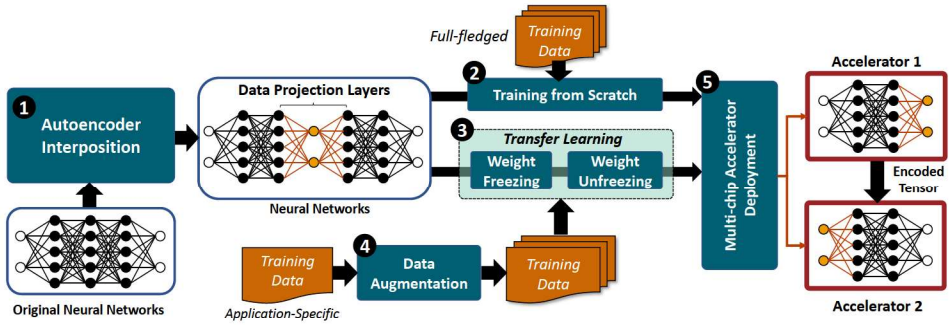


Figure 1: QuiltNet Overview

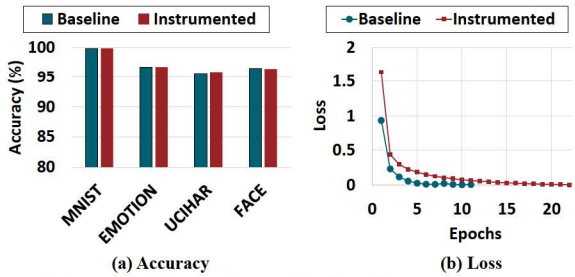


Figure 2: Comparison between the baseline and instrumented models

given input data, e.g., skewing, cropping, etc for image data, so that the autoencoder could learn more general patterns for future inputs. It would also potentially improve the prediction quality.

Once trained, we can split the model into two different parts for the multi-chip acceleration[1]. In this example (❺), the first part of the model ends with the encoding layer; the second part takes the encoded tensor $\mathbf{T_e}$ as the input of the decoding layer and proceeds with $\mathbf{T_d}$ to the next layer which originally exists in the given model. Since the two parts of the model only need to communicate the encoded tensor $\mathbf{T_e}$ which is smaller than the original tensor $\mathbf{T_p}$, it can reduce the communication overhead by the degree of $\frac{|\mathbf{T_p}|=|\mathbf{T_d}|}{|\mathbf{T_e}|}$. In the rest of the paper, we denote this reduction degree as R.

Feasibility of autoencoder interposition Since the data projection layers intentionally create an information bottleneck in the entire model, it would degrade the accuracy of the prediction model. To show the feasibility of the autoencoder interposition, we conduct a preliminary study on simple but practical datasets which could be learned with an appropriate quality only with fully-connected networks. In this study, we build a baseline model that has two hidden layers of 512 neurons. Between the hidden layers, we insert an autoencoder structure, which produces a *single floating-point value*, resulting in a very extreme reduction rate of 512×.

Figure 2a compares the prediction quality between the baseline model and instrumented model for four different datasets [16]. The results show that the autoencoder interposition can create negligible accuracy loss. On average, the instrumented model degrades the accuracy only by 0.05% as compared to the baseline. We also observed that the instrumented model may take larger training epochs to be sufficiently converged. Figure 2b shows how the categorical cross-entropy loss is changed over the training epochs for the MNIST dataset. The results present that, although the autoencoder interposition would incur longer training time, the loss eventually converges to the same level that the baseline achieves.

In fact, the feasibility of the information compaction has been observed in other earlier research. For example, the work in [17] showed a concept of hierarchical inference for various machine learning models in Internet of Things networks. In their work, each IoT device computes an aggregated representation of input data instead of sending the raw large data to the cloud for learning applications. Our experimental results for the autoencoder instrumentation agree with this finding, i.e., the tensors intermediately produced in the learning model can be hugely compacted without sacrificing quality; rather the autoencoder plays a role of regularization, potentially addressing overfitting issues. Motivated by this observation, we design the QuiltNet framework to enable multi-chip acceleration with significantly reduced communication overheads.

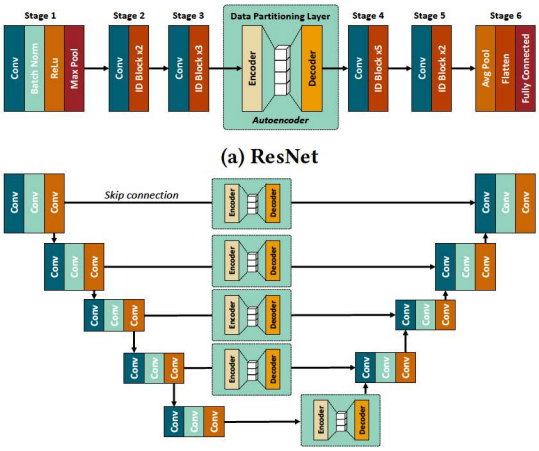
3.2 Autoencoder Interposition

The QuiltNet framework first instruments the data projection layers at the desired place for model partitioning. As mentioned in Section 3.1, the data projection layers project the original tensor into a smaller dimension using an autoencoder structure. A typical 1D autoencoder is composed of two dense (fully-connected) layers, one of each performing the encoding and decoding, respectively. An issue with this structure is that it could not maintain spatial information. Considering that the data projection layers would be inserted at any location, e.g., between convolution layers whose positional information are also important in the tensor, the fully-connected layer can not be always used.

For such cases, the proposed framework utilizes the 2D convolution autoencoder, which reduces the number of channels while using the same feature map size. It should be noted that some advanced autoencoder structures would utilize deeper layers more than two for higher quality; however, such modification does not provide better results in our evaluation. Thus, our data projection layers use one layer for each of the encoder and decoder. Please note that our proposed technique is completely learning-driven in that we compact the information only using the existing layers of typical deep learning computation; it does not need any changes in the well-established accelerator designs.

Figure 3a shows how we could instrument the state-of-the-art ResNet structure [18] with the 2D convolution autoencoder. The ResNet structure consists of multiple stages, each of which has different channel depths. We could insert the autoencoder at any place, e.g., at the end of a stage, to assimilate the tensor with a reduced channel depth. Some modern deep learning models may

[1]We can also apply multiple autoencoders to partition the model into multiple accelerators of more than two. We discuss this with detailed evaluation results in Section 4.5.



(a) ResNet

(b) UNet

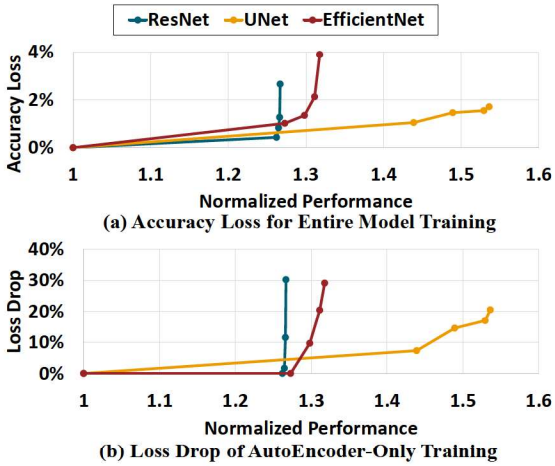
Figure 3: Autoencoder interposition

have complex skip connections, requiring the use of multiple autoencoders to completely partition the model into two pieces. As an example, Figure 3b shows the autoencoder interposition for the UNet structure [19], a fully convolutional network (FCN) that is widely used for image segmentation tasks. The UNet structure has multiple skip connections, which link the corresponding convolution layers between the contracting path and expansive path. In this case, we insert the autoencoder for each skip connection so that the model is completely divided into two pieces.

3.3 Transferring Knowledge from Pretrained Model

Since the data projection layers, i.e., the autoencoders, are integrated with the original neural network model, we can learn their weights with standard backpropagation process when the model needs to be trained from scratch. However, in practice, the model pretrained with a rich dataset is often already available while the model is fine-tuned through *transfer learning* with a specific dataset for the target application. When retraining the model with the specific dataset, the autoencoder part poses particular challenges since an untrained autoencoder initially produces random outputs, making it hard to utilize the trained weight values of the next layers. Furthermore, it is widely observed that continual learning on multiple tasks would abruptly lose the knowledge of the previously learned task when the tasks are sufficiently different. This phenomenon is known as *catastrophic forgetting*. In our case, the untrained autoencoder makes the weights of the rest of the layers meaningless, i.e., mistakenly considering the previous rich datasets are completely different from the target dataset.

We address the issue by devising a model retraining technique called *weight freezing*. The underlying reason for the catastrophic forgetting is that the autoencoder does not produce any useful information at the beginning of the learning. Thus, the weight freezing technique performs the training in two phases: (i) in the first phase, we only update the weights of the autoencoders while keeping (freezing) the weights of the other layers, i.e., the pretrained weights. This phase continues until the loss sufficiently converges so that the autoencoders have high enough quality. (ii) Then, during the second phase, we perform fine-tuning by retraining the weights of all layers. Since the autoencoders now produce meaningful information compatible with the pretrained weights, we can proceed the training while mitigating the catastrophic forgetting issue.



(a) Accuracy Loss for Entire Model Training

(b) Loss Drop of AutoEncoder-Only Training

Figure 4: Accuracy and loss relationship for normalized performance over different data reduction rates

Data augmentation A key challenge of training the autoencoder is how to make it insensitive enough to the inputs by avoiding overfitting the training data. We employ the data augmentation technique [15], which is widely used for training general-enough models; in our case, it also greatly helps overcome the overfitting issue of the autoencoder. The data augmentation technique creates variants of the training data samples, which enables the autoencoder to correctly handle diverse inputs. As a result, it improves the generality of the model including the data projection layers as well as the prediction accuracy based on the enriched dataset.

3.4 Interposition Decision

The QuiltNet efficiency is dependent on where/how to place the data projection layers in the given model. There exist two decision parameters: (i) the position of the data projection layers and (ii) the data reduction rate. The followings describe how our proposed framework identifies the two parameters.

Position: The optimal positions of data projection layers are where the split models take the same amount of time to execute their input tensor. To automate the decision process, we exploit a neural network performance modeling method shown in Paleo [20] to estimate the execution times of split models on each accelerator. The estimated execution time is the sum of the communication time and computation time. The communication time between accelerators is linear to the size of the transferred tensor for a given bandwidth, and the computation time is expressed with a first-order function to the number of flops. In our experimental setup, the bandwidth from host to FPGA is measured by 9.2GB/s, while the opposite is 11.7GB/s. For the computation, the model in practice may not be split with the same amount of execution time between each other. Thus, QuiltNet tries to identify the split models whose execution times are as similar as possible. That is, it finds the partition having the minimum value of $\sum_{i=1,i\neq i^*}^{k} |exec_{i^*} - exec_i|$ among all possible cases, where k is the number of partitions, $exec_i (= comp_i + comm_i)$ is the total execution time on i^{th} accelerator, and $exec_{i^*} = \max_{1\leq i\leq k} exec_i$. This method automatically identifies the layer to place the autoencoders, e.g., between stage 3 and 4 for ResNet50.

Data reduction rate: Once the position is determined, we identify the data reduction rate. An increase in the tensor reduction rate improves efficiency while reducing the model accuracy. Although

the decision is upon deployment requirements, the QuiltNet framework can guide the decision process by providing the trade-off relationships. As an example, Figure 4(a) shows the trade-off relationships between the accuracy loss and normalized performance for the three models over different data reduction rates. The result shows that there usually exists a *sweet spot* for the data reduction rate, which leads to high efficiency with negligible accuracy loss. For example, the accuracy loss is only 0.4% with R = 64 for ResNet50; but it significantly increases beyond this data reduction rate without having many performance improvements. Thus, we would select such rates to balance the accuracy loss and efficiency.

A potential issue is that computing the accurate trade-off relationship requires multiple entire model training trials, which result in high training costs to test various reduction rate candidates. We address this issue by training only the autoencoder parts and examining the loss changes. Let us assume that the model is split into two partitions, $f_{\theta^L}(\mathbf{x})$ and $f_{\theta^R}(\mathbf{x}')$, where $\mathbf{x}$ is the input tensor (an image) for the model and $\mathbf{x}'$ is the transferred tensor whose size is reduced by the data projection layers. In general, the goal of the autoencoder is to produce a transformed tensor similar to the given input tensor. In our case, as long as the data projection layers with a certain reduction rate are capable of creating $f_{\theta^L}(\mathbf{x}) \simeq \mathbf{x}'$, we can expect that it will effectively compress the transferred tensors.

Thus, we can only train the autoencoder with $f_{\theta^L}(\mathbf{x})$ to examine how it behaves over various reduction rates at a high level. Figure 4(b) shows that observing the loss drop of the *autoencoder-only* training is an appropriate, cost-effective proxy to examine the accuracy loss changes. It means that, even if we do not identify the accurate trade-off relationship shown in Figure 4(a), we can select the appropriate data reduction rate based on the results of Figure 4(b). In practice, training the autoencoder has much less training costs as compared to the entire model training. For example, we observe that the autoencoder-only training is 76.9× faster than the entire model training for ResNet50.

4 Evaluation

4.1 Experimental Setup

We implemented the QuiltNet software framework using PyTorch. The proposed QuiltNet could be deployed in various multi-chip environments; to evaluate the inference efficiency on real systems, we built a representative infrastructure by utilizing Xilinx Vitis platform, which runs the split deep learning models on Alveo U200 FPGA boards. The infrastructure runs the given model in a pipelined fashion, i.e., different boards run simultaneously to process tensors for different inputs sequentially given. We instrumented the Vitis library to profile the runtime and power consumption of the FPGA boards as well as the tensor sizes communicated. The host system runs on Intel Xeon 4215R processor with 32GB RDIMM PC4-21300R, connected with the FPGA board with PCIe 3.0. We apply the PartNet technique to three types of DNN models, ResNet50, UNet, and EfficientNet. The automated interposition decision process of QuiltNet inserted a data projection layer after the stage 3 for ResNet50, for UNet as described in Section 3.2, and after the stage 5 for EfficientNet; we explore other choices in Section 4.5. For training datasets, we utilize ImageNet (ILSVRC 2012).

4.2 Efficiency

We first evaluate the efficiency of the proposed QuiltNet when the models are split into two parts and executed on different accelerator boards. Figure 5 shows the improvements of QuiltNet in performance and energy efficiency as compared to the baseline implementation, which runs on multiple accelerators communicating original-sized tensors without using the data projection layer. We show the results over different data reduction rates, R. Our result shows that QuiltNet achieves high efficiency even with moderate data reduction rates, e.g., 20.8% (19.2%) with R = 64 for ResNet, 30.5% (28.4%) with R = 7.5 for UNet, and 21.4% (17.5%) with R = 8 for EfficientNet in terms of performance (energy efficiency), where the data reduction rates are decided by the method described in Section 3.4 (0.8% accuracy loss on average.) We can also expect larger benefits with higher data compression rates. For example, for UNet, when using R = 120, which reduces the tensor dimension by up to 1, QuiltNet provides 34.9% performance improvement with 32.9% energy saving.

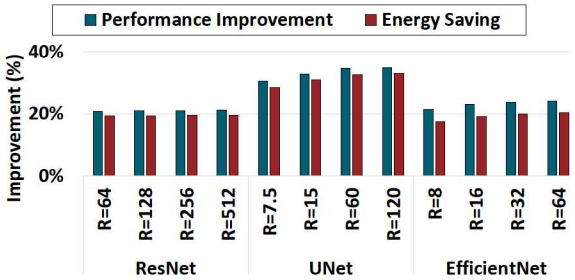


Figure 5: Energy saving and performance improvements

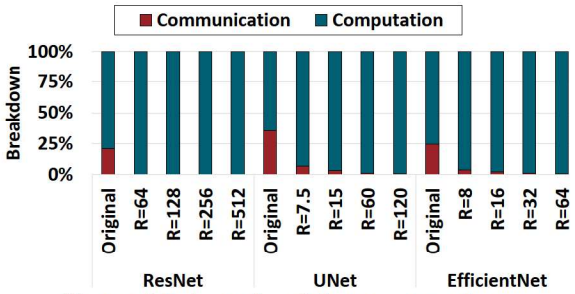


Figure 6: Latency breakdown comparison

4.3 Computation and Communication Tradeoff

To better illustrate where the benefit comes from, we measured the latency for the computation and communication. Figure 6 shows the breakdown results for the same setting. The results show that the proposed method effectively reduces the communication costs by projecting the communicated tensors into a low dimension. QuiltNet makes the computation costs dominant over the communication, resulting in much higher utilization for the deep learning accelerators. Note that the computation costs include the time to process the inserted autoencoders as extra overheads; however, in our evaluation, the overheads are very negligible, e.g., less than 0.4% of the total computation latency for all the tested cases.

4.4 Accuracy

Training from Scratch: When given training data are large enough, the QuiltNet model could be trained from scratch. Table 1 shows the validation error of ResNet50-Quilt for the ImageNet dataset. We use R = 64 which shows the best accuracy-efficiency tradeoff as discussed in Section 3.4. The result shows ResNet50-Quilt 64x trained from scratch lost only 1.69% and 0.97% for Top-1 and Top-5 error respectively, as compared to the pretrained ResNet50.

Table 1: ResNet trained from scratch for ImageNet

	Top-1 error	Top-5 error
Original ResNet50	23.85%	7.13%
ResNet50-Quilt (from scratch)	25.54%	8.10%

Table 2: Impact of weight freezing on model accuracy

	ResNet50 -Quilt 64x	UNet -Quilt 120x	EfficientNet -Quilt 64x
wo/ Weight Freezing	95.22%	88.55%	88.59%
w/ Weight Freezing	95.62%	92.66%	90.52%

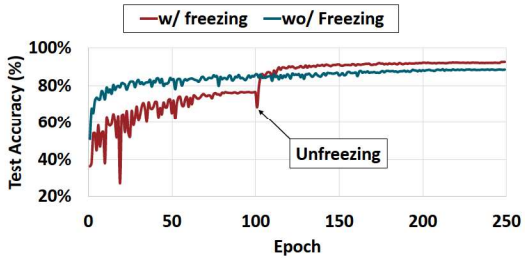


Figure 7: Impact of weight freezing for UNet

Retraining for Transfer Learning: To achieve high classification accuracies when retraining models for transfer learning, it is essential to train models utilizing the pretrained weights without catastrophic forgetting as discussed in Section 3.1. QuiltNet addressed this issue using the technique called the weight freezing. We simulate the transfer learning scenarios by using the model pretrained with ImageNet and retrained with Cifar-10 [21] to insert the autoencoder. Table 2 shows the impact of the weight freezing technique. We observe that for all the models, the weight freezing helps QuiltNet address the catastrophic forgetting issue. For example, the weight freezing allows training UNet with 4.1% higher accuracy. Figure 7 shows the accuracy changes over training epochs for UNet. We observe that the weight freezing starts training the autoencoder carefully at the beginning of the training epoch. Consequently, the autoencoder has produced useful information by mitigating catastrophic forgetting through the weight freezing. Once trained, it unfreezes the weights of other layers and eventually achieves more accurate results at the end of training.

4.5 Interposition Exploration

Different acceleration environments would require different autoencoder interposition policies for optimal performance. For example, when deploying heterogeneous accelerators, we may need to split the networks with different sizes of computational workloads. In addition, we may operate multiple accelerators (more than two), requiring the insertion of multiple autoencoders. Table 3 shows the exploration results for ResNet50 trained with Cifar-10. The results show that QuiltNet can provide a relatively low accuracy loss regardless of different inserted positions when using a single autoencoder with R = 64. Even in the worst case, the accuracy loss is 0.47% when inserting the autoencoder between stage 3 and 4. We also tested the case (Multi-AE) that inserts three autoencoders for every stage, splitting the model into four pieces. For this case, QuiltNet also creates an accurate model, resulting in 1.44× performance improvement.

5 Conclusion

In this paper, we proposed QuiltNet, which enables significant reduction in accelerator-to-accelerator communication costs for scalable deep learning process. The proposed solution intelligently divides the deep learning model into multiple pieces while adding data projection layers that reduce the amount of communicated data. We show how to train and place the inserted data projection layer while balancing the efficiency-accuracy tradeoff. The experimental results show that the proposed technique significantly saves performance and energy efficiency by up to 30.5% and 28.4% with minimal accuracy loss.

Table 3: Interposition exploration

		Accuracy	Normalized Perf.
Original ResNet wo/ autoencoder		96.05%	1.00
R = 64	Between stage 1 & 2	95.91%	1.22
	Between stage 2 & 3	95.62%	1.26
	Between stage 3 & 4	95.58%	1.31
Multi-AE		95.46%	1.44

6 Acknowledgement

This work was supported in part by SK hynix. This work also supported by the National Research Foundation (NRF) of Korea (NRF-2018R1A5A1060031). This work was also partially supported by Semiconductor Research Corporation (SRC) Task No. 2988.001.

References

[1] Szymon Migacz. Nvidia 8 bit inference width tensorrt. In *GPU Technology Conference*, volume 10, 2017.
[2] Yunji Chen, et al. Diannao family: energy-efficient hardware accelerators for machine learning. *Communications of the ACM*, 59(11):105–112, 2016.
[3] Norman P Jouppi, et al. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th annual international symposium on computer architecture*, pages 1–12, 2017.
[4] Ahmad Shawahna, et al. Fpga-based accelerators of deep learning networks for learning and classification: A review. *IEEE Access*, 7:7823–7859, 2018.
[5] Denis Foley et al. Ultra-performance pascal gpu and nvlink interconnect. *IEEE Micro*, 37(2):7–17, 2017.
[6] Anastasia Koloskova, et al. Decentralized deep learning with arbitrary communication compression. *arXiv preprint arXiv:1907.09356*, 2019.
[7] Ganggang Dong, et al. A review of the autoencoder and its variants: A comparative perspective from target recognition in synthetic-aperture radar images. *IEEE Geoscience and Remote Sensing Magazine*, 6(3):44–68, 2018.
[8] Alex Krizhevsky. One weird trick for parallelizing convolutional neural networks. *CoRR*, abs/1404.5997, 2014.
[9] Linghao Song, et al. Hypar: Towards hybrid parallelism for deep learning accelerator array. In *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 56–68, 2019.
[10] Minjie Wang, et al. Supporting very large models using automatic dataflow graph partitioning. In *Proceedings of the Fourteenth EuroSys Conference 2019*, EuroSys '19, New York, NY, USA, 2019. Association for Computing Machinery.
[11] Linghao Song, et al. Accpar: Tensor partitioning for heterogeneous deep learning accelerators. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 342–355, 2020.
[12] Yakun Sophia Shao, et al. Simba: Scaling deep-learning inference with multi-chip-module-based architecture. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '52, page 14–27, New York, NY, USA, 2019. Association for Computing Machinery.
[13] Pierre Baldi. Autoencoders, unsupervised learning, and deep architectures. In *Proceedings of ICML workshop on unsupervised and transfer learning*, pages 37–49. JMLR Workshop and Conference Proceedings, 2012.
[14] Chuanqi Tan, et al. A survey on deep transfer learning. In *International conference on artificial neural networks*, pages 270–279. Springer, 2018.
[15] Luis Perez et al. The effectiveness of data augmentation in image classification using deep learning. *arXiv preprint arXiv:1712.04621*, 2017.
[16] Uci machine learning repository. https://archive.ics.uci.edu/.
[17] Anthony Thomas, et al. Hierarchical and distributed machine learning inference beyond the edge. In *2019 IEEE 16th International Conference on Networking, Sensing and Control (ICNSC)*, pages 18–23. IEEE, 2019.
[18] Kaiming He, et al. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.
[19] Olaf Ronneberger, et al. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.
[20] Hang Qi, et al. Paleo: A performance model for deep neural networks. In *Proceedings of the International Conference on Learning Representations*, 2017.
[21] Alex Krizhevsky, et al. Learning multiple layers of features from tiny images. 2009.